Softw Syst Model (2019) 18:1379–1418
https://doi.org/10.1007/s10270-017-0609-6

REGULAR PAPER

Uncertainty-Wise Cyber-Physical System test modeling

Man Zhang[1] · **Shaukat Ali**[1] · **Tao Yue**[1,2] · **Roland Norgren**[3] · **Oscar Okariz**[4]

Received: 3 July 2016 / Revised: 13 March 2017 / Accepted: 9 July 2017 / Published online: 25 July 2017

Abstract It is important that a Cyber-Physical System (CPS) with uncertainty in its behavior caused by its unpredictable operating environment, to ensure its reliable operation. One method to ensure that the CPS will handle such uncertainty during its operation is by testing the CPS with model-based testing (MBT) techniques. However, existing MBT techniques do not explicitly capture uncertainty in test ready models, i.e., capturing the uncertain expected behavior of a CPS in the presence of environment uncertainty. To fill this gap, we present an *Uncertainty-Wise* test-modeling framework, named as *UncerTum*, to create test ready models to support MBT of CPSs facing uncertainty. *UncerTum* relies on the definition of a UML profile [the UML Uncertainty Profile (*UUP*)] and a set of UML Model Libraries extending the UML profile for Modeling and Analysis of Real-Time and Embedded Systems (MARTE). *UncerTum* also benefits from the UML Testing Profile V.2 to support standard-based MBT. *UncerTum* was evaluated with two industrial CPS case studies, one real-world case study, and one open-source CPS case study from the following four perspectives: (1) *Completeness* and *Coverage* of the profiles and Model Libraries in terms of concepts defined in their underlying uncertainty conceptual model for CPSs, i.e., U-Model and MARTE, (2) *Effort* required to model uncertainty with *UncerTum*, and (3) *Correctness* of the developed test ready models, which was assessed via model execution. Based on the evaluation, we can conclude that we were successful in modeling all the uncertainties identified in the four case studies, which gives us an indication that *UncerTum* is sufficiently complete. In terms of modeling effort, we concluded that on average *UncerTum* requires 18.5% more time to apply stereotypes from *UUP* on test ready models.

Keywords Uncertainty · Cyber-Physical System · UML · Model-based testing

Communicated by Dr. Bernhard Schaetz and Dr. Jeff Gray.

✉ Man Zhang
manzhang@simula.no

Shaukat Ali
shaukat@simula.no

Tao Yue
tao@simula.no

Roland Norgren
roland.norgren@fpx.se

Oscar Okariz
ookariz@manutencion.ulma.es

[1] Simula Research Laboratory, Oslo, Norway

[2] University of Oslo, Oslo, Norway

[3] Future Position X, Gävle, Sweden

[4] ULMA Handling Systems, Oñati, Spain

1 Introduction

"Cyber-Physical Systems (CPS) are integrations of computation, networking, and physical processes. Embedded computers and networks monitor and control the physical processes, with feedback loops where physical processes affect computations and vice versa" [1]. These systems often function in the unpredictable physical environment, leaving them vulnerable to uncertainty during their operation [2–4]. CPSs are often designed and developed with known assumptions on their operating physical environment, which may not hold during their operation. Currently, a common practice is to develop CPSs by integrating physical units without knowing their internals. Consequently, even during testing, assumptions about the expected behavior of CPSs and their

operating environment are often made. Thus, we argue that when applying model-based testing (MBT), uncertainty (i.e., "lack of knowledge" [5,6] about the internal behavior of a CPS and its composed physical units, and its operating environment) must be explicitly captured in test ready models, i.e., the models representing the expected behavior of the CPS being tested and are detailed enough such that test cases can be generated from them. We took a *subjective* approach to capture uncertainty since a test modeler(s) creates test ready models, during which assumptions are made by the modeler(s) about the internal behavior of a CPS and its physical units, and its operating environment, based on her/his (their) belief at the time the models are created.

Uncertainty in the context of CPSs is an immature area of research in general, and several efforts have just begun to study uncertainty in CPSs [7]. In this paper, we report one such effort, where we aim to devise a set of modeling methodologies for explicitly modeling test ready models (with uncertainty) for CPSs under test with the ultimate aim of automatically generating test cases from test ready models with MBT techniques. We report an *Uncertainty-Wise* Modeling Framework, named as *UncerTum* (Fig. 1), which is developed as part of an EU project [8]. The project has various types of partners contributing to the overall approach such as researchers, use case providers, tool vendor, and test bed providers, as shown in Fig. 1. *UncerTum*, developed by researchers, supports modeling test ready models with known uncertainty based on uncertainty test requirements provided by use case providers (Fig. 1). In the project, the first use case provider is Future Position X, Sweden [9], who provides the CPS case study about GeoSports (GS) from the healthcare domain, whereas the second use case provider is ULMA Handling Systems [10] who provides case study about Automated Warehouse (AW) from the logistics domain.

The core of *UncerTum* is the UML Uncertainty Profile (*UUP*) (Fig. 1), which is defined based on the uncertainty conceptual model for CPSs (*U-Model*) [7]. The *UUP* profile consists of three parts (i.e., *Belief*, *Uncertainty*, and *Measurement* profiles) and an internal library containing enumerations required in the profiles. In addition, *UncerTum* also defines an extensive set of UML Model Libraries (Model Libraries in Fig. 1) by either extending the UML profile for Modeling and Analysis of Real-Time and Embedded Systems (MARTE) [11] or defining new ones that were not covered by existing standards. The key libraries include *Uncertainty Pattern Library, Measure Library,* and *Time Library*. Moreover, *UncerTum* relies on the UML Testing Profile (*UTP*) V.2 to model test ready models for the purpose of enabling MBT. Last, *UncerTum* includes a set of guidelines (Fig. 1) with recommendations and alternative scenarios for applying the proposed modeling notations.

UncerTum was deployed on IBM Rational Software Architect (RSA) [12] as shown in Fig. 1. Once test ready models are created, they are inputted into the CertifyIt [13] MBT tool, which is a plugin to IBM RSA. With this tool, a set of executable test cases can be generated based on various test strategies that are devised and prototyped by researchers. Both the implementation of *UncerTum* and test case generation strategies will be integrated into CertifyIt by the tool vendor (EGM [14]). Finally, test bed providers provide facilities to execute generated test cases on the provided CPSs case studies. This includes *Test Infrastructure* (physical infrastructures and test emulators/simulators) and *Test APIs* to control and monitor both the test infrastructure itself and the CPS being tested. In the context of the *U*-Test project, Nordic Med Test [15] (NMT) provides the facility to execute test cases on GS, whereas in the case of AW, ULMA [10], and IK4-Ikerlan [16] provide the corresponding facility. Finally, the tool vendor implements the *Test Case Execution Platform*, which executes test cases on the CPS (Fig. 1). Note that the focus of this paper is only on *UncerTum*, which is indicated by a dashed line box of Fig. 1 (i.e., "Scope of the paper") and the rest is ongoing.

UncerTum was evaluated with two industrial case studies, one real world, and one open-source case study from the literature. The first two case studies are GS and AW available to us as part of the project, whereas the third case study is embedded Videoconferencing Systems (VCSs) developed by Cisco, Norway [17], and was used in the second author's previous work [18]. Currently, we have access to several VCSs in our research laboratory due to our long-term collaboration with Cisco and we modeled them ourselves for the purpose of evaluating *UncerTum*. Thus, this case study is a real case study, but using it to evaluate *UncerTum* is not performed in a real industrial setting. The GS and AW case studies were however performed in real industrial settings. The fourth case study (SafeHome) is an open-source case study from [19], and we extended it for our purpose. With these case studies, we performed evaluation from these three perspectives: (1) *Completeness* and *Coverage* of *UUP*/Model Libraries to U-Model and MARTE, (2) *Effort* required to model uncertainty using *UncerTum* in terms of the number of model elements and effort measured in terms of time, and (3) *Correctness* of the developed models by executing the models.

In our previous work, we developed a generic conceptual model (called U-Model) to understand uncertainty independent of its final use [7]. Notice that to keep the paper self-contained, we have provided U-Model and definitions of its concepts in "Appendix A" and we refer to it when necessary. In this paper, U-Model was implemented as *UUP*, i.e., one of the key contributions of this paper, to enable the development of test ready models for supporting MBT. Other contributions include a set of Model Libraries to model (partially extending MARTE), for example, various types of uncertainties and their measures and a set of precise guidelines to create test ready models using *UUP*, UTP V.2 and Model Libraries.

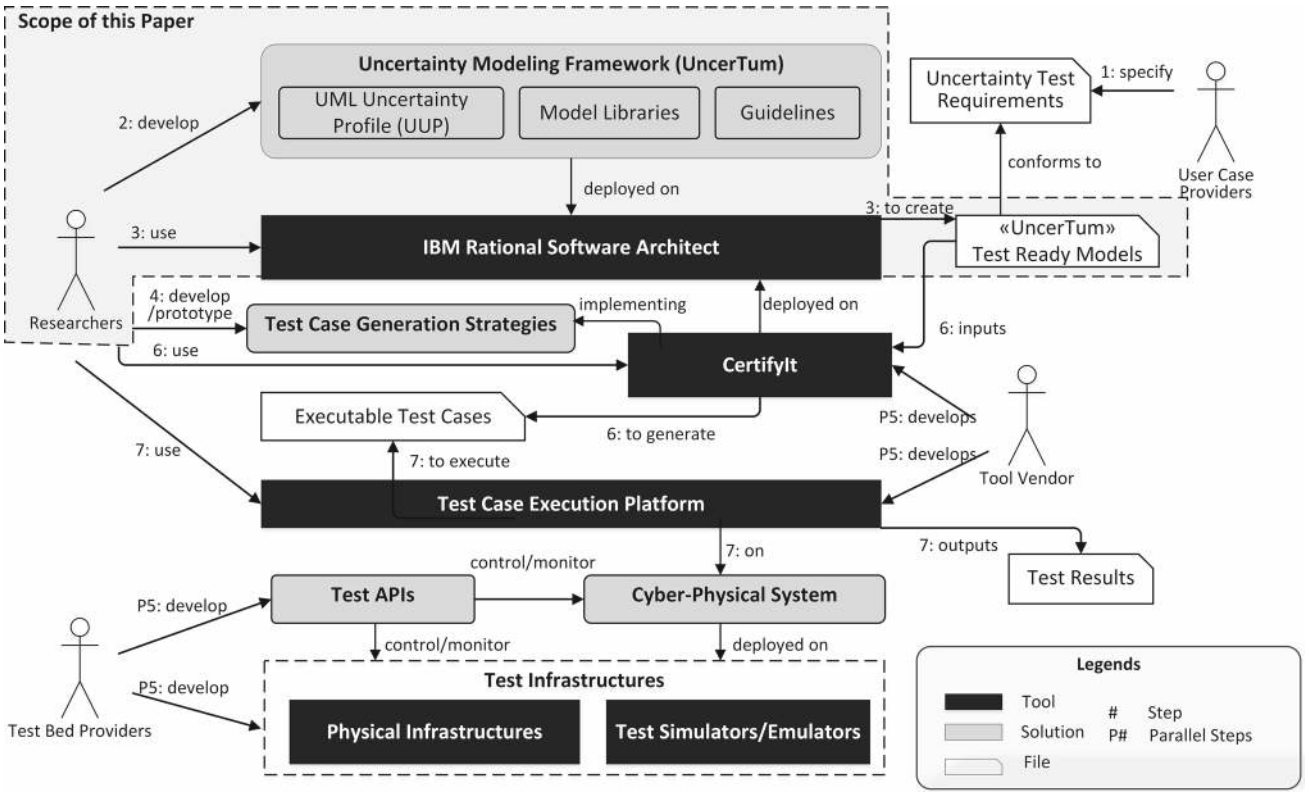


Fig. 1 Overall workflow of the *U*-Test EU project

Note that the development of UTP V.2 is not a contribution of this paper; rather its application to create test ready models with uncertainty is one of our contributions. Notice that this is one of the first papers reporting the application of UTP V.2 to industrial case studies. Another contribution of the paper is our modeling approach to check the correctness of test ready models through model execution. Finally, we consider the extensive evaluation of the applicability of *UncerTum* with the three real industrial case studies as a contribution as well.

The rest of the paper is organized as follows. Section 2 presents the background, followed by a running example (Sect. 3). Section 4 presents the overview of *UncerTum*. Section 5 discusses details of the *UUP* profile, and Sect. 6 discusses the Model Libraries. Section 7 presents the guidelines for applying *UncerTum*. Section 8 presents our modeling approach for checking the correctness of test ready models with model execution. Section 9 provides the evaluation, and Sect. 10 presents the related work. We conclude the paper in Sect. 11.

2 Background

2.1 Cyber-Physical Systems and testing levels

A CPS is defined in [7] as: "A set of heterogeneous physical units (e.g., sensors, control modules) communicating via heterogeneous networks (using networking equipment) and potentially interacting with applications deployed on cloud infrastructures and/or humans to achieve a common goal" and is conceptually shown in Fig. 2. Uncertainty can occur at the following three logical levels [7] (Fig. 2): (1) *Application level*, due to events/data originating from an application (one or more software components) of a physical unit of the CPS; (2) *Infrastructure level*, due to data transmission via information network enabled through networking infrastructure and/or cloud infrastructure; (3) *Integration level*, due to either interactions of applications across the physical units at the application level, or interactions of physical units across the application and infrastructure levels. Notice that we chose the definition of CPS from [7] as it was defined in the context of our project and was further used to define the three levels of uncertainties in CPS that are modeled in this paper and conforms to the well-known definition in [1].

2.2 U-Model

In our previous work [7], to understand uncertainty in CPSs, we developed a conceptual model called U-Model to define uncertainty and associated concepts, and their relationships at a conceptual level. Some of the U-Model concepts were extended for supporting MBT of all the three levels of CPS under uncertainty (Sect. 2.1). U-Model was developed based on an extensive review of existing literature on uncertainty

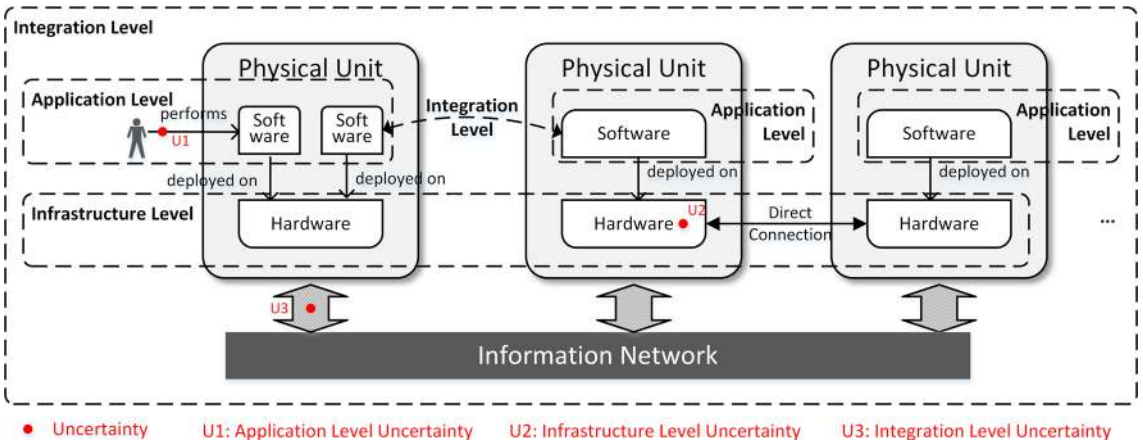


Fig. 2 Conceptual model of a Cyber-Physical System and the three levels

from several disciplines including philosophy, healthcare, and physics, and two industrial CPS case studies from the two industrial partners of the *U*-Test EU project. In this paper, we implement U-Model as *UncerTum* to support the construction of test ready models with uncertainty. Details of U-Model are given in [7], and part of U-Model is provided in "Appendix A" for the purpose of keeping this paper self-contained.

2.3 UML Testing Profile (UTP)

UML Testing Profile (UTP) [20,21] is a standard at Object Management Group (OMG) for enabling MBT. With UTP, the expected behavior of a system under test can be modeled, from where test cases can be derived. UTP can be also used to directly model test cases. Transformations from models specified with UTP to executable test cases can be performed using specialized test generators. Since UTP is defined as a UML profile, it is often applied to UML sequence, activity diagrams, and state machines for describing behaviors of a system under test or test cases. The key purpose is to introduce testing-related concepts [e.g., *Test Case*, *Test Data*, and *Test Design Model* and Model Libraries such as various types of test case *Verdict* (pass, fail)] to UML models for the purpose of enabling automated generation of test cases. UTP V.2 [21] is the latest revision of the UTP profile, which is conceptually composed of five packages of concepts: Test Analysis and Design, Test Architecture, Test Behavior, Test Data, and Test Evaluations. Various numbers of stereotypes have been defined for some concepts of these packages. Similar to other modeling notations, it is never been an objective to exhaustively apply all the stereotypes when using UTP V.2 to annotate UML models with testing concepts [21]. Which stereotypes to apply and how to apply them are however problem/purpose specific and should be defined by users of the profile. More information about the UTP V.2 standardization and the team can be found in [22,23].

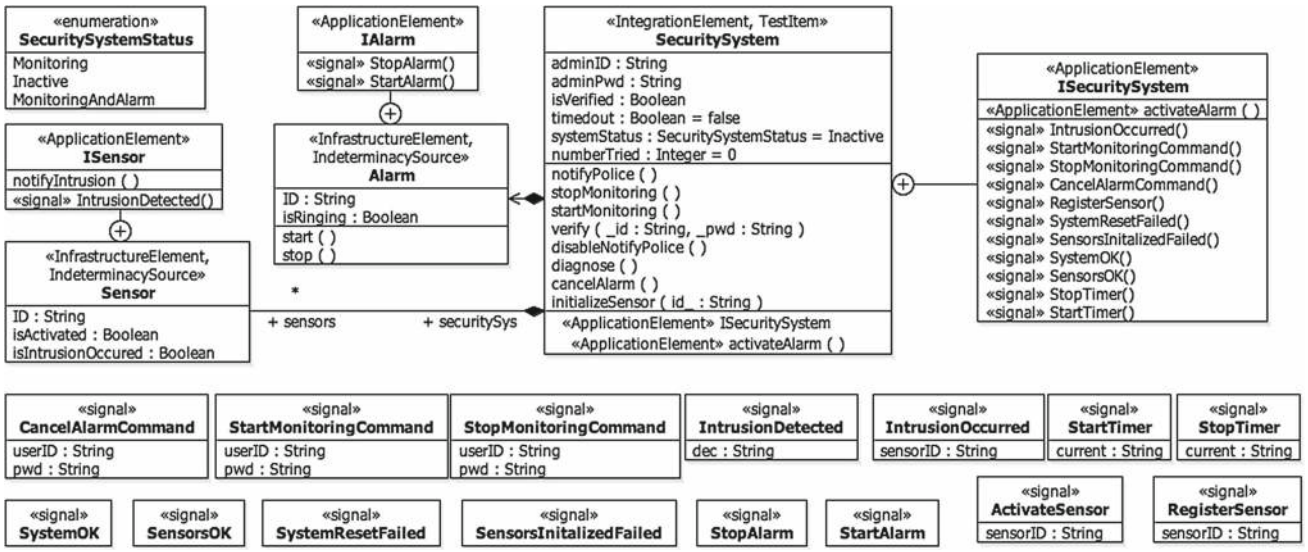
To enable MBT of CPSs under uncertainty, we rely on UTP V.2 to model the testing aspect of test ready models. In our context, only a subset of UTP V.2 was used.

3 Running example

To illustrate *UncerTum* throughout the paper, we present a running example in this section, which is a simplified security function of the SafeHome system described in [19]. The developed test ready model of the running example includes a class diagram (Fig. 3), a composite structure diagram (Fig. 4), and a set of state machines (Figs. 5, 6, 7) using IBM Rational Software Architect (RSA) 9.1 [12]. For the sake of simplicity, we only show one security function related to intrusion detection. Notice that, even though we present all the diagrams of the model of the running example in this section (including the application of the profiles and Model Libraries), we illustrate them using the running example when they are discussed in later sections.

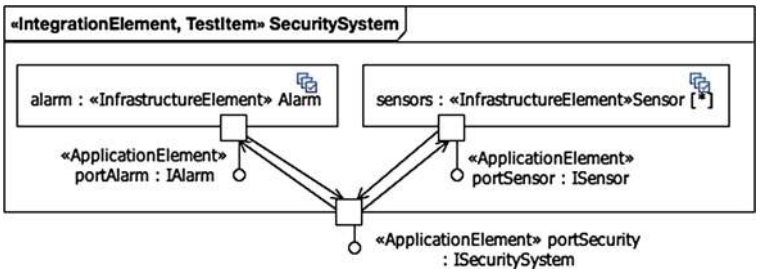
In general, the security system controls and configures *Alarm* and related *Sensor*s through their corresponding interfaces (class diagram in Fig. 3, detailed explanation in Table 1). In Fig. 4, we show a composite structure of the security system. Notice that the alarm and sensors do not talk to each other directly. Instead, they communicate via the provided interface of the port of the system: *ISecuritySystem*. For example, the security system receives the *IntrusionOccurred* signal via *portSecurity*, which is sent by a sensor from *portSensor* when an intrusion is detected (see the implementation of effect *notifyIntrusion* in Fig. 6).

Behaviors of the alarm, sensors, and the system were specified as the three state machines by the first author of the paper (modeled in Fig. 10) shown in Figs. 5, 6, and 7, respectively. The alarm state machine has two states: *AlarmDeactivated* and *AlarmActivated*. *AlarmDeactivated* represents the state that the alarm is not ringing, whereas



-- «TestItem» is from UTP V.2; «ApplicationElement», «InfrastructureElement» and «IntegrationElement» are from the *CPS Testing Level*s profile; «IndeterminacySource» is from *UUP*; Note that «enumeration» and «signal» are not stereotypes. They are used in IBM RSA to denote different types of UML model elements.

Fig. 3 Class diagram of the simplified *security system*



-- «TestItem» is from UTP V.2; «ApplicationElement», «InfrastructureElement» and «IntegrationElement» are from the *CPS Testing Level*s profile; Connectors between two ports are applied with «IntegrationElement», but IBM RSA does not visualize them in the diagram.

Fig. 4 Composite structure diagram of *security system*

the *AlarmActivated* state denotes that the alarm is ringing. The sensor state machine has two states (Fig. 6): *SensorDeactivated* denotes the state that a sensor is deactivated to detect intrusion, whereas *SensorActivated* represents that a sensor is activated to sense intrusion. The security system state machine (Fig. 7) has two concurrent regions in the composite state *MonitoringAndAlarm* and a set of sequential states (e.g., *Idle* and *Ready*). The top region (*Monitor Intrusion*) of the *MonitoringAndAlarm* composite state has two states: *Normal* and *IntrusionDetected*, which represent that an intrusion is not detected and detected, respectively. The bottom region (*Timer Control*) has three states: *Timer Stopped*, *Timer Started*, and *Police Notified*, representing the states that the timer of the system is stopped to notify the police (*TimerStopped*), the timer is activated to wait for 3 min before notifying the police (*TimerStarted*), and the police is notified (*PoliceNotified*).

These three state machines communicate via signals using the ports defined in the composite structure (Fig. 4). One typical scenario in case of security breach is: (1) When a sensor is in the state of *SensorActivated*, it sends the *IntrusionOccurred* signal to the security system (UML Action Language (UAL) [24] code in the comment in Fig. 6) once the intrusion is detected via the effect *notifyIntrusion* defined in the self-transition (Fig. 6, A) of the *SensorActivated* state; (2) when the *Security System* receives the *IntrusionOccurred* signal, it transits to the *IntrusionDetected* state from the *Normal* state (Fig. 7 B.1). In parallel, as one can see from the bottom region (*Timer Control*) of the *MonitoringAndAlarm* composite state of the system (Fig. 7), the system sends the *StartAlarm* signal to the *Alarm* state machine via *activateAlarm* (Fig. 7 and effect* in Table 1) and triggers *StartTimer()* when entering the *IntrusionDetected* state (Figs. 7 B.2, 8), which leads to the transition from *TimerStopped* to *TimerStarted* (Fig. 7). From *TimerStarted*, after 3 min (time event), the system notifies the police and transits to *PoliceNotified*; (3) the *Alarm* state machine receives the *StartAlarm* signal in the *AlarmDeactivated*

Table 1 An example of classifier *SecuritySystem* using *UncerTum*

Name	SecuritySytem (Fig 3)	
Description	The security system controls and configures *Alarm* and related *Sensor*s through their corresponding interfaces.	
Stereotype	«TestItem, IntegrationElement»	
Provided Interface	«ApplicationElement» ISecuritySystem	
Is Composed Of	«InfrastructureElement, IndeterminacySource» Sensor [*] «InfrastructureElement, IndeterminacySource» Alarm [1]	
Ports	portSecurity: «ApplicationElement» ISecuritySystem (Fig. 4) communicates with *portSensor* of *Sensor*:«ApplicationElement» ISensor communicates with *portAlarm* of *Alarm*: «ApplicationElement» IAlarm	
State Machine	«IntegrationElement, BeliefElement» SecuritySytem (Fig. 7) **Stereotypes** **agent:** «BeliefAgent» *Man_Simula* (Fig. 10)	
	Transition	«BeliefElement» *Normal*: State→*IntrusionDetected*: State (Fig. 7, B.1) **trigger***: <SignalEvent>*IntrusionOccurred(sensorID:String)* **effect*:** *activateAlarm()* the body of this operation is: portSecurity.send(new StartAlarm()) **Stereotypes** **agent:** «BeliefAgent» *Man_Simula* (Fig. 10) **measurement:** -measureInDTViaClass: «BeliefMeasure» *ReceiveIntrusionOccurred* (Fig. 9) -measurementInVS:<InstanceValue>*VeryLikely* **uncertainty:** -kind: *UncertaintyKind::Occurrence* -referredIndeterminacySourceInClassifier: «IndeterminacySource, InfrastructureElement»*Sensor* -referredCause: «BeliefElement, Cause» *notifyIntrusion* (see Fig. 6, A) -referredEffect: «BeliefElement, Effect» *AlarmActivated* (Fig. 5, C)

trigger * represents the "triggers" attribute of Transition in UML. effect * represents the "effects" attribute of Transition in UML

Fig. 5 State machine of *alarm*

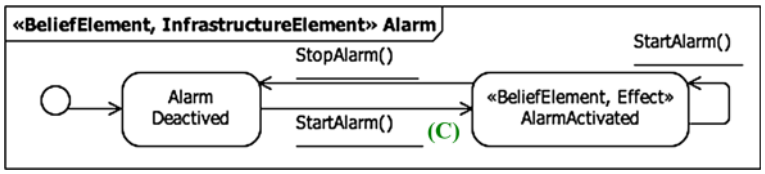
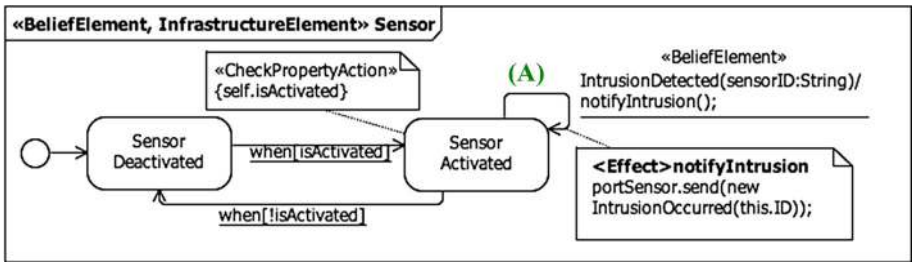


Fig. 6 State machine of *sensor*



state and activates the alarm and transits to *AlarmActivated*.

Figures 9 and 10 illustrate how we model uncertainty using *UUP/Model Libraries*, whereas Figs. 3 and 4 show the application of the CPS Testing Levels profile and UTP V.2 on the models of the running example. As an example, the detailed description for classifier *SecuritySystem* is shown in Table 1. An example of using the Model Libraries is shown in Table 1, on transition B.1, where the probability of successful intrusion detection is measured in a 7 scale of probability (*Probability_7Scale* defined in the probability library) as *VeryLikely* (see the *Transition* row in Table 1 and *Success:ReceiveIntrsionOccurred* in Fig. 9). More details are presented in the following sections.

4 Overview of UncerTum

Figure 11 shows the overall flow of using *UncerTum*, and an overview of *UncerTum* is presented in Fig. 12. Step 1 in Fig. 11 is about creating test ready models using the UML profiles (e.g., *UUP*), Model Libraries, and guidelines defined

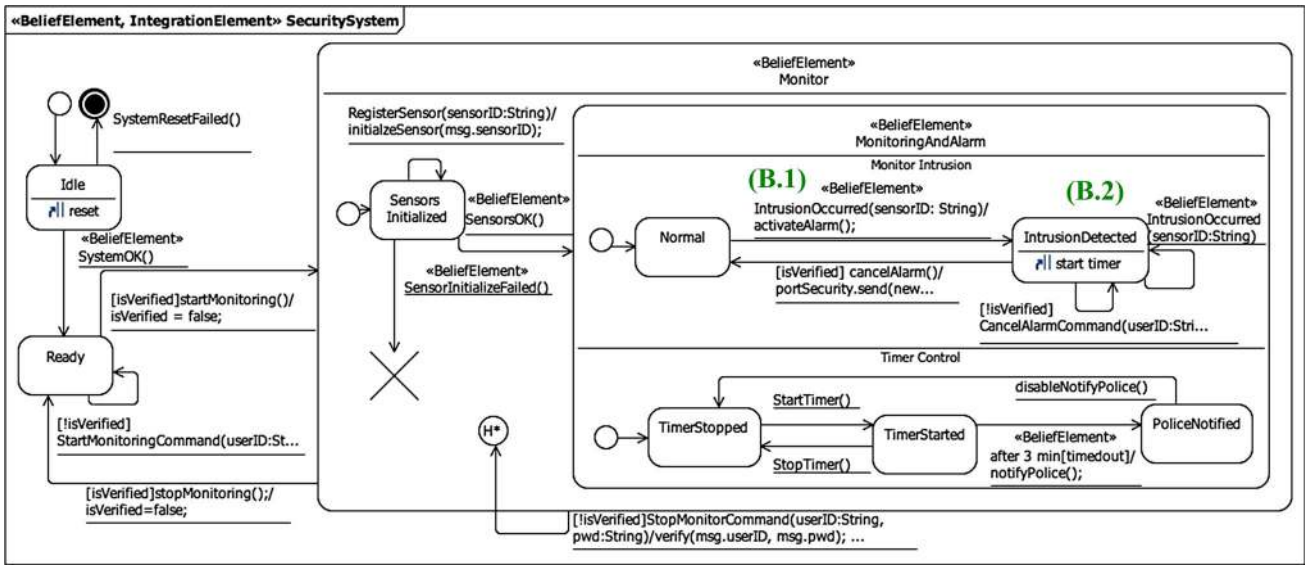


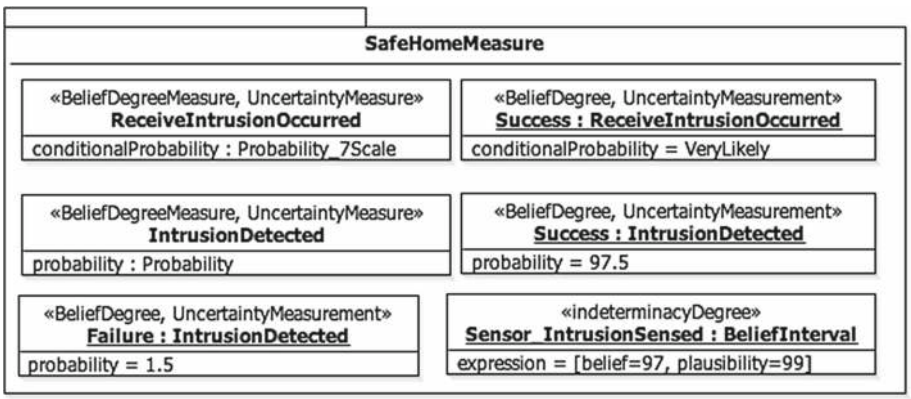
Fig. 7 State machine of security system

StateInvariant «CheckPropertyAction» of *IntrusionDetected*

```
(self.systemStatus = SecuritySystemStatus::Monitoring or (self.systemStatus = SecuritySystemStatus::
MonitoringAndAlarm and self.alarm.isRinging)) and self.sensors->forAll (s:Sensor|s.isActivated) and
self.sensors->one(s:Sensor|s.isIntrusionOccured)
```

Fig. 8 StateInvariant (in OCL) of *IntrusionDetected* (B.2)

Fig. 9 The example of modeling measurement/measure



-- Probability, Probability_7Scale and BeliefInterval are from the Measure library

Fig. 10 The example of modeling BeliefAgent

in *UncerTum*. Section 5 presents the profiles in detail, Sect. 6 presents the Model Libraries, whereas Sect. 7 presents the guidelines. Step 2 in Fig. 11 involves developing executable test ready models to support validation of these models based on the guidelines defined in Sect. 8.1. Step 3 executes these executable test ready models and in case errors are found a test modeler can use our guidelines defined in Sect. 8.2 to fix these errors. Notice that our framework only supports test modeling, i.e., creating test ready models that can be used to generate executable test cases. Such type of modeling is less detailed as compared to, e.g., modeling for automated code

generation. This is mainly because, during test modeling, we are only interested in modeling test interfaces (e.g., APIs to send a stimulus to the system and capturing state variables) and the expected behavior of a system.

The core of *UncerTum* is the implementation of U-Model (relevant part of U-Model in "Appendix A" and complete details in [7]) as *UUP* (Fig. 12). Notice that U-Model was used as the reference model to create *UUP*. While developing *UUP* based on U-Model, we took three types of decisions: (1) Some concepts from U-Model (e.g., *Uncertainty*) were incorporated into *UUP* as it is; (2) some concepts from U-Model (e.g., *Belief*, which is an abstract concept) were not implemented in *UUP*; (3) some concepts from U-Model were refined in *UUP*. For example, the *BeliefStatement* concept was implemented as «BeliefElement» in *UUP* since it corresponds to an explicit representation of model elements in the modeling context. At a high level, the core part of U-Model is implemented as *UUP* comprising of three parts:

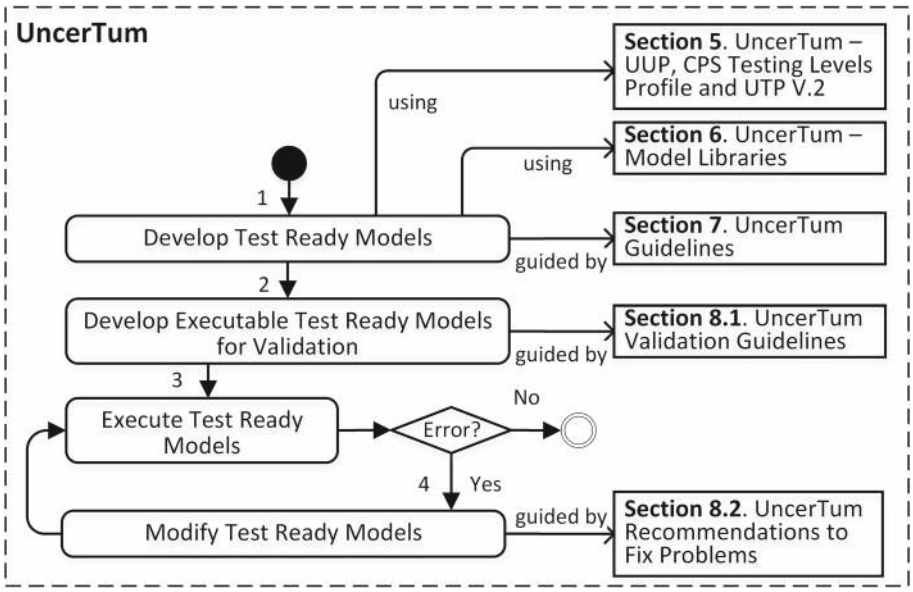


Fig. 11 Overall flow of using *UncerTum*

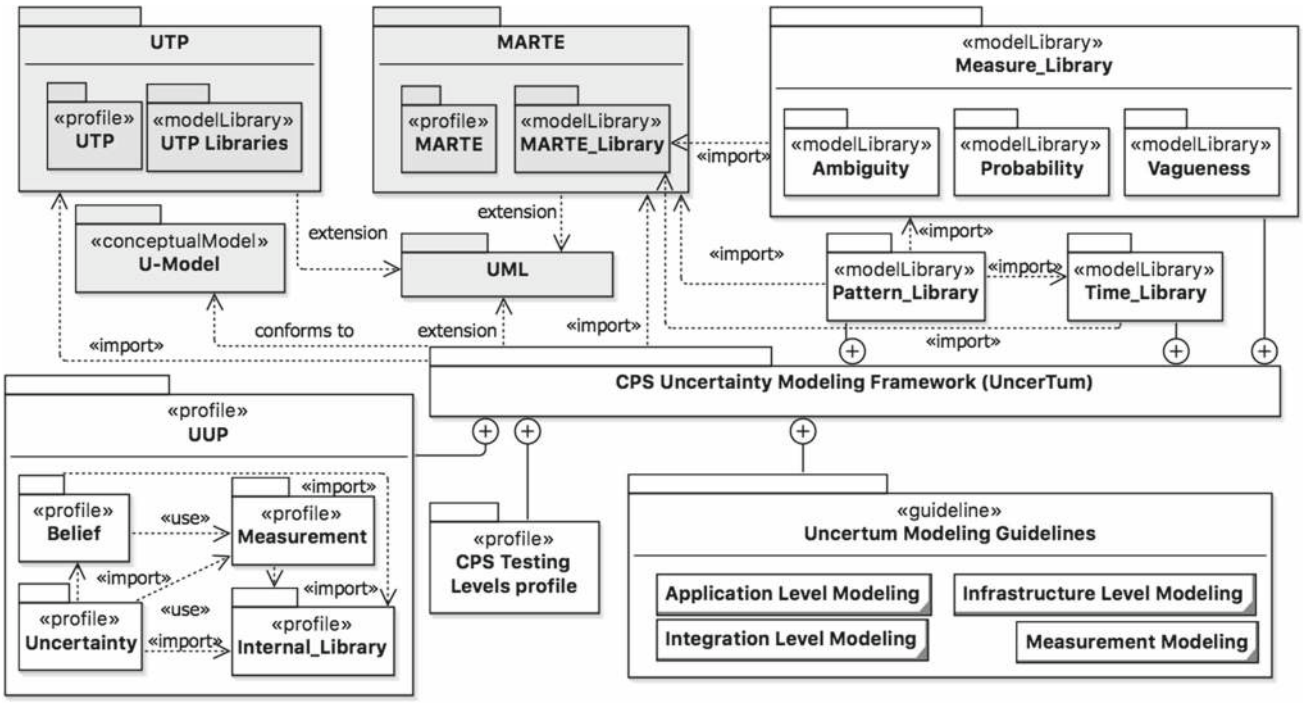


Fig. 12 Overview of *UncerTum*

Belief, *Uncertainty,* and *Measurement*. All these profiles import *Internal_Library* that define necessary enumerations required in the profiles. The framework also consists of a small *CPS Testing Levels* profile, which permits modeling specific aspects of the three testing levels of CPSs, i.e., *Application*, *Infrastructure*, and *Integration*, just for MBT.

The framework also consists of three UML Model Libraries: *Measure Library, Pattern Library*, and *Time Library* (which extend MARTE [11]). We would like to highlight that we imported *Time Library* from MARTE without any modifications and thus we will not describe it in the paper. The framework relies on UTP V.2 to bring necessary MBT concepts to test ready models. Finally, the framework provides a set of guidelines on how to use its modeling notations to construct test ready models for enabling MBT of CPSs under uncertainty.

5 *UUP* and CPS testing levels profile

This section presents *UUP*, whose modeling notations are composed of stereotypes and classes for *Belief* (Sect. 5.1), *Uncertainty*, and *Measurement* (Sect. 5.2), as shown in Figs. 13, 14, and 15. The complete profile documentation (including constraints) is provided in [25], and the mapping between concepts in *UUP* and U-Model is provided in Table 2. We also present the *CPS Testing Levels* profile in Sect. 5.3. Notice that in this section, we describe the *UUP* concepts at a high level and please refer to definitions of the U-Model concepts in "Appendix A" and the detailed profile specification in [25].

5.1 *UUP* belief

The *Belief* part of *UUP* is one of the key components of *UUP* since we focus on *subjective* uncertainty ("lack of knowledge"), where a test modeler(s) (*BeliefAgent*(s), see "Appendix A") creates test ready models of a CPS based on her/his/their assumptions (*Belief*, see "Appendix A") about the expected behavior of the CPS and its operating environment. The *Belief* part of *UUP* thus defines concrete concepts to model *Belief* of test modelers with UML. As shown in Fig. 13, it implements the high-level concepts defined in U-Model:BeliefModel provided in "Appendix A.1" (the mapping is provided in Table 2, and further discussion is provided in Sect. 9.2.1). As shown in Fig. 13 and Table 2, five stereotypes are defined, among which «BeliefElement» is the key stereotype associated with various UML metaclasses. This stereotype is used to signify which UML model elements are representing belief of belief agent(s). Generally speaking, any model element may represent a belief, but in the context of our work, we only extend UML metaclasses that are required to support MBT. For example, a *StateMachine* (subtype of metaclass *Behavior*) itself can be a belief element (i.e., expected state-based behavior of a CPS and its operating environment), such that «BeliefElement » can be applied on it to characterize it with additional information such as to which extent a test modeler (stereotyped with «BeliefAgent») is confident about the state machine (i.e., *beliefDegree* of *BeliefStatement*), all uncertainties (i.e., *Uncertainty*) associated with the state machine, and their *Measurement*s. In our context, we extend UML state machines; however, it is worth mentioning that «BeliefElement» can be used, for example, with activity and sequence diagrams if needed. We intentionally kept the profile generic (e.g., by making «BeliefElement» extend the UML metaclass *Behavior*) such that different MBT techniques based on different diagrams can be defined when needed.

In case that there is some evidence, e.g., existing data to support the belief, «Evidence» can be used. It is defined to capture any evidence for supporting the definition of a measurement for an uncertainty. The stereotype extends UML metaclass *Element*, implying that any UML model element (e.g., *Class*) can be used to specify evidence, e.g., as a *ValueSpecification*. Each uncertainty is also associated with a set of indeterminacy sources (definition in "Appendix A"), which can be explicitly specified using «IndeterminacySource» and classified with enumeration *IndeterminacyNature* (Fig. 13) as defined in "Appendix A."

The profile also implements OCL constraints defined in U-Model. For example, as shown in Fig. 13, each *beliefDegree* (an instance of *Measurement*) of a «BeliefStatement» must have exactly one measure associated with it, which can be specified in three different ways: a «Measure» (via attribute *measure* of *Measurement*), *DataType* (via *measureInDT*) or *Class* (via *measureInDTViaClass*). This OCL constraint is given:

```
context BeliefStatement:

  self.beliefDegree->size()>0 and self.beliefDegree->select(measurement:Measurement|measurement->size()>0)-
  >forAll(measurement:Measurement|(measurement.measureInDT->size()+measurement.measureInDTViaClass->size()=1)
  xor (not measurement.measure.oclIsUndefined()))
```

When we look at the running example, the belief agent (Fig. 10) is *Man_Simula* (stereotyped with «BeliefAgent») who defines three state machines: one for the alarm, one for the sensors, and one for the security system itself. As shown in Table 1, «BeliefElement» is applied on the *IntrusionOccurred* transition from *Normal* to *IntrusionDetected* (Fig. 7 B.1). The belief agent of this belief element is specified as

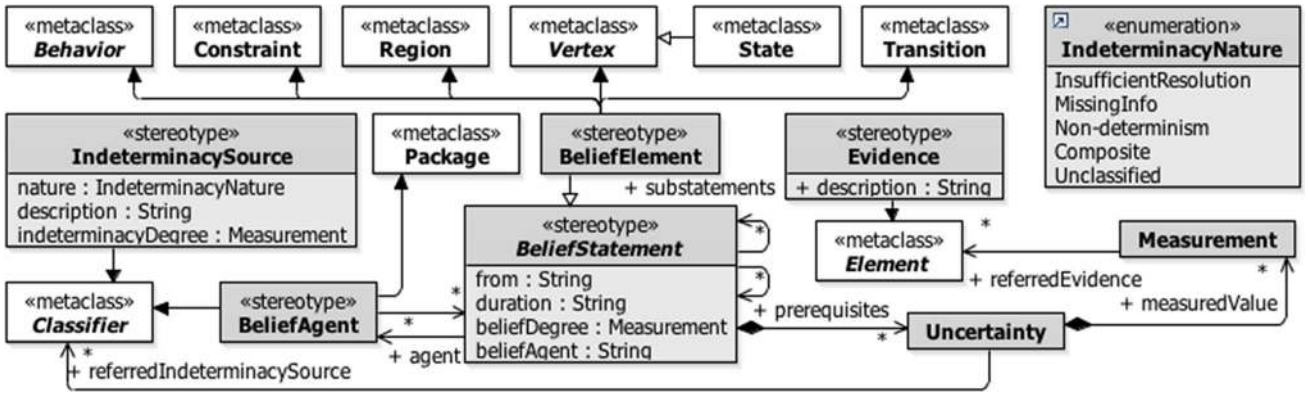


Fig. 13 The *Belief* Profile

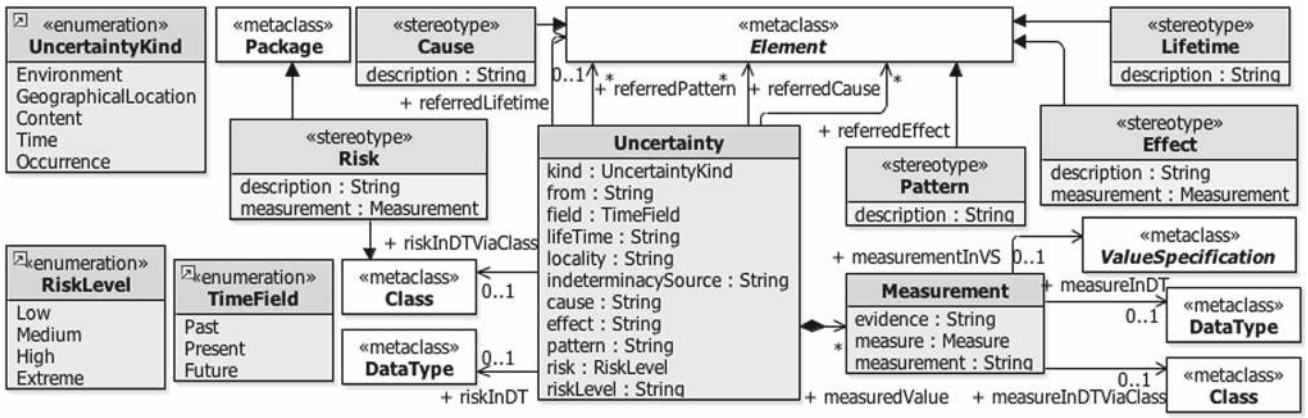


Fig. 14 The *Uncertainty* Profile

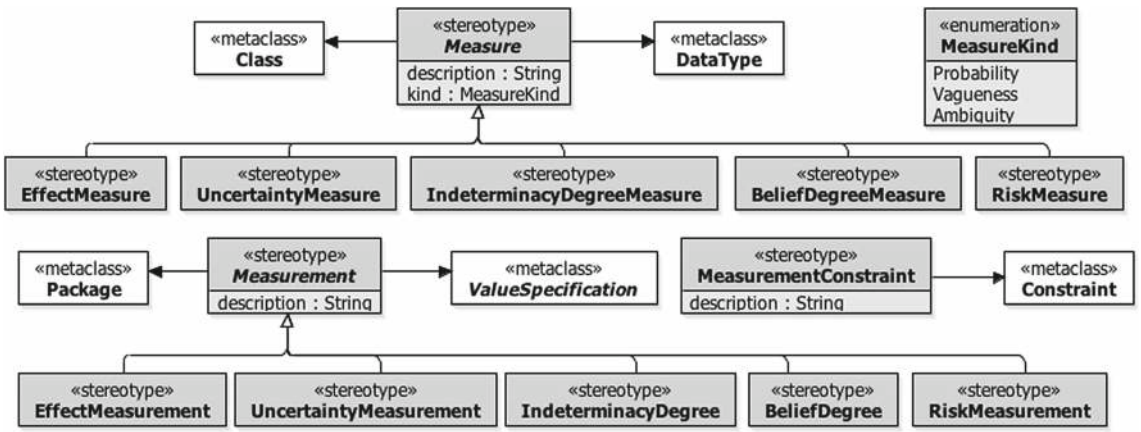


Fig. 15 The *Measurement* Profile

class *Man_Simula* (stereotyped with «BeliefAgent» shown in Fig. 10). The belief degree of this belief element is specified as a value specification «VeryLikely» and measured as *Probability_7Scale*. The belief element has one occurrence uncertainty, which is associated to «BeliefElement, Cause» *notifyIntrusion* of «IndeterminacySource» *Sensor* (Table 1).

5.2 *UUP* uncertainty and measurement

The second key component of *UUP* is about the implementation of concepts related to *Uncertainty* («BeliefElement » composed of *Uncertainty* in Fig. 14) and is presented in Fig. 14. In addition, each *Uncertainty* may have associated measurements that are captured in the *Measurement*

Table 2 Definitions of the stereotypes and classes in *UUP*

Profile	Stereotype/class in *UUP*	Concept in U-Model (Appendix A)
Belief	«BeliefStatement»	BeliefModel::BeliefStatement
	«BeliefElement»	–
	«BeliefAgent»	BeliefModel::BeliefAgent
	«IndeterminacySource»	BeliefModel::IndeterminacySource
	Uncertainty	BeliefModel::Uncertainty
	Measurement	BeliefModel::Measurement
	«Evidence»	BeliefModel::Evidence
Uncertainty	«Cause»	–
	«Effect»	UncertaintyModel::Effect
	«Lifetime»	UncertaintyModel::Lifetime
	«Risk»	UncertaintyModel::Risk
	«Pattern»	UncertaintyModel::Pattern
Measurement/Measure	«Measurement»	BeliefModel::Measurement
	«BeliefDegree»	"beliefDegree" attribute of Belief
	«Indeterminacy Degree»	"indeterminacyDegree "attribute of IndeterminacySource
	«EffectMeasurement»	"measurement" attribute of Effect
	«RiskMeasurement»	–
	«UncertaintyMeasurement»	"measuredValue" attribute of Uncertainty
	«Measure»	MeasureModel::Measure
	«BeliefDegreeMeasure»	"measure" attribute of Measurement
	«IndeterminacyDegreeMeasure»	"measure" attribute of Measurement
	«RiskMeasure»	–
	«UncertaintyMeasure»	"measure" attribute of Measurement
	«EffectMeasure»	"measure" attribute of Measurement

part as shown in Fig. 15. In Fig. 14, the key element is *Uncertainty*, which is characterized with a list of attributes such as *kind* (typed with enumeration *UncertaintyKind*) indicating a particular type of uncertainties. All of the attributes (except for *kind* and *field*) are optional. For example, an uncertainty might or might not have an indeterminacy source (i.e., *indeterminacySource* as defined in "Appendix A").

The U-Model concepts of *Effect*, *Pattern*, *Lifetime*, *and Risk* ("Appendix A") can be specified with *UUP* in difference ways. For example, one can specify the effect (i.e., the result of an uncertainty, as defined in "Appendix A") of an uncertainty simply as a string (attribute *effect* of *Uncertainty*). One can also create a UML model element and stereotype it with «Effect», and the uncertainty can then be associated with it (i.e., *referredEffect*). More details regarding the possible alternatives can be found in Sect. 7.

«IndeterminacySource», «BeliefStatement», *Uncertainty*, «Effect», and «Risk» can be further elaborated with *Measurement*. A measurement can be specified in different ways: (1) as a string (attribute *measurement* of class *Measurement*), (2) as a value specification (*measurementInVS*), (3) as a package stereotyped with a subtype of «Measurement», and (4) a constraint stereotyped with «MeasurementConstraint». «Measurement» is further classified into five subtypes, corresponding to the five types of elements to be measured.

«Measure» is defined to classify different types of measures and provide users an option to denote classes and data types with concrete measure types such as «EffectMeasure». Such a stereotyped class or data type is linked via *Measurement* to «IndeterminacySource», «Effect», *Uncertainty*, «Risk» or «BeliefStatement».

A set of OCL constraints has been implemented in *UUP*. One of the examples is that *Element* with applied «Effect» should be referred at least once via the "referredEffect" attribute of the *Uncertainty* instance:

```
context Effect:

    self.base_Element.getAppliedStereotype('UUP::Uncertainty::Effect')<>null implies

    Uncertainty.allInstances()->one(u:Uncertainty|u.referredEffect->includes(self))
```

For the running example, «BeliefElement, Effect» *ActivatedAlarm* is associated with *Uncertainty* of «BeliefStatement» *IntrusionOccurred* via the "referredEffect" attribute (Table 1).

5.3 CPS testing levels profile

We define a small *CPS Testing Levels* profile with the three stereotypes (Fig. 16) to denote which model element belongs to which level of the three: «Application», "Infrastructure," and «Integration» . This enables a test generator to use different mechanisms (if used) to control and access elements from different levels. For example, infrastructure-level variables may be accessed with different tools/APIs as compared to application-level variables. All the three stereotypes extend the UML metaclass *Element*, as one can apply them to a class, a state, a state machine, and many other model elements.

Note that this profile is defined for enabling MBT of CPS under uncertainty from the three logical levels, and we have no intention to break down CPS according to their system architectures by denoting physical units, sensors, network, etc. For example, class *Sensor* in Fig. 3 is stereotyped with «IndeterminacySource» and «InfrastructureElement», meaning that sensors are infrastructure elements. As shown in Fig. 4, the composite structure of the system describes the interactions between the infrastructure elements (*Alarm* and *Sensor*s) and the application-level elements: *portSensor*, *portAlarm,* and *portSecurity*, which are typed by three interfaces (i.e., *ISensor*, *IAlarm*, and *ISecuritySystem*) as shown in Fig. 3. This is the reason that the composite structure is stereotyped as «IntegrationElement».

6 Model Libraries

To model uncertainty with advanced modeling features, we define three Model Libraries that can be used together with *UUP* for modeling uncertainty *Pattern*s (in Fig. 20), uncertainty *Measurement*s (corresponding to *Probability*, *Vagueness*, and *Ambiguity* in Figs. 17, 18, 19), and *Time*-related properties. *Measure*, *Pattern*, and *Time* libraries import the MARTE_PrimitiveTypes library [11] to facilitate the expression of data in the domain of CPSs such as *Real*. Respectively, the *Measure* library adapts the operation of NFP_CommonType of MARTE [11] related to probability distributions. The *Pattern* library imports elements related to *Pattern* from the BasicNFP_Types library of MARTE [11] (e.g., AperiodicPattern) and further extends them. For example, the *Transient* pattern does not exist in MARTE [11] and has been newly defined. The *Time* library imports the time concepts from MARTE_DataTypes library [11] to facilitate the expression of time, e.g., *Duration* and *Frequency*, and thus does not discuss these in this paper.

6.1 Measure Libraries

We define three measure packages (*Probability*, *Ambiguity*, and *Vagueness*) to facilitate modeling with different uncertainty measures (Figs. 17, 18, 19; Table 3). Notice that in U-Model (Appendix A), these three concepts were defined only at a very high level; no breakdown of *Probability*, *Ambiguity*, and *Vagueness* was provided in U-Model. In this paper, we largely extended and implemented the detailed measures

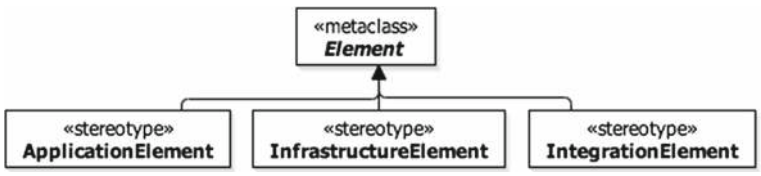


Fig. 16 The CPS testing-level profile

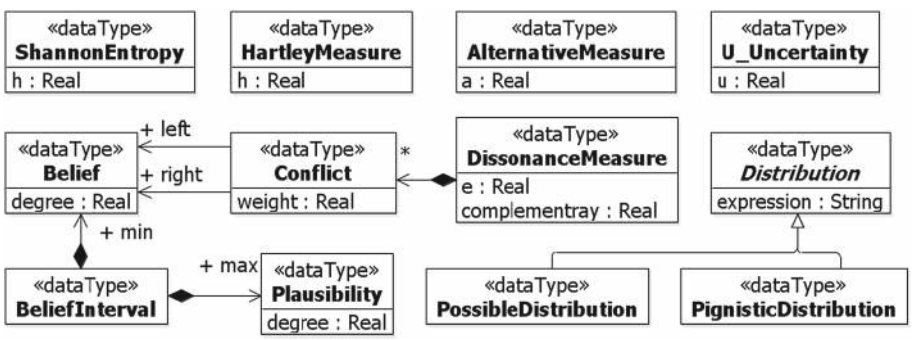


Fig. 17 The *Ambiguity* model library

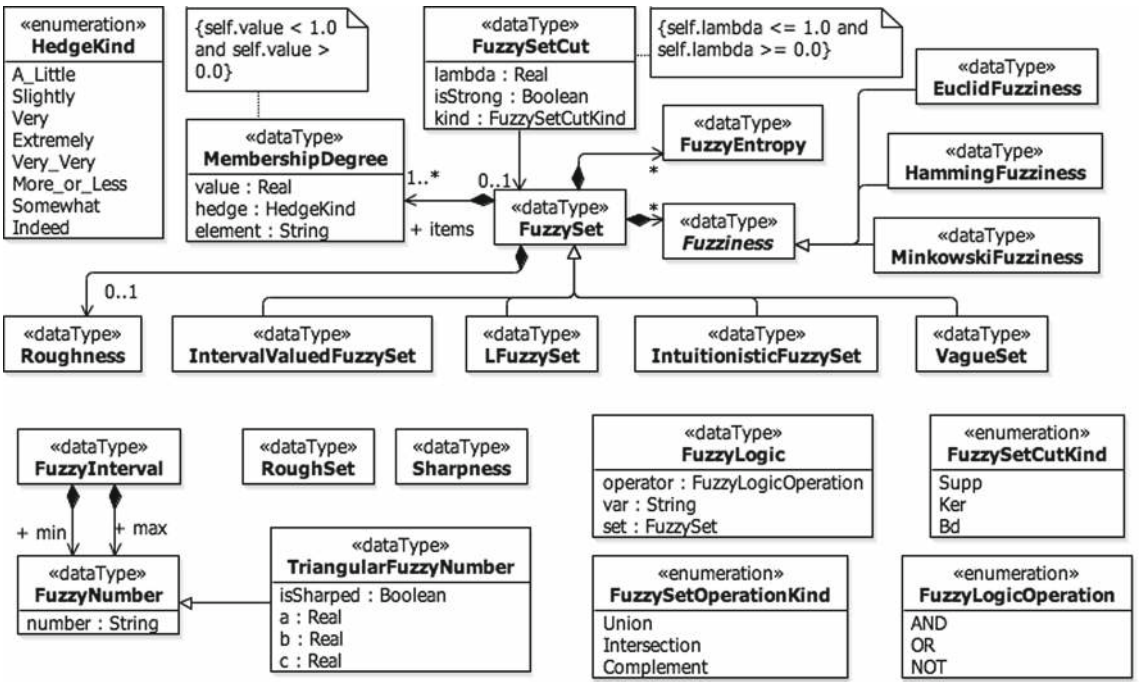


Fig. 18 The *Vagueness* model library

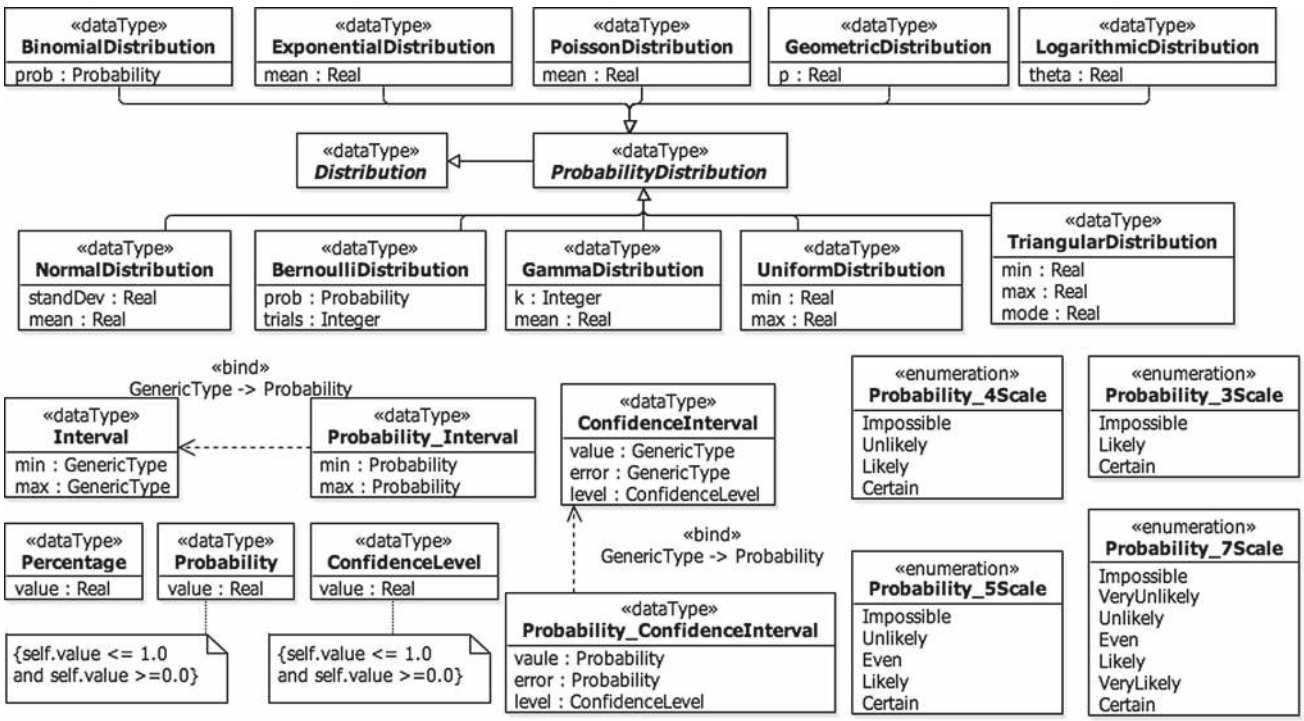


Fig. 19 The *Probability* model library

belonging to these three categories/packages, based on various theories such as Fuzzy Set and Probability Theory.

In the *Ambiguity* library, we define the data types corresponding to the relevant *Ambiguity* measures published in the literature (Fig. 17). Since these measures are well known, we do not provide further details in this paper; however, interested readers may consult the provided references listed in Table 3 for more details and the technical report corresponding to this paper [25]. The concepts of the fuzzy set theory

Table 3 The main concepts in Measure Libraries

Measure Library	Concept	References
Ambiguity	BeliefInterval, Belief, Plausibility, ShannonEntropy,	Belief theory [27]
	Conflict	[28]
	HartleyMeasure	[29]
	AlternativeMeasure	[30]
	DissonanceMeasure	[31]
	U_Uncertainty	[32]
	PossibleDistribution	[33]
	PignisticDistribution	[34]
Vagueness	MembershipDegree, FuzzySet, FuzzySetOperationKind, FuzzyLogicOperation, FuzzyLogic, FuzzyNumber	Fuzzy set and fuzzy logic theory [26]
	FuzzySetCut	[35]
	FuzzyEntropy	[36]
	Fuzziness, EuclidFuzziness, HammingFuzziness, MinkowskiFuzziness	[37,38]
	Roughness and RoughSet	[39]
	LFuzzySet	[40]
	IntuitionisticFuzzySet	[41]
	IntervalValuedFuzzySet	[42–44]
	VagueSet	[45]
	Sharpness	[46]
Probability	NormalDistribution, BernoulliDistribution, ExponentialDistribution, GammaDistribution, PoissonDistribution, UniformDistribution, GeometricDistribution, TriangularDistribution, LogarithmicDistribution	Probability distribution [47]
	Probability, ConfidenceLevel, ConfidenceInterval	[47]

[26] are defined in the *Vagueness* library (Fig. 18), and Table 3 lists the relevant references.

Various data types related to the probability are defined in the *Probability* library (Fig. 19). All the modeled probability distributions are well known, and thus we do not present further details in this paper. Some details of these distributions are provided in the technical report corresponding to this paper [25]. The other data types such as *Percentage*, *Probability*, *Probability_Interval*, and different qualitative scales of probability (e.g., *Probability_4Scale*) are from basic statistics and thus are not further explained.

For example, as shown in Fig. 9, the *IndeterminacyDegree* of *Sensor_IntrusionSensed*, which is used to measure the occurrence of successful sensing intrusion of *Sensor*, the self-transition of *SensorActivated* (Fig. 6), is expressed by *BeliefInterval* [27], which is composed of belief (97%) and plausibility (99%), which are predefined in the *Ambiguity* library (Fig. 17). Further details are provided in the technical report corresponding to this paper [25] for references.

6.2 Pattern Library

This section presents *Pattern Library* shown in Fig. 20 to support modeling various known patterns, in which an uncertainty may occur. All the patterns except for *Transient* and*PersistentPattern* are imported from MARTE [11]. Details of these patterns can be consulted from the MARTE specification and the technical report corresponding to this paper [25]. The definition of *Transient* is adopted from [7], i.e., "Transient is the situation whereby an uncertainty does not last long". *Transient* inherits from *IrregularPattern*. The newly introduced attributes are *minDuration* and *maxDu-*

ration describing the duration for which the uncertainty lasts. The definition of *PersistentPattern* is adopted from [7], i.e., "the uncertainty that lasts forever". The definition of "forever" varies. For example, an uncertainty may exist permanently until appropriate actions are taken to deal with the uncertainty. On the other hand, an uncertainty may not be able to resolve and stays forever. The *duration* attribute of *PersistentPattern* is set to "forever" to indicate that the uncertainty with this pattern stays forever until resolved. In the context of testing, "forever" may be the duration for which a test case is being executed on a CPS.

7 *UncerTum* modeling methodology

In this section, we present our modeling methodology for *UncerTum*. The rest of this section is organized as follows: Sect. 7.1 presents the overview of modeling activities, Sect. 7.2 presents modeling activities at *Application Level*, Sect. 7.3 presents modeling activities at *Infrastructure Level*, Sect. 7.4 presents modeling activities at *Integration level*, and Sect. 7.5 presents the modeling activities of *applying UUP* which is invoked at the above three levels. Notice that we used the activity diagram to provide a step-wise procedure to create test ready models and this activity diagram is not used for testing. We used structured activities in the activity diagram when it was necessary to break an activity down. Whenever an activity is used by multiple activity diagrams, we created the activity and call it from the multiple activity diagrams using call behavior activity nodes.

To facilitate the construction of test ready models, we made a set of design decisions, which are summarized, along with the rationales behind, in Table 4. We refer to them in the text whenever those are discussed.

7.1 Overview

The modeling methodology is naturally organized from the viewpoints of the three types of stakeholders: *Application* Modeler, *Infrastructure* Modeler, and *Integration* Modeler, as shown in Fig. 21. For activities performed by each type of modelers, we distinguish them by tagging each of them (in their names) using "AP", "IF", and "IT", respectively.

As shown in Fig. 21, all modelers are recommended to start from creating a package (i.e., AP1, IF1, and IT1), which is used to group and contain model elements for each respective level (DD1 in Table 4). Next, application and infrastructure modelers apply the *UUP* notations to model system behaviors of the application and infrastructure levels, respectively (i.e., AP2 and IF2). These two structured activities are further elaborated in Sects. 7.2 and 7.3. When these two activities are finished, integration modelers take their results as inputs and perform *IT2: Model Integration Behavior*. Details of this structured activity are further discussed in Sect. 7.4.

7.2 Application-level modeling

The application-level modeling activities (Fig. 22) include four sequential steps: creating application-level class diagrams (AP2.1, DD2 in Table 4), creating application-level state machines (AP2.2, DD5 in Table 4), applying CPS testing levels profile (AP2.3), and applying the *UUP* notations on the created class and state machines (AP2.4).

A class diagram (DD2) created for the application level captures application-level state variables (attributes), whose values either can be accessed directly or with dedicated APIs. We also model operations representing APIs to send stimuli to the CPS being tested. Also, it is important to mention that such a class diagram usually needs to specify *Signal*, which is a *Classifier* for specifying communication of send requests across different state machines. In addition, a class in the class diagram may receive signals from other classes (even across levels) that are modeled as signal reception (DD3/DD4 in Table 4). When creating a class diagram for the application level, for each class, each of its attributes captures an observable system attribute, which may be typed by a DataType in the *UUP*'s Model Libraries (Sect. [25]) or MARTE_Library [11]. An attribute may represent a physical observation on a device (e.g., battery status on an X4 device). Each operation of a class in a class diagram represents either an API of the application software or an action physically performed by an operator (e.g., switching on or off of an X4 device). Each signal reception represents the stimulus that can be received from a different state machine.

In a state machine (DD5-DD8 in Table 4), each state is precisely defined with an OCL constraint specifying its state invariants (DD6 in Table 4). Such an OCL constraint is constructed, based on one or more attributes of one or more classes of an application-level class diagram. Each transition in a state machine should have its trigger defined as a call event corresponding to an API or a physical action defined in the class diagrams of the application level and has its guard condition modeled as an OCL constraint on the input parameters of the transition's trigger (DD6/DD7 in Table 4).

Next, application modelers need to apply *UUP* on state machines (AP2.4) to specify uncertainties and apply the UTP profile to add testing information (e.g., indicating *TestItem*). The application of *UUP* is the same for the three levels, and thus we only describe it under the integration-level modeling section (Sect. 7.4).

7.3 Infrastructure-level modeling

For the infrastructure level, a similar modeling procedure as the one defined for the application level should be fol-

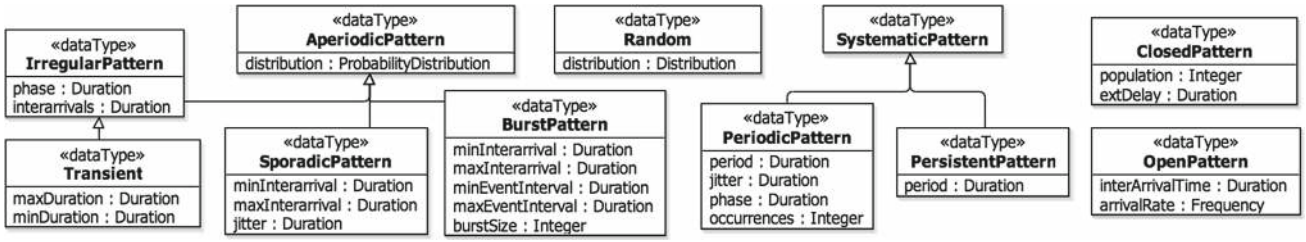


Fig. 20 Pattern Library

Table 4 Design decisions in *UncerTum* to model test ready models

#	Scope	Decision	Justification
DD1	Package	Group model elements belonging to different levels in different packages	The purpose is to enable separation of concerns, based on each logical level, e.g., application, and enable reuse of model elements
DD2	Class Diagram	Use *Class Diagram* to model the structure of a CPS	*Class diagram*s are commonly used to capture the structure of a CPS as state variables, test APIs (as operations), configuration variables, signals, and receptions
DD3	Class Diagram/Signal	Use *Signals* to facilitate sending stimulus from one physical unit to another	*Signal*s can model asynchronous communication across various physical units of a CPS, which is the purpose of UML defining signals
DD4	Class Diagram/Reception	Use signal *Reception* to model the stimulus that a physical unit can receive from another	The rationale conforms to the purpose of UML defining signal*Reception*
DD5	State Machine	Use *State Machine*s to model the expected behavior of a CPS and its operating environment with uncertainty	The reason is that a large number of CPSs exhibit state-based behaviors [48,49] In addition, we have already developed test generators to generate test cases from UML state machines [50], some part of which can be extended for testing CPSs under uncertainty when needed
DD6	State Machine/Guard, State Invariants	Specify a *State Invariant* as an OCL constraint modeling test oracles. *Guard* conditions are also specified as OCL constraints that are used to generate test data to fire triggers on transitions	OCL is a standard language for specifying constraints on UML models. Several tools for evaluating OCL constraints (e.g., Eclipse OCL [51] and Dresden OCL [52]) and solving OCL constraints (e.g., EsOCL) are available
DD7	State Machine/Transition	*Trigger*s on transitions are specified as *SignalEvent*, *CallEvent*, *TimeEvent* or *ChangeEvent*	(1) *SignalEvent* is used to facilitate communication across state machines of different physical units of a CPS; (2) *CallEvent* is used to model invocation of a testing API or manual operation to a CPS; (3) *TimeEvent* models time-related events; (4) *ChangeEvent* models changes in values of state variables. All these elements are used as they are intended in UML
DD8	State Machine/Terminate	*Terminate* is used to interrupt the *State Machine*	The purpose is to indicate the termination of the execution of a test case on a CPS
DD9	Class Diagram and State Machine	UAL is used to enable the execution of models	Our overall approach is implemented in CertifyIt, i.e., a plug-in to IBM RSA (Sect. 1). UAL [24] is implemented based on the OMG Alf standard and in IBM RSA Simulation Toolkit. Thus, we used it to fit in the overall approach

Table 4 continued

#	Scope	Decision	Justification
DD10	Composite Structure Diagram	Use *Composite Structure Diagram*s to model interactions of a CPS with outside the world and among different physical units of the CPS	In UML, *Composite Structure Diagram*s are for capturing the internal structure of a classifier, its interaction with environment or other physical units via *Port*s. Our use of composite structure diagrams conforms to UML
DD11	Composite Structure Diagram/Port, Connector	Use *Ports*/*Connector*s to model communication of a CPS with outside the world and communications across physical units of a CPS	*Ports*/*Connector*s in the UML are defined to facilitate communication in the same way as we intend
DD12	*UUP*/Belief Agent, Evidence, Lifetime, Measurement, Cause, Pattern, Effect, Risk, IndeterminacySource	Model these concepts as *String* values	It is recommended if test ready models are annotated with information describing these concepts not for enabling test generation. Doing so can help reducing modeling effort
DD13	*UUP*/Belief Agent, Evidence, Lifetime, Measurement, Cause, Pattern, Effect, Risk, IndeterminacySource	Model these concepts by applying stereotypes on model elements (e.g., classes, packages) and group them in dedicated packages	This option facilitates defining specific test strategies based on the captured information via these stereotypes. In addition, it helps to facilitate reuse of model elements

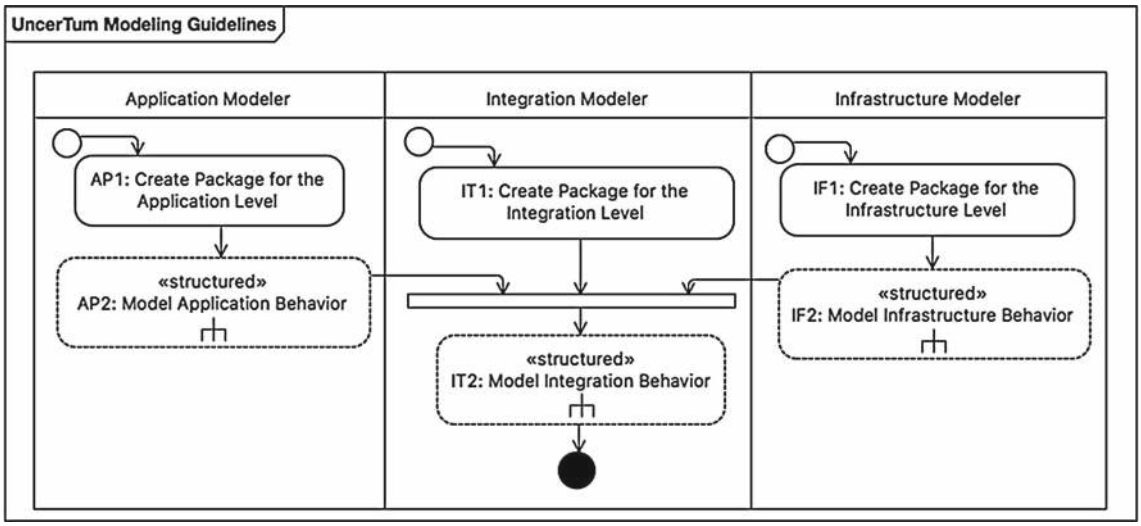


Fig. 21 The top-level guidelines

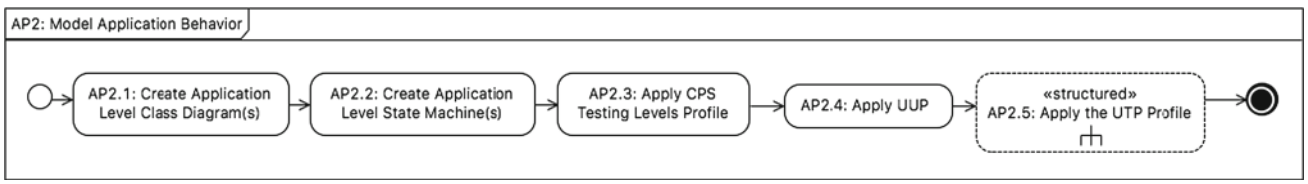


Fig. 22 Application-level guidelines

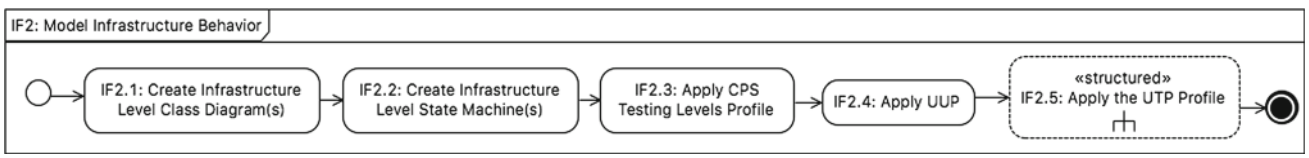


Fig. 23 Infrastructure-level guidelines

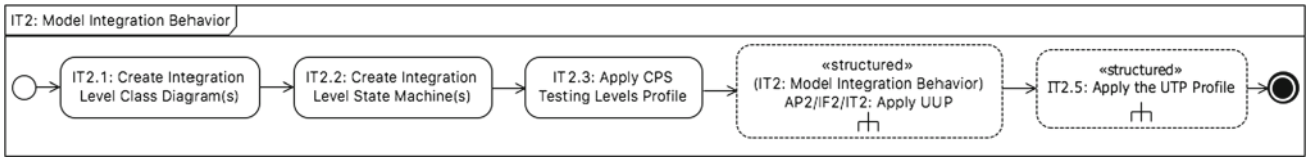


Fig. 24 Integration-level guidelines

lowed to derive class diagrams and state machines, apply *UUP* and UTP (further details in Sect. 7.4), as shown in Fig. 23. One difference is that attributes of infrastructure-level class diagrams should capture observable infrastructure attributes. For example, an attribute (*isIntrusionOccurred* in Fig. 3) can reflect the occurrence of intrusion sensed by Sensor. Operations of infrastructure-level class diagrams represent APIs for manipulating infrastructure-level components. Regarding state machines, they should be consistent with the infrastructure-level class diagrams. In other words, states should have their invariants defined as OCL constraints based on the attributes defined in the infrastructure-level class diagrams, and transitions having their triggers defined as call events or time/change events (DD5–DD7 in Table 4).

7.4 Integration-level modeling

Recall that, activity IT2 is started after class diagrams and state machines created for the application and infrastructure levels. As shown in Fig. 24, the IT2 activity starts from creating integration-level class diagrams (IT2.1) and state machines (IT2.2) and applying the CPS testing levels profile (IT2.3), followed by applying *UUP* and UTP.

Regarding creating class diagrams for the integration level, such a class diagram should focus on specifying interactions between the application software and infrastructure. Particularly, signal receptions should be defined to model events that a class can receive from the infrastructure and/or application levels (DD3–DD4). Each signal reception corresponds to an instance of UML *Signal* defined in a created integration-level class diagram (DD3–DD4). Notice that creating class diagrams for the integration level is not mandatory (DD1). Model elements that have been defined in the application and infrastructure-level class diagrams can appear in the integration-level class diagrams, and they should be specified from the perspective of integration-level modelers.

There are different ways of defining model elements for the integration level. One way is to refine the created application and infrastructure-level state machines by directly introducing new model elements to them. For example, a state in the application level can send a *Signal* to the infrastructure level and vice versa. Transitions of a state machine in the application (infrastructure) level should capture triggers of type *Signal Reception* and effects containing *Signal*s from the infrastructure (application) level. Another way is to keep application and infrastructure-level state machines untouched by applying a specific modeling methodology (e.g., Aspect Oriented Modeling methodologies) to specify crosscutting behaviors separately. In addition, one should also benefit from advanced features of UML state machines (e.g., concurrent state machines, parallel regions) to for example refer to existing state machines defined in the application and infrastructure levels.

7.5 Apply *UUP* (AP2/IF2/IT2)

Since test ready models can be created in several different ways, we propose a set of options to restrict the way, in which test modelers apply *UUP*. Notice that our test generators will only be able to generate test cases when one of these options is followed. The same modeling decisions (D12, D13 in Table 4) are taken for several concepts in *UUP* including *Belief Agent, Evidence, Measurement, Lifetime, Cause, Pattern, Effect, IndeterminacySource,* and*Risk*. All these can be simply modeled as String values. This option is informal since a test modeler is allowed to provide any String value. Second, a more formal way is to model these concepts as Fig. 10 (e.g., a Ph.D. student at Simula class for a particular *BeliefAgent*) with possible attributes and operations inside a dedicated package (e.g., for all *BeliefAgent*s for a CPS under test). Followed by this, we recommend applying dedicated stereotypes (e.g., «BeliefAgent») either on classes, package, or both. The justification of these design decisions is summarized in Table 4.

Recall that the activity of applying *UUP* is invoked at all the three levels. We tag each type of the activities of the activity diagrams from Figs. 25, 26, 27, 28, 29, 30, 31, 32, and 33 with S, C, and A to represent structured activities, call behavior, and normal activity nodes (standard semantics as in UML). Note that these activity diagrams are developed to explain the step-wise procedure to create test ready models and themselves are not part of the test ready models. As shown in Fig. 25, applying *UUP* starts from applying «BeliefElement» on any *UUP* allowed state machine model element. Then a modeler can specify values for the "from" and "duration" attributes of the stereotype, model belief agents, model belief degree, and/or model uncertainties (Fig. 25).

As shown in Fig. 26, there are two ways (D12, D13) to model belief agents (S1.1 and S1.2). A modeler can specify belief agents simply as one or more strings via the "beliefAgent" attribute of «BeliefElement» (S1.1). She/he can also

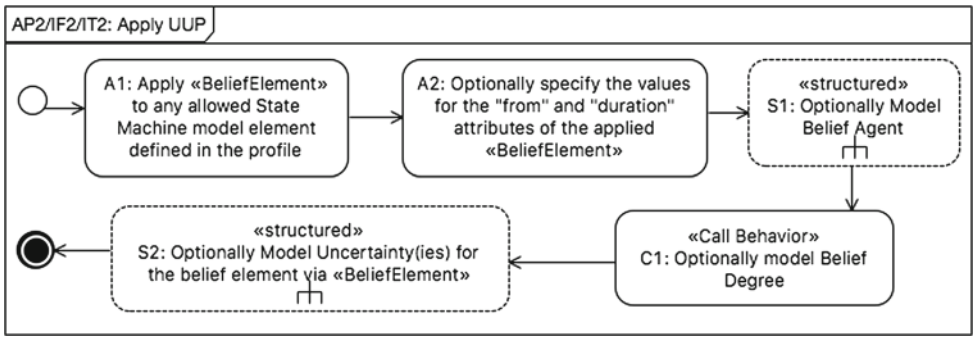


Fig. 25 Applying *UUP*

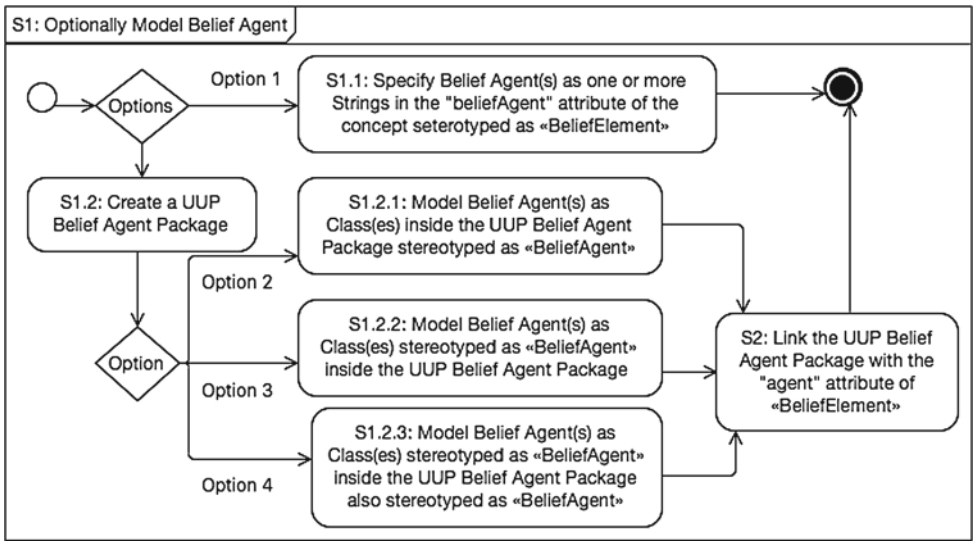


Fig. 26 Model «BeliefAgent»

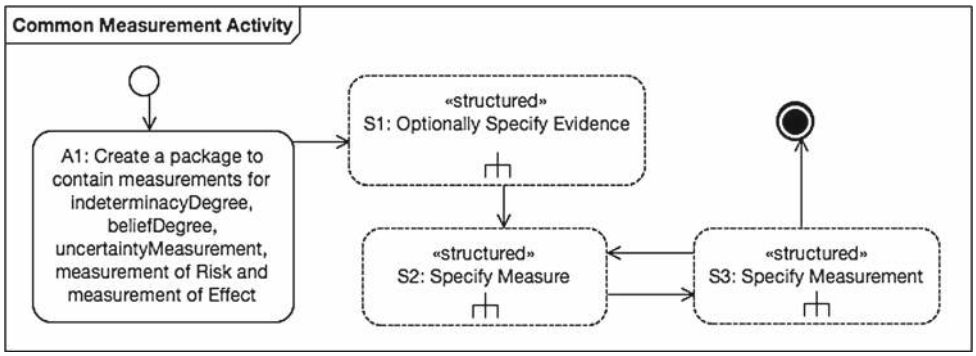


Fig. 27 Common measurement modeling activity

create a package to organize all the belief agents (S1.2). In this case, each belief agent can be modeled as a class in the package and the package is stereotyped with «BeliefAgent» . Alternatively, one can model each belief agent as a class and stereotype it with «BeliefAgent». The other option is to model each belief agent as a class and stereotype it with «BeliefAgent» and also stereotype the package with «BeliefAgent» . When choosing to apply options 2, 3, and 4, one needs to link a created belief agent package to the agent attribute of «BeliefElement» (S2). For example, we modeled the belief agent, *Man_Simula,* using *Option 3* as shown in Fig. 10.

Modeling *BeliefDegree* is presented in Sect. 7.5.1, and modeling uncertainties are discussed in Sect. 7.5.2.

7.5.1 Measurement modeling

Modeling measurements and measures are important for applying *UUP*. These activities are used to measure *beliefDegree*, *Uncertainty*, *indeterminacyDegree*, *Risk,* and *Effect*. As shown in Fig. 27, one first needs to create a package to contain measurements for *indeterminacyDegree*, *beliefDegree*, *uncertaintyMeasurement*, measurement of *Risk* and measurement of *Effect* (A1). Then, a modeler can option-

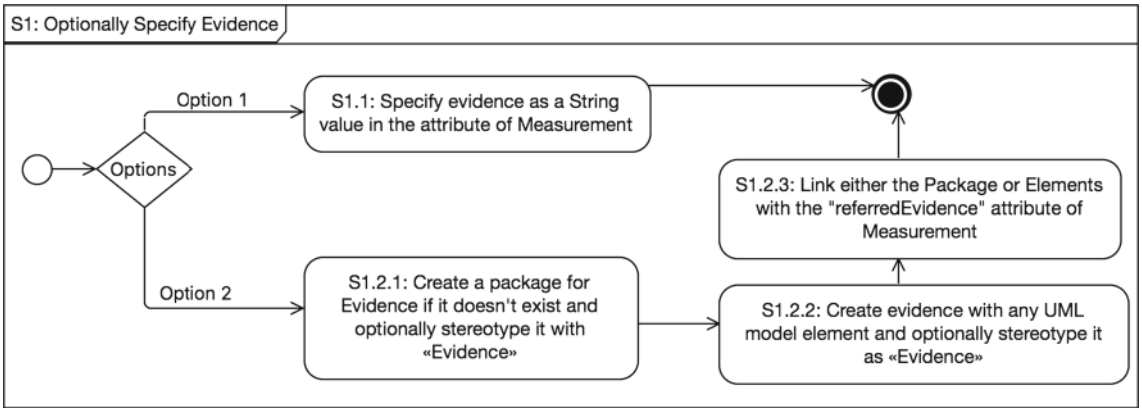


Fig. 28 Specify evidence

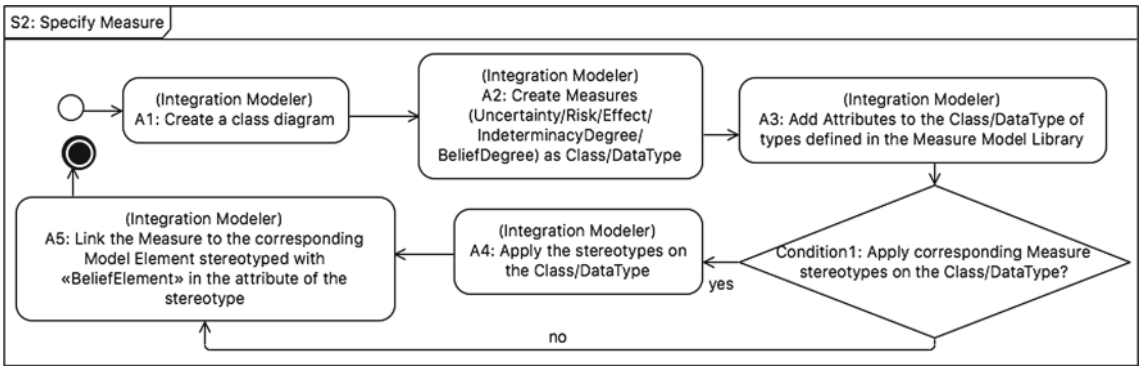
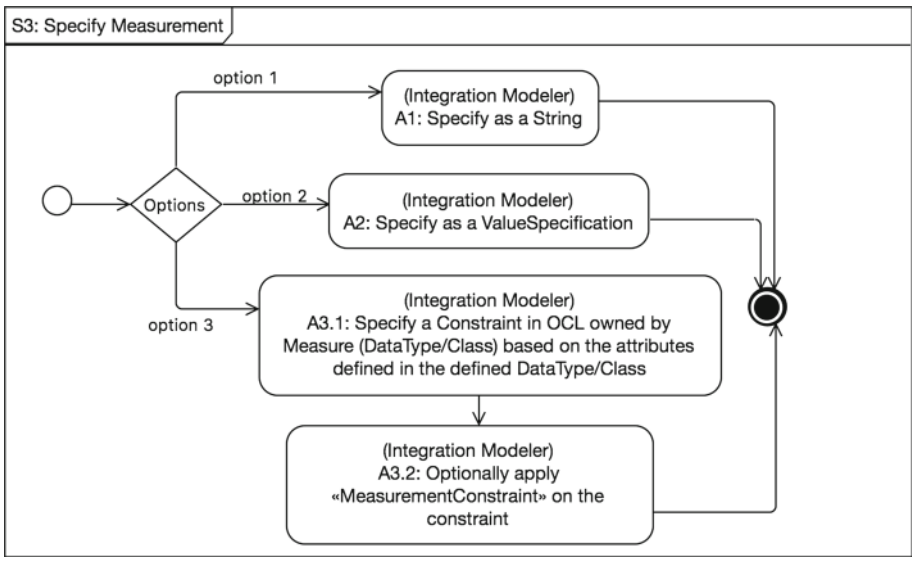


Fig. 29 Specify measure

Fig. 30 Specify measurement



ally specify *Evidence* (S1), followed by the specification of each measurement instance and its corresponding measure (S3 and S2).

A. Specify Evidence

As shown in Fig. 28, there are two ways (D12, D13) to specify evidence. Option 1 is to specify evidence as a String value (in the "measurement" attribute of *Measurement*). Option 2 is to create a package for evidence if such a package does not exist and optionally stereotype it with «Evidence» (S1.2.1). One can then create any UML model element to represent evidence, according to *UUP* and optionally stereotype it with «Evidence» (S1.2.2). The last step of Option 2 is to link either the package or UML model elements

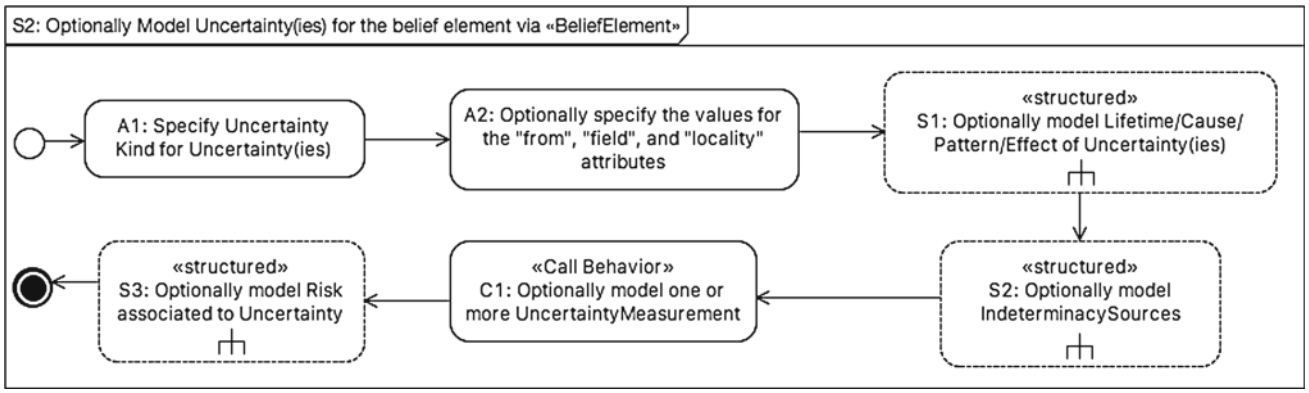


Fig. 31 Model uncertainty

representing evidence to the "referredEvidence" attribute of *Measurement* (S1.2.3).

B. Specify Measure

As shown in Fig. 29, to specify a measure, a modeler needs to create a class diagram (A1) and then create instances of *Measures* (for measurements of either "indeterminacyDegree," "beliefDegree," "uncertaintyMeasurement," measurement of *Risk* or measurement of *Effect*) as classes or data types (A2). One then needs to add attributes to these classes or data types by using the data types defined in the *Measure Libraries* (Sect. 6.1). One can optionally apply corresponding measure stereotypes (e.g., «UncertaintyMeasure») to the classes or datatypes (A4). The last step is to link a measure to an instance of *Measurement* (A5).

C. Specify Measurement

There are three ways (D12, D13) to specify measurements (in Fig. 30): specifying a measurement as a String of the measurement attribute of *Measurement* (A1), *ValueSpecification* (A2), and an OCL constraint owned by a class or datatype representing a measure, based on the attributes defined in the class or datatype (A3.1). One can also optionally apply «MeasurementConstraint» to an OCL constraint defined to specify a measurement (A3.2).

7.5.2 Uncertainty modeling

As shown in Fig. 31, one first needs to specify the kind of an uncertainty (A1), optionally specify values for attributes "from," "field," and "locality" of the uncertainty, optionally model *Lifetime* (or *Cause*, *Pattern*, *Effect*) of the uncertainty, optionally define *IndeterminacySource*(s), optionally model *uncertaintyMeasurement* and *Risk*.

A. Model Lifetime/Cause/Pattern/Effect of Uncertainty

A modeler has two options (D12, D13) to specify *Lifetime*/*Cause*/*Pattern*/*Effect* of an uncertainty, as shown in Fig. 32. One option is to simply specify an instance of these as a String value owned by the uncertainty (via attributes "lifetime," "cause," "effect," "pattern" or "risk" of *Uncertainty*). The second option needs to start from creating a package for *Lifetime/Cause/Pattern/Effect* if such a package does not exist, and optionally apply «Lifetime», «Cause», «Pattern», or «Effect» (S1.2.1). After creating packages, one needs to create *Lifetime/Cause/Pattern/Effect* as any UML model element and optionally apply the corresponding stereotypes. Since *Effect* can be measured, an instance of it can be optionally associated with one or more measurements (Sect. 7.5.1). The last step of Option 2 is to associate each created package or element to corresponding attributes of *Uncertainty*, i.e., "referredPattern," "referredEffect," "referredLifetime," or "referredCause."

B. Model IndeterminacySource

As shown in Fig. 33, a modeler can simply specify an indeterminacy source as a String value (D12) of attribute "indeterminacySource" of *Uncertainty* (Option 1). Alternatively, one can create a package (D13) to organize indeterminacy sources (A2.2.1), create instances of any UML Classifier to represent an indeterminacy source and apply «IndeterminacySource» on them (A2.2.2), specify the nature and description of each indeterminacy source (A2.2.3), specify measurements for each indeterminacy source (C1), and associate the created classifiers to the "referredIndeterminacySource" attribute of *Uncertainty*.

C. Model Risk

A modeler can optionally associate an uncertainty to *Risk* (D12, D13). As shown in Fig. 34, one can simply specify *Risk* as a String value of the "riskLevel" attribute of *Uncertainty* (Option 1) or one of the predefined risk levels in enumeration *RiskLevel* (Option 2). Alternatively, one can create a package for *Risk* if such a package does not exist, followed by creating classes and/or data types to represent *Risks* and optionally applying "Risk" (A4.3.2). Afterward, a modeler can also optionally specify measurement for *Risk* (C1) and link the created classes and datatypes to *Uncertainty* via the "riskInDTViaClass" and/or "riskInDT" attributes (A4.3.3).

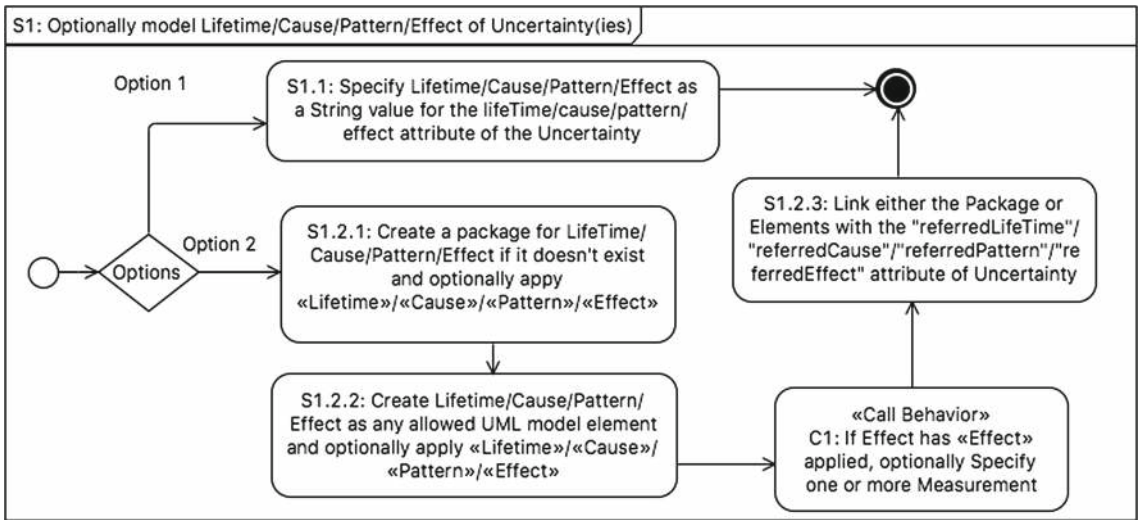


Fig. 32 Model Lifetime/Cause/Patten/Effect of Uncertainty

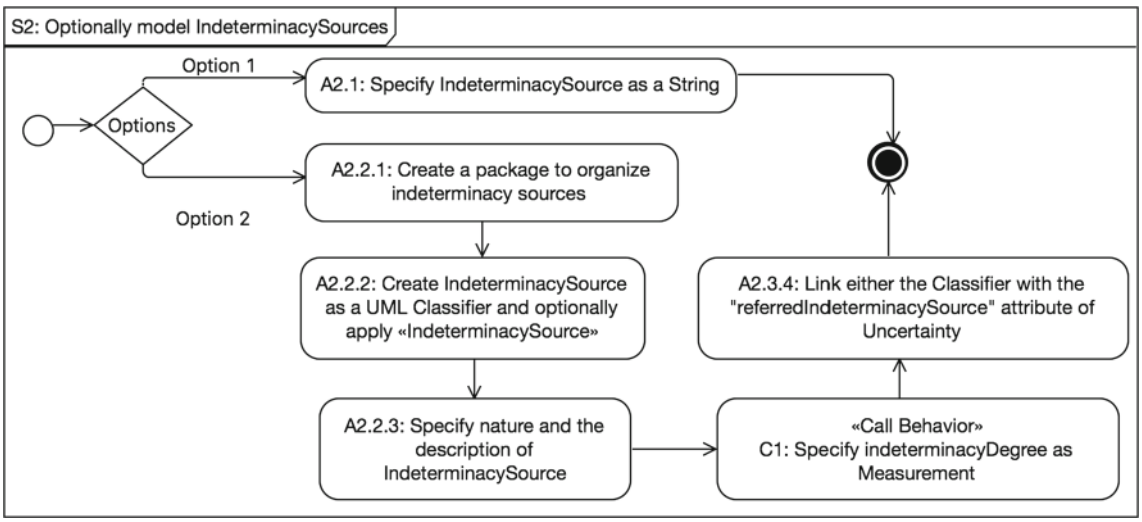


Fig. 33 Model IndeterminacySource

8 *UncerTum* validation process

In this section, we explain our validation process, which ensures that test ready models are syntactically correct and communication across state machines of various physical units constituting a CPS takes place correctly. Such validation is aimed at finding modeling errors that may have been introduced by a test modeler accidently. Once test ready models have been validated without any problems, test cases can be then generated from them. Since the execution of test ready models requires data to execute triggers, we generate data manually as follows: (1) If a trigger (Call Event/Signal Event) is guarded with 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.

To validate test ready models, we apply UAL [24] to execute them with IBM RSA Simulation Toolkit [53] (DD9 in Table 4). We decided to use UAL and IBM RSA Simulation Toolkit since our test generators are built in CertifyIt [13], which is a plugin for IBM RSA as we discussed in Sect. 1. Further, we provide a set of guidelines as an activity diagram to add UAL code on the test ready models in Sect. 8.1 and propose a set of recommended actions in Sect. 8.2, based

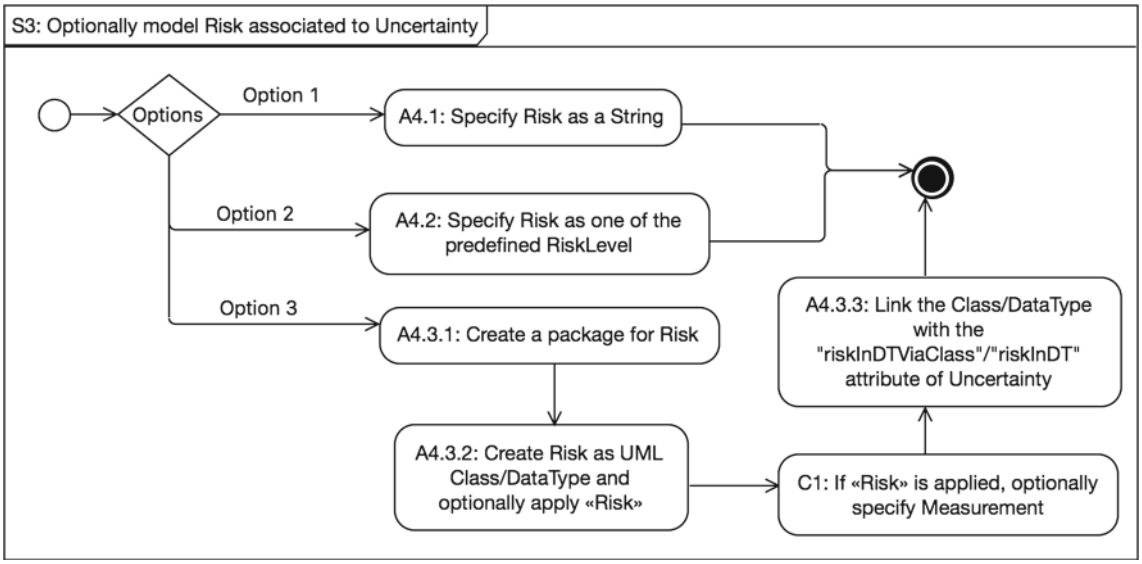


Fig. 34 Model risk

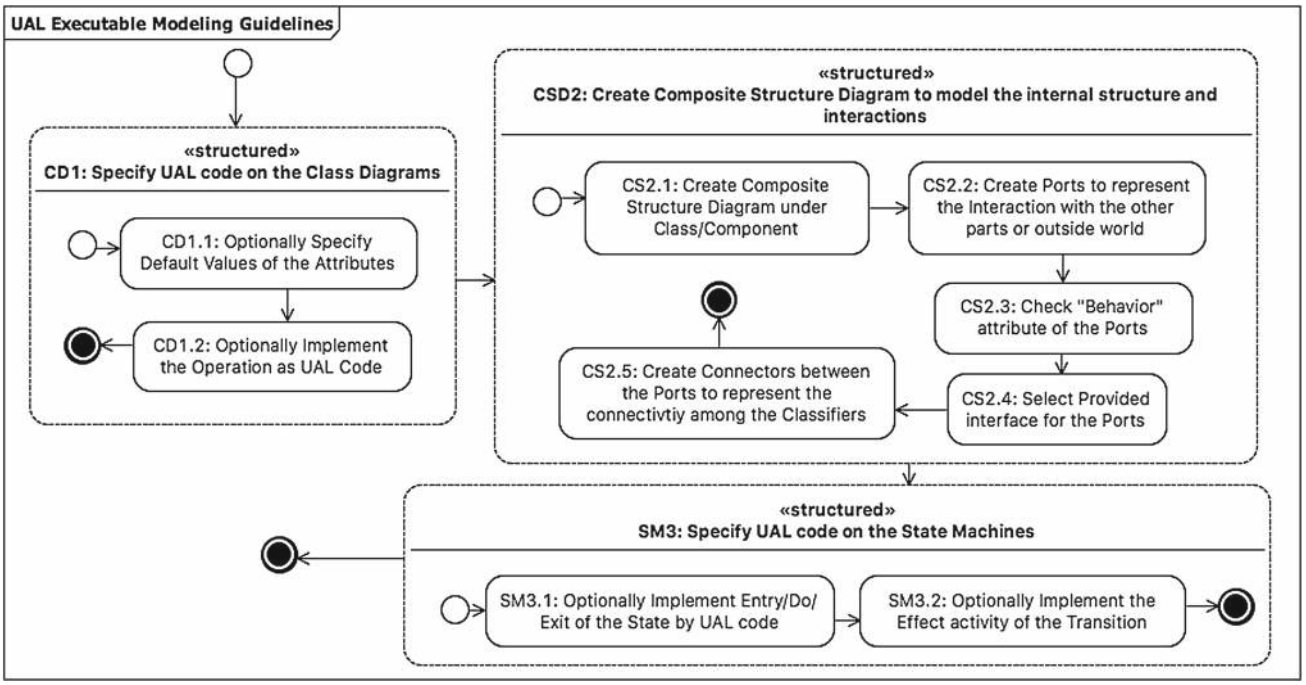


Fig. 35 Guidelines to create executable test ready models

on various types of problems identified while executing test ready models to help test modelers fix them.

8.1 UAL executable modeling guidelines

In this section, we describe the guidelines (in Fig. 35) to convert test ready models that were created based on the guidelines in the last section into executable models to facilitate validation.

As shown in Fig. 35, (1) in the CD1 activity, a test modeler can optionally specify UAL code on the model elements of classes (e.g., specifying default values for attributes and implementing bodies of operations). For example, the UAL code of the *timeout* attribute of *SecuritySystem* (Fig. 3) is *false*, i.e., its default value. (2) As shown in the CSD2 activity in Fig. 35, a test modeler should create a composite structure diagram (DD10, DD11 in Table 4) to model the internal structure of the Classifier (e.g., a physical unit) and interactions with other associated Classifiers (other physical units) or the

operating environment of the CPS. For example, the *portSecurity* port of *SecuritySystem* (Fig. 4) specifies an interaction point, through which *SecuritySystem* can communicate with its surrounding environment or with *Alarm* or *Sensor*. The provided interface of the *portSecurity* is *ISecuritySystem,* which enables the reception of the *IntrusionOccurred* signal and other *Signal Receptions* in Fig. 3. Two connectors between *portSecurity* and *portSensor* (Fig. 4) are created to enable two-way communications between *SecuritySystem* and *Sensor*. (3) As shown in SM3 in Fig. 35, a test modeler can specify UAL code on the effect and entry/do/exit activity of a state in a state machine to implement a specific activity, especially the ones that involve sending signals across state machines. For example, the effect of the *A* transition in Fig. 6 is implemented with UAL as portSensor.send(new IntrusionOccurred(this.ID)). Since *portSensor* and *portSecurity* are connected (Fig. 4) and provided interface *ISecuritySystem* of *portSecurity* has the capability to receive the *IntrusionOccurred* signal (in Fig. 3), the $B.1/B.2$ transition (in Fig. 7) can be triggered when *SecuritySystem* receives the *IntrusionOccurred* signal through *portSecurity*.

8.2 Recommendations to fix problems in test ready models

This section represents our recommendations (Table 5) to fix test ready models, once these are executed and problems are observed. For example, one observed problem is that the *IntrusionOccurred* signal event cannot be triggered (Fig. 7) even when it was sent out (O4, Table 5). One possible reason is that the *IntrusionOccurred Signal Reception* in the *ISecuritySystem* interface of *SecuritySystem* is missing (SA7).

9 Evaluation

In this section, we present the process of the development and validation of *UncerTum* with two industrial case studies (i.e., GS and AW), which were available to us as part of the project, one real-world case study (VCS), and one case study from the literature in Sect. 9.1, the results are described in Sect. 9.2, and overall discussion and limitations are presented in Sect. 9.3.

9.1 Development and validation of *UncerTum* and test ready models

As previously discussed, the project has two official CPS case study providers. First, the first one is from the healthcare domain, which is about GeoSports (GS) provided by Future Position X (FPX) [9] Sweden. This case study includes attaching devices to Bandy[1] players that record various measurements (e.g., heartbeat, speed, location) periodically. These measurements are communicated during a Bandy game via a receiver station to the sprint system, where coaches can monitor them at runtime. In addition, these measurements can also be used offline for analyses, for example, aimed at improving the performance of an individual player or a team. To test this CPS in a laboratory setting without real players, Nordic Med Test (NMT) [15] provides a test infrastructure to execute test cases. The second case study is about Automated Warehouse (AW) provided by ULMA Handling Systems [10], Spain. ULMA develops automated handling systems for worldwide warehouses of different natures such as Food and Beverages, Industrial, Textile, and Storage. Each handling facility (e.g., cranes, conveyors, sorting systems, picking systems, rolling tables, lifts, and intermediate storage) forms a physical unit, and together they are deployed to one handling system application (e.g., Storage). A handling system cloud supervision system (HSCS) generally interacts with diverse types of physical units, network equipment, and cloud services. Application-specific processes in HSCS are executed spanning clouds and CPS requiring different configurations. This case study implements several key industrial scenarios, i.e., introducing a large number of pallets to the warehouse, transferring the items by Stacker Crane. To test these scenarios, ULMA [10], and IK4-Ikerlan [16] developed and provided relevant simulators and emulators. Further details on the case studies can be consulted in [54].

In addition, we used a real-world case study of embedded Videoconferencing System (VCS) developed by Cisco Systems, Norway. Simula has been collaborating with Cisco since 2008. As part of our long-term collaboration under the umbrella of Certus Center [55], we have access to real VCS systems. We created test ready models for one of the real CPSs ourselves without involving Cisco, based on the previous work [18] of the second author of this paper. The fourth case study is a modified version of the SafeHome case study provided in [19]. This case study implements various security and safety features in smart homes including intrusion detection, fire detection, and flooding.

The development and validation procedure of *UncerTum* and test ready models is summarized in Fig. 36, which involves four stakeholders: (1) Simula Researchers (including the first three authors of this paper) play the key role of developing *UncerTum* and creating test ready models; (2) Use Case Providers (i.e., FPX and ULMA) provided uncertainty test requirements and real operational data from previous Bandy games in the case of GS, and manually checked the conformance of the developed test ready models to their corresponding uncertainty test requirements; (3)

[1] Bandy is a variation of ice hockey commonly played in Northern Europe.

Table 5 Recommended actions to fix test ready models based on observed problems

No.	Observed problem	Related problems and recommended action
O1	State change does not happen	**State Machines** SA1: Check the *Exit* activity of this *State*; SA2: Check *Guards* of all the outgoing *Transitions* of this *State*; SA3: Check if one or more outgoing *Transitions* are missing; **Related Problems** O4, O5, O6
O2	State invariant cannot be satisfied	**State Machines** SA4: Check the *State Invariant* of this *State* SA5: Check the incoming *Transition(s)* of this *State*; SA6: Check if one or more *States* are missing; **Related Problems** O7
O3	State cannot be reached	**Related Problems** O7, O9
O4	Signal Event cannot be triggered	**Class Diagrams** SA7: Check the *Reception* of the *Interface/Class/Component* **Composite Structure Diagrams** SA8: Check if the *Port* related to this signal event is linked with the correct *Provided Interface*; SA9: Check the *Connectors* between *Ports*; **State Machines** SA10: Check if the *Signal* corresponding to this *SignalEvent* is modeled;
O5	Call Event cannot be triggered	**State Machines** SA11: Check the invocation of the *Operation* corresponding to the *CallEvent*;
O6	Change Event cannot be triggered	**State Machines** SA12: Check the specified condition of this *ChangeEvent*; SA13: Check activities in parallel regions that manipulate the same attributes;
O7	Transition happens without any trigger	**State Machines** SA14: Check the *Trigger* of this *Transition*, especially for *ChangeEvent* and *TimeEvent*; SA15: Check the *Guard* of this *Transition*;
O8	State invariant of this state is overlapping with another state invariant(s) leading to firing an unexpected transition	**State Machines** SA16: Check if the *Guard* conditions of all or subset of the outgoing *Transition*s of this state have overlapping. SA17: Check if *Uncertainty(ies)* of this *Transition* are missing;
O9	Unexpected loop in the State Machine	**Related Problems** O7

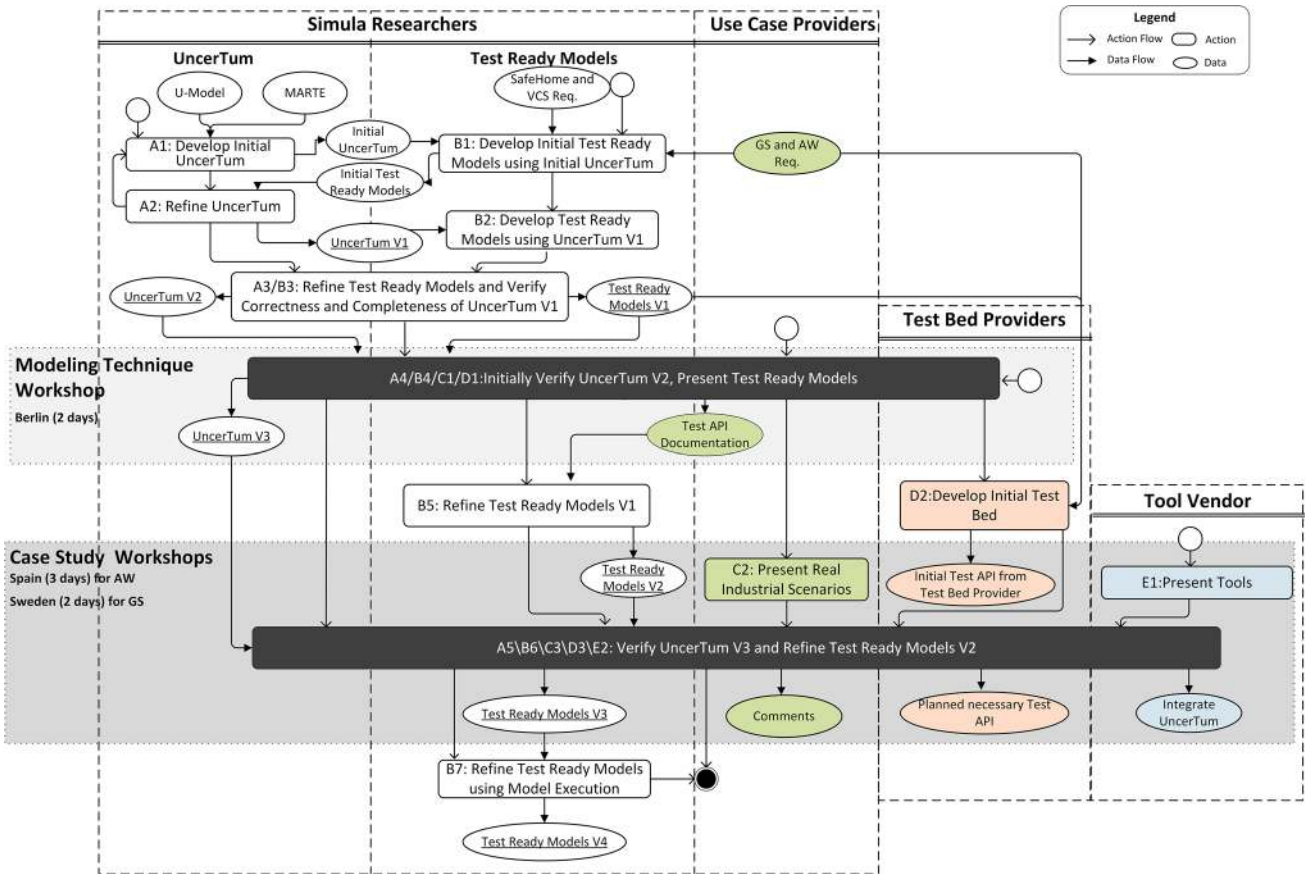


Fig. 36 Development and validation of *UncerTum* and test ready models

Test Bed Providers (NMT and ULMA/IK4-Ikerlan) provide physical and software infrastructures (including test APIs) to automate the execution of test cases and manually checked that the test ready models conform to the provided implementation of the test APIs; (4) Tool Vendor is responsible to integrate *UncerTum* and the proposed test case strategies to facilitate test case generation and execution. Please note that the *UncerTum* methodology reported in this paper is fully developed by Simula Research Laboratory, which is generic and therefore can be applied to test CPS at the three levels. Notice that it is also possible to develop different modeling methodologies than the one proposed in this paper, e.g., one such instance is reported in [56] for the application level by one of the our project partners. Such modeling methodologies can be potentially compared when needed in the future.

The development of *UncerTum* took place incrementally (Activities A1 and A2 in Fig. 36). First, *UncerTum* (A1) was developed by researchers based on U-Model and MARTE, in parallel to creating the initial test ready models (B1) for VCS, SafeHome, GS, and AW with this initial version of *UncerTum*. For GS and AW, uncertainty test requirements were provided by FPX and ULMA; for VCS, we had some requirements available to us from our previous work [18]; SafeHome is from the literature. Based on our experience of creating these test ready models, we further refined *UncerTum* (A2) and as a result *UncerTum* V.1 was developed. This was in turn used to further refine the initial test ready models (B2). At this point, both versions of the test ready models and *UncerTum* were refined once again by researchers. As a result, Test Ready Model V.1 and *UncerTum* V.2 were produced (Fig. 36).

UncerTum V2 and Test Ready Models V1 were then used in the modeling technique workshop (two days) conducted with the industrial use case providers (FPX and ULMA), test bed providers (ULMA/IK4-Ikerlan and NMT), tool vendor (Easy Global Market (EGM))[14], and two other research partners who focused on their own modeling methodologies and models. During the workshop, *UncerTum* and test ready models were presented to the participants of the workshop and their feedback was collected. In addition, the test API documentation was also presented. Based on the feedback and test APIs, a plan was devised to further refine the test ready models after the workshop. The key output of the workshop from our side was *UncerTum* (V.3), which is presented in this paper. Based on the feedback and test API documentation, we refined the test ready models (i.e., Test Ready Models

Table 6 Descriptive statistics of the case studies

Case study	CPS profile	Class diagram/composite structure diagram	State machine	Total
SafeHome	# of diagrams	2	3	5
	# Application Elements	15	10	66
	# Infrastructure Elements	16	19	
	# Integration Elements	3	3	
	# Uncertainties/IndeterminacySource	7	10	17
VCS	# of diagrams	6	12	18
	# Application Elements	92	59	442
	# Infrastructure Elements	103	67	
	# Integration Elements	51	70	
	# Uncertainties/IndeterminacySource	24	83	107
GS	# of diagrams	3	4	7
	# Application Elements	31	99	226
	# Infrastructure Elements	36	34	
	# Integration Elements	6	20	
	# Uncertainties/IndeterminacySource	10	29	39
AW	# of diagrams	4	11	15
	# Application Elements	39	52	91
	# Infrastructure Elements	52	75	127
	# Integration Elements	11	33	44
	# Uncertainties/IndeterminacySource	20	52	72

V2 in Fig. 36) after the workshop. In parallel, the test bed providers started to develop the test infrastructures to enable the execution of test cases, which is not in the scope of this paper.

To further refine the test ready models, another two workshops were conducted: one for AW and one for GS arranged by the respective industrial partners. The first workshop took place at IK4-Ikerlan [16], where Simula researchers and ULMA participated and the workshop lasted for three days. During the workshop, detailed uncertainty test requirements, test ready models, and detailed implementation of test execution were discussed. The second workshop lasted for two days and took place at the NMT's site in Sweden. FPX, EGM, and other research partners participated. Similar discussion as with ULMA took place with FPX/NMT. In addition, EGM presented their tool (CertifyIt) and their plans to integrate *UncerTum* and further implementation of test execution APIs. The outputs of these workshops were Test Ready Models V3 as shown in Fig. 36. Finally, we validated Test Ready Models V3 using IBM RSA Simulation Toolkit (see Sect. 9.2.3 for results).

9.2 Evaluation results

Descriptive statistics of the test ready models developed for the four case studies are provided in Table 6. For each case study, (1) the number of modeled UML diagrams is presented in the first row, (2) the second, third, and fourth rows represent the number of application, infrastructure, and integration-level elements, respectively, (3) the last row shows the number of uncertainties and indeterminacy sources modeled for each case study. Notice that these statistics provide an indication of the complexity and scale of the developed test ready models.

9.2.1 Mapping UUP/Model Libraries to U-Model and MARTE

This section provides the descriptive statistics for the mapping of the *UUP* model elements and the Model Libraries to concepts defined in U-Model and elements in MARTE.

Table 7 is divided into four main sections. First, we provide the statistics of elements in *UUP*/Model Libraries that can be directly mapped to U-Model. For example, «BeliefStatement» in *UUP* can be directly mapped to the *BeliefStatement* concept defined in U-Model. Second, we provide the statistics of elements in *UUP*/Model Libraries (e.g., *BeliefInterval*) that can be indirectly mapped to *U-Model* concepts (e.g., *Ambiguity*). Third, we provide statistics of elements that are introduced to *UUP*/Model Libraries (e.g., «BeliefElement») by extending U-Model concepts (e.g., *BeliefStatement*). Fourth, since the Model Libraries are developed via extending MARTE, we also provide statistics for mapping elements in *UUP*/Model Libraries to elements in

Table 7 Mapping *UUP*/model libraries to U-Model and MARTE

UncerTum model elements	U-Model															
	Directly mapped (x, y, z, t)				Indirectly mapped (x, y, z, t)				Newly added (x, y, z, t)				*MARTE*		Coverage $(n,\ p(\%))$	
UUP																
Belief	8	13	3	24	0	6	0	6	1	0	0	1	0	27	30	
Uncertainty	7	12	7	26	1	9	0	10	1	3	0	4	0	32	36	
Measure	7	5	5	17	0	1	12	13	12	10	0	22	0	15	17	
Total	22	30	15	67	1	16	12	29	14	13	0	27	0	74	83	
Model Library																
Risk	1	0	0	1	0	0	0	0	9	0	0	9	0	0*	0	
Pattern	7	4	0	11	0	0	0	0	4	0	0	4	8	6	7	
Measure	0	0	0	0	3	0	0	3	55	34	3	92	10	0*	0	
Time	2	0	0	2	0	0	0	0	4	0	0	4	6	2	2	
Total	10	4	0	14	3	0	0	3	72	34	3	109	24	8	9	
Total	32	34	15	81	4	16	12	32	86	47	3	136	24	82	92	
Percentage (%)	13	14	6	33	2	6	5	13	35	19	1	54	10			

#x is the number of Class/Stereotype/Enumeration/DataType in *UUP*/Model Libraries
#y is the number of Attributes/Associations in *UUP*/Model Libraries
#z is the number of Constraint(s) in *UUP*/Model Libraries
#t is the sum of #x, #y and #z
#n is the number of concepts (Class/Enumeration/ Association) in U-Model that are mapped to *UUP*
#p is the percentage of coverage, $p = \frac{n}{89}$ (the total number of concepts of U-Model is 89)
0* means the number that is covered by others

MARTE. For example, 10 data types in the *Measure* library can be mapped to MARTE.

As we can see from the last row of Table 7, 33% of the elements in *UUP*/Model Libraries can be directly mapped to U-Model, whereas 13% of elements can be indirectly mapped to U-Model, 54% of elements were newly introduced by extending U-Model concepts, most of which are for measures. In addition, 10% of *UUP*/Model Libraries elements were either directly adopted from MARTE or are extensions of MARTE elements. The last column of Table 7 shows the coverage of the U-Model concepts, from which one can observe that 83% of the U-Model concepts were implemented in *UUP*, whereas 9% of the U-Model concepts were implemented in the Model Libraries. The remaining 8% of the concepts that were not mapped to any element of *UUP*, and the Model Libraries are the ones related to *Knowledge*. Such concepts are important at the conceptual level and are defined based on well-defined taxonomies of *Knowledge* [57], but are not required to be implemented in *UUP* and the Model Libraries. From these results, we can see that U-Model is comprehensive enough to develop *UncerTum* and it has potential to be used as the basis for other researchers and practitioners to develop similar kinds of uncertainty-related modeling solutions in the future. We, therefore, consider data reported here as a useful experience that can be shared with the community. On the other side, from the reported data, one can get confidence about *UncerTum*, as it was indeed developed by following a rigor process and a comprehensive conceptual model.

9.2.2 Application of UUP/Model Libraries

In this section, we present the results of our evaluation with the aim of assessing the applicability of *UncerTum* in terms of effort required to create test ready models. We conducted the evaluation from two aspects: (1) the percentage of the applied *UUP*/Model Libraries elements in all the test ready models (UML class diagrams and state machines) developed for all the four case studies and (2) the effort in terms of time required to apply *UUP*/Model Libraries. The first aspect focuses on assessing the effort in terms of the number of model elements and gives us a surrogate measure of measuring effort, whereas the second aspect focuses on measuring the effort in terms of time taken by the test modelers to create the test ready models. In our case studies, the first author (second year Ph.D. candidate) created the first version of the test ready models, which were iteratively discussed with the second (a senior scientist) and third (a chief scientist) authors of this paper. In addition, as we discussed in Sect. 9.1, the test ready models were discussed with other partners involved in the project. As it does not exist an approach comparable with *UncerTum* in the literature (see more discussions in Sect. 10), we, therefore, do not have a comparison baseline. Conducting controlled experiments with test modelers could

Table 8 Percentage of *UUP*/Model Libraries Concepts to UML Concepts

Case study	Class diagram		State machine		%*UUP*/Model Libraries elements
	Class (u/t)	Relationship (t)	State (u/t)	Transition (u/t)	
SafeHome	7/21	18	3/17	7/29	20
VCS	24/197	303	39/216	61/278	12
GeoSports	10/62	56	13/82	26/106	16
AW	20/92	166	17/88	42/122	17
Average percentage of effort in terms of additional model elements: 16.25%					

#u: the number of elements with applied *UUP*/Model Libraries; #t: the total number of elements modeled using UML

Table 9 Effort (time in hours) of applying *UUP*/Model Libraries

Case study	Class diagram		State machine		% Time
	UML Modeling	*UUP*/Model Libraries Modeling	UML Modeling	*UUP*/Model Libraries Modeling	
SafeHome	4.5	0.5	22.5	7.5	22
VCS	22.5	6	45	15	23
GeoSports	37.5	3.5	52.5	12.5	15
AW	39.5	5.5	75	12.5	14
Average percentage of effort in time: 18.5%					

be a better option, which is fortunately under the plan and is a future work item, though it is notably that conducting such controlled experiments is often time and monetarywise expensive.

As shown in Table 8, for the SafeHome case study, in total we modeled 21 classes in the class diagrams, 7 out of which have *UUP* stereotypes applied (e.g., the «IndeterminacySource» sensor is applied to *Sensor*, see Fig. 3). For the modeled state machines, three out of 17 states and seven out of 29 transitions require the application of *UUP*/Model Libraries. In total, as shown in the last column of the table, around 20% of the modeling elements of the SafeHome case study required the application of *UUP*/Model Libraries. Similarly, 12% (16, 17%) of the modeling elements for the VCS (GS, AW) case study required the application of *UUP*/Model Libraries. For all the four case studies, on average 16.25% of the model elements require applying *UUP*/Model Libraries.

Table 9 summarizes effort [measured in time (hours)] spent by the first author (the modeler) on constructing the test ready models for the four case studies. The effort is divided into two parts: time for applying standard UML notations and additional effort required for applying *UUP*/Model Libraries. For example, as shown in Table 9, for *SafeHome*, it took the modeler 4.5 h for modeling the UML class diagrams, whereas additional 0.5 h was spent on applying *UUP*/Model Libraries. For the UML state machines, it took 22.5 h, whereas additional 7.5 h were spent on applying *UUP*/Model Libraries. For *SafeHome*, as shown in the last column (%Time) of Table 9, it took additional 22% of time to apply *UUP*/Model Libraries. Similarly, for VCS it took additional 23% of time, 15% of additional time for GS and 14% of additional time for AW. On average, for all the four case studies, modeling with *UUP*/Model Libraries required additional 18.5% of the total modeling effort.

9.2.3 Validation of test ready models via model execution

In this section, we present the results of the validation of the test ready models developed with *UncerTum* for the four case studies. The overall aim is to check the correctness of the test ready models against collected (uncertainty) requirements. The test ready models were enriched with UAL (a implementation of the Action Language For Foundational UML [24], Alf [58]))—a formal language supported in IBM RSA [12] for executing UML models implemented in Java. UML models with UAL can be executed with IBM RSA Simulation Toolkit [53] as we discussed in Sect. 8.

Table 10 shows the results of the validation. We classified identified problems during the validation process into two main categories: incorrect and incomplete model elements (states and transitions) for each case study. For *State*, we report problems identified in state invariants and «BeliefElement» . For *Transition*, we report problems identified in *Guard*, *Trigger*, *Effect*, and «BeliefElement» . For *State*, in total, 79 problems (17 + 62) were identified across the four case studies, where 17 problems were related to *Incorrectness* and 62 were related to *Incompleteness*. For «BeliefElement» related to *State*, we identified 32 missing stereotypes.

Table 10 Results of the validation of the test ready models

Case study	State		Transition				Total
	StateInvariant	«BeliefElement»	Guard	Trigger	Effect	«BeliefElement»	
Incorrect							
SafeHome	1	0	0	0	1	0	38
VCS	6	0	0	5	0	0	
GeoSports	3	0	2	1	0	0	
AW	7	0	1	8	3	0	
Incomplete							
SafeHome	5	2	0	7	2	3	226
VCS	30	13	15	23	21	18	
GeoSports	11	9	2	4	2	4	
AW	16	8	12	6	6	7	
Total	79 (17, 62)	32	122 (22, 100)			32	264

#Incorrect: the number of elements corrected after simulation; #Incomplete: the number of concepts newly added after simulation; #of triggers: #CallEvent + #SignalEvent + #TimeEvent

For *Transition*, we discovered 122 problems, 22 (100) of which were related to *Incorrectness* (*Incompleteness*). For «BeliefElement» related to *Transition*, we identified 32 missing stereotypes.

The typical problems identified include: (1) A transition between two states was fired without any event (O7 in Table 5); (2) after firing a transition the state change did not occur or the state changed to an unexpected one (O1, O2 in Table 5); (3) failed to send signals across concurrent state machines (O4 in Table 5); (4) there were no non-deterministic transitions from a state (O8 in Table 5); (5) unexpected exit, block, or deadlock were observed in a state machine (O1, O9 in Table 5); (6) unreachable states were discovered (O3 in Table 5); and (7) a guard condition was always true (O2, O7 in Table 5). Notice that these problems are not a comprehensive set of problems, but demonstrate the most commonly observed ones. After simulating the test ready models, we ensure that our models are correct and complete and hence can be used for facilitating MBT.

9.2.4 Application of UTP V.2

Applying UTP V.2 is the last step of *UncerTum* modeling as shown in Fig. 24. In the running example, «TestItem» from the *Test Context* package of UTP V.2 was applied on *SecuritySystem* (Fig. 3) and «CheckPropertyAction» from the *Arbitration Specification* package of UTP V.2 was applied to the state invariant of *IntrusionDetected* (Fig. 7).

Table 11 reports the results of the application of UTP V.2 to the models of the case studies. Notice that we only report the descriptive statistics of the high-level packages (e.g., Arbitration Specification) of UTP V.2 instead of the number of applications of each stereotype. Notice that each high-level package contains a set of related stereotypes. For *SafeHome*, in total UTP V.2 stereotypes were applied 54 times, whereas 551 for VCS, 209 for GS and 247 for AW.

Table 11 Applications of UTP V.2 Stereotypes

Category	SafeHome	VCS	GeoSports	AW
Arbitration specification	20	246	92	101
Test data	29	278	106	122
Test configuration	2	15	7	12
Test context	3	12	4	12
Total	54	551	209	247

Based on our experience of applying UTP V.2, we discovered that it is a generic UML profile for MBT and does meet all our needs. However, we discovered that combining *UUP*/Model Libraries and UTP V.2 together is sufficient to model test ready models with uncertainty in our case.

9.3 Overall discussion and limitations

Based on the results presented in Sect. 9.2, we conclude our findings as follows: (1) With *UncerTum*, we were able to model all the identified uncertainties in the four case studies. Such modeling suggests that *UncerTum* is sufficiently complete to create test ready models of CPS with explicit consideration of various types of uncertainties to support testing of CPS in the presence of such uncertainties; (2) in terms of estimating the effort required to apply the *UUP* stereotypes and Model Libraries, we conclude that we need to apply them to on average 16.25% of model elements (Table 8). When estimating effort in terms of time, we observed that we needed on average additional 18.5% of time to apply *UUP* (Table 9); (3) with our model execution-based model validation, we managed to identify and fix in total 264 problems

across the four case studies (Table 10) which are necessary before test case generation as otherwise generated test cases would have been incorrect.

In terms of evaluation, we would like to highlight the fact this section reported a preliminary evaluation of *UncerTum* from various perspectives. A more thorough evaluation would require conducting surveys and questionnaires from the participants from our industrial partners to solicit their views about the modeling methodology in terms of, for example, understandability and usability. We plan to conduct such evaluation at the end of our project when the complete results have been transferred to the industry partners with the participants who are not the co-authors of this paper in order to obtain unbiased feedback about *UncerTum*.

We would also like to mention that *UncerTum* cannot be used to model detailed continuous behaviors of a CPS, to support, for example, analyses during the system design and analysis phase or to generate code. *UncerTum* only supports test modeling for enabling the generation of executable test cases. Such types of models are less detailed as compared to models used for code generation or models for design time analyses. This is due to the fact that testing is always concerned with sending a stimulus to the system and observing whether the system transits to a correct state because of the stimulus according to the expected behavior specified in a test ready model, developed for the system.

10 Related work

There are some works in the literature that attempt to deal with modeling uncertainty with UML. For example, the authors of [59] proposed to perform fuzzy modeling with UML 1.5 without violating its semantics, based on theoretical analyses of UML 1.5. However, the proposed extensions to UML 1.5 were not implemented and validated. Moreover, there is no evidence to show the proposed extensions can be applicable for UML 2.x.

To model uncertainty (inherent in real-world applications) with UML class diagrams, an extension was proposed in [60–62], which is referred to as fuzzy UML data modeling. The extension relies on two theories: fuzzy set and possibility distribution, and was later on further extended in [63] to transform fuzzy UML data models into representations in the fuzzy description logic (FDLR) to check the correctness of fuzzy properties. Furthermore, another automated transformation was proposed in [64] to transform fuzzy UML data models into web ontologies to support automated reasoning on fuzzy properties in the context of web services.

In [65], the UML profile (named as fuzzy UML) was proposed to model uncertainty on use case diagrams, sequence diagrams, and state machines. Another work in [66] formalizes UML state diagrams with fuzzy information and transforms them into fuzzy petri nets for supporting automated verification and performance analysis. In [67], the authors developed two stereotypes: *moveTo* and *moveTo?* for UML collaboration diagrams. The first stereotype is applied when a modeler has full confidence, whereas the second stereotype is used when the modeler lacks confidence.

In comparison with these works, *UncerTum* focuses on modeling uncertainty in a comprehensive and precise manner by considering various types of measures such as probability, vagueness, and fuzziness. The methodologies proposed in [60–62] for specifying fuzzy UML data can easily be integrated with our Model Libraries when needed. Notice that *UncerTum* is proposed to explicitly capture the uncertainty of CPSs for the purpose of supporting MBT of CPSs under uncertainty, and there is no evidence showing that these works can be used for this purpose.

The work reported in [68] is the closest to our work, where uncertainty in time is modeled in UML sequence diagrams applied with the UML-SPT profile [69]. These sequence diagrams are then used for test case generation by taking into consideration the uncertainties in time. This work, however, only supports modeling uncertainty in time on messages of sequence diagrams. In contrast, *UncerTum* covers other types of uncertainties, in addition to time, such as content and environment. Moreover, the work does not account for sources of time uncertainties that are essential to be explicitly captured in order to introduce uncertainties for test execution.

In [70], the authors presented a solution to transform UML use case diagrams and state diagrams into usage graphs appended with probability information about expected use of the software. Such probability information can be obtained in several ways by relying on domain expertise or usage profiles of software, for example. Usage graphs with probability can be eventually used for testing. This work only deals with modeling uncertainty using probabilities and does not support other types of uncertainty measures such as ambiguity as supported in *UncerTum*. In addition, the work only supports modeling application-level uncertainties and cannot be used to model uncertainties in the other two CPS levels as *UncerTum*.

In [71], a language-independent solution was proposed for *Partial Modeling* with four types of partialities: *May partiality*, *Abs partiality*, *Var partiality*, and *OW partiality*, to denote the degree of incompleteness specified by model designers. The work also provides a solution for merging and reasoning possible partial models with tool support [72,73]. The approach was demonstrated on UML class and sequence diagrams [71]. This work is related to our work in terms of expressing the uncertainty of modelers. In *UUP*, the *Belief*-related stereotypes and classes capture subjective views of modelers and provide modeling notations for specifying the degree of their confidence (uncertainty) on the models they built. A set of possible models may have different belief

degrees provided by different belief agents at the same time. In the context of their work, the focus is on uncertainty in partial models for supporting model refinement and evolution. In contrast, *UUP* focuses on modeling uncertainty (lack of confidence) in test ready models to support MBT of CPSs under uncertainty.

11 Conclusion and future work

To facilitate model-based testing (MBT) of Cyber-Physical Systems (CPSs) under uncertainty, we proposed in this paper Uncertainty Modeling Framework (*UncerTum*). *UncerTum* allows creating test ready models with uncertainty at three logical testing levels of CPSs: *Application*, *Infrastructure*, and *Integration*. The core of *UncerTum* is the UML Uncertainty Profile (*UUP*), which implements an existing uncertainty conceptual model, called U-Model. In addition, *UncerTum* defines a comprehensive set of UML Model Libraries extending the UML profile for Modeling and Analysis of Real-Time and Embedded Systems (MARTE), which can be used together with *UUP*. *UncerTum* also relies on UML Testing Profile (UTP) V.2 to construct test ready models. Finally, *UncerTum* defines concrete guidelines for supporting the use of *UncerTum* for creating and validating test ready models with uncertainty. We evaluated *UncerTum* with two industrial, one real-world case study, and one open-source case studies. As a future work, we are implementing test generators that can take test ready models created with *UncerTum* as input and generate executable test cases.

Acknowledgements This research was supported by the EU Horizon 2020-funded project (testing Cyber-Physical Systems under Uncertainty, Project number: 645463). Tao Yue and Shaukat Ali are also supported by RCN-funded Zen-Configurator project, RFF Hovedstaden-funded MBE-CR project, RCN-funded MBT4CPS project, and RCN-funded Certus SFI.

Appendix A: The definitions of U-Model

To understand uncertainty in CPSs, we define the conceptual model, U-Model, to specify, classify, and identify uncertainty and its associated concepts that was evaluated by two industrial case studies [7]. *UncerTum* is an implementation of U-Model to enable MBT of CPSs under uncertainty by specializing the modeling notations and modeling activities. U-Model [7] was published in ECMFA 2016, and we have added definitions from the U-Model in this section to make this paper self-contained.

A.1: Belief Model

The Belief Model in Fig. 37 takes the subjective way to represent uncertainty—uncertainty is a subjective phenomenon that is indelibly bound to the worldview held by a belief agent,—that, for whatever reason, is incapable of possessing complete and fully accurate knowledge about some subject of interest [7]. In addition, the definitions of the concepts in Belief Model are represented in Table 12.

A.2: Uncertainty Model

The Uncertainty Model in Fig. 38 inspired by the literature of uncertainty expands on uncertainty from several different viewpoints and introduces related abstractions [7], i.e., risk, pattern, and the definitions of the concepts in Uncertainty Model are represented in Table 13.

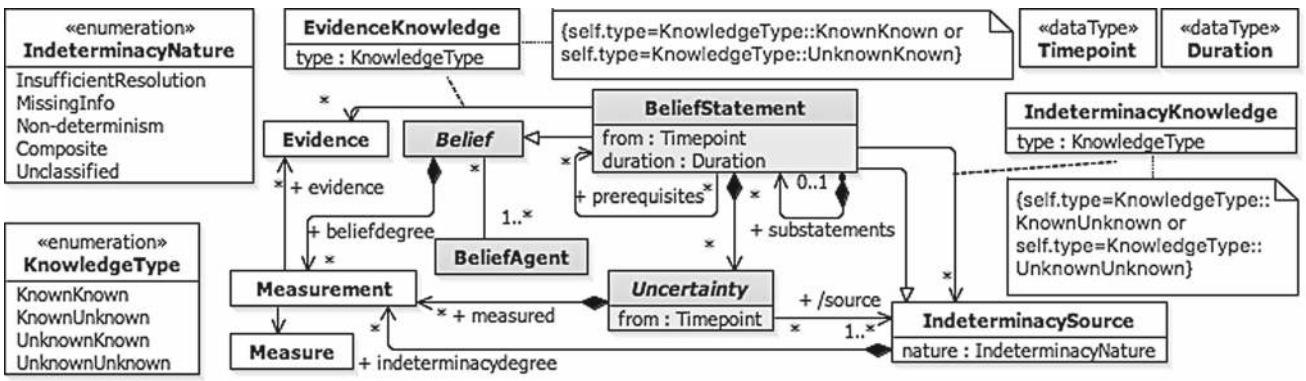


Fig. 37 The core belief model of U-Model

Table 12 The definitions of concepts in the Belief Model

Concept	Definition
Belief (abstract)	A belief is an *implicit* subjective explanation or description of some phenomena or notions[2] that is held by a *BeliefAgent*
	Semantics: This is an abstract concept whose only concrete manifestation is in the form of a belief statement
	Features:
	beliefdegree [*] The *Measurement* of *Belief* derived from *Measurement* of *Uncertainties* in this Belief
	beliefAgents [1..*] The set of *BeliefAgent* held this *Belief*
BeliefAgent	*BeliefAgent* represents an individual, a community of individuals sharing the same set of beliefs, or a technology, such as a software system, with built-in beliefs
	Semantics: A belief agent is a physical entity[3] that holds (i.e., owns) one or more beliefs about phenomena or notions associated with one or more subject areas derived from Indeterminacy. This could be a human individual or group, an institution, a living organism, or even a machine such as a computer. Crucially, a belief agent is capable of actions based on its beliefs
	Features:
	beliefs [*] The set of *Belief* that represent the full set of beliefs held explicitly by the *BeliefAgent*
BeliefStatement	A *BeliefStatement* is an *explicit* specification of some *Belief* about a possible phenomenon or notions belonging to a given subject area
	Generalizations: *Belief*, *IndeterminacySource*
	Semantics: The concrete form of this statement can vary, and may represent informal pronouncements made by individuals or groups, documented textual specifications expressed in either natural or formal languages, formal or informal diagrams, etc. Since it represents a belief, which is a subjective concept, a *BeliefStatement* may not necessarily correspond to objective reality. This means that it could be completely false, or only partially true, or completely true. However, due to the complex nature of objective reality, it may not always be possible to determine whether or not a *BeliefStatement* is valid. Furthermore, the validity of a statement may only be meaningfully defined within a given context or purpose. Thus, the statement that "the Earth can be represented as a perfect sphere" may be perfectly valid for some purposes but invalid or only partly valid for others. For our needs, we are less interested in the validity of a *BeliefStatement* than we are in the level of *Uncertainty* that a belief agent associates with it
	Features:
	substatements [*]—The set of finer-grained *BeliefStatements* that are components of a composite *BeliefStatement*
	prerequisites [*]—The set of *BeliefStatement* on which this *BeliefStatement* depends.
	indeterminacySource [*]—The set of *IndeterminacySource* that this *BeliefStatement* involves.
	evidence [*]—The set of *Evidence* providing this *BeliefStatement*.
	uncertainties [*]—The set of expressions of uncertainty that qualify and/or quantify the degree to which the *BeliefAgent* lacks confidence in this BeliefStatement; this attribute provides the core link between the Belief portion and the Uncertainty portion of the core uncertainty model
	from [0..1]—The *Timepoint* when *BeliefStatement* is initialized
	duration [0..1]—The *Duration* when *BeliefStatement* is active
Evidence	*Evidence* is either the observation of or record of a real-world event occurrence or, alternatively, the conclusion of some formalized chain of logical inference, which provides information that may contribute to determining the validity (i.e., truthfulness) of a *BeliefStatement*
	Semantics: *Evidence* is fundamentally an objective phenomenon, representing *something that actually happened*. This means that we do exclude here the possibility of counterfeit or invented evidence. Nevertheless, although *Evidence* represents objective reality, it need not be conclusive in the sense that it removes all doubt (uncertainty) about a *BeliefStatement*. On the other hand, any valid *BeliefStatement* must have at least some *Evidence* to support it
EvidenceKnowledge	*EvidenceKnowledge* expresses an objective relationship between a *BeliefStatement* and relevant *Evidence*
	Semantics: *EvidenceKnowledge* identifies whether the corresponding *BeliefAgent* is aware of the appropriate *Evidence*. Thus, an agent may be either aware that it knows something (*KnownKnown*), or it may be completely unaware of *Evidence* (*UnknownKnown*)
IndeterminacyNature	IndeterminacyNature represents the kind of indeterminacy[4]
	Enumeration literals:
	InsufficientResolution—The information available about the phenomenon in question is not sufficiently precise

Table 12 continued

Concept	Definition
	MissingInfo—The full set of data is unavailable at the time the statement is made
	Non-determinism—The phenomenon in question is either practically or inherently non-deterministic
	Composite—This represents some combination of multiple other kinds of indeterminacy
	Unclassified—Indeterminate indeterminacy
IndeterminacySource	*IndeterminacySource* represents a situation whereby the information required to ascertain the validity of a *BeliefStatement* is indeterminate in some way, resulting in uncertainty being associated with that statement.
	Semantics: One possible source of indeterminacy could be another *BeliefStatement*. A given indeterminacy source could in some cases be decomposed into more basic sources
	Features:
	indeterminacydegree [*]—This set of Measurement represents the quantification (or qualification) of this IndeterminacySource
	nature [1]—The *IndeterminacyNature* represents the kind of indeterminacy reason
IndeterminacyKnowledge	*IndeterminacyKnowledge* expresses an objective relationship between an *IndeterminacySource* and the awareness that the *BeliefAgent* has of that source
	Semantics: *IndeterminacyKnowledge* identifies whether the corresponding *BeliefAgent* is aware of the appropriate *IndeterminacySource*. So, even though it is agent specific, it is still an objective concept since it does not represent something that is declared by the agent. For instance, an agent may be aware that it does not know something about a possible source (*KnownUnknown*), or the agent may be completely unaware of a possible source of indeterminacy (*UnknownUnknown*)
KnowledgeType	*KnowledgeType* represents the type of the knowledge
	Enumeration literals:
	KnownKnown—Indicates that an associated *BeliefAgent* is consciously aware of some relevant aspect
	KnownUnknown (Conscious Ignorance)—Indicates that an associated *BeliefAgent* understands that it is ignorant of some aspect
	UnknownKnown (Tacit Knowledge)—Indicates that an associated *BeliefAgent* is not explicitly aware of some relevant aspect that it, nevertheless, may be able to exploit in some way
	UnknownUnknown (Meta Ignorance)—Indicates that an associated *BeliefAgent* is unaware of some relevant aspect
Measure	*Measure* represents the way of measuring *Belief*/*Uncertainty*/*IndeterminacySource*
	Semantics: *Measure* is *objective concept*, and specifies the existing way/theory to measure uncertainty.
Measurement	*Measurement* represents the optional quantification (or qualification) that specifies the degree of *Belief*/*Uncertainty*/*IndeterminacySource*
	Semantics: It may be possible to specify a *Measurement* that quantifies in some way (e.g., as a probability or a percentage) the degree of *Uncertainty* by the agent making the belief statement. Note, however, that this is a subjective measure defined by the *BeliefAgent*
	Features:
	measure [1]—This *Measure* represents the related way of measuring *Belief*/*Uncertainty*/*IndeterminacySource*
Uncertainty	*Uncertainty* (lack of confidence) represents *a state of affairs whereby a BeliefAgent does not have full confidence in a Belief that it holds*
	Semantics: "Full confidence" here means that the agent does not have any doubts about the validity of a statement. It is important to distinguish here between certainty and validity. That is, an agent could have full confidence in a *BeliefStatement* that is actually false; i.e., a statement that does not match (objective) truth. In general, the source of uncertainty associated with a *BeliefStatement* is that, for some reason, the agent does not have full knowledge of all relevant facts pertaining to the phenomena or notions that are the subject of the statement
	Features:
	from [0..1]—The *Timepoint* when *Uncertainty* is initialized

Table 12 continued

Concept	Definition
	measured [*]—This *Measurement* is used for representing confidence degree of *Uncertainty* by the agent making the *BeliefStatement*
	source [1..*]—This set of *IndeterminacySource* derived from the involves association and generalization of *BeliefStatement*

[2] The term "phenomena" here is intended to cover aspects of objective reality, whereas "notion" covers abstract concepts, such those encountered in mathematics or philosophy.
[3] We exclude here from this definition "virtual" belief agents, such as those that might occur in virtual reality systems and computer games.
[4] Indeterminacy represents a situation whereby the full knowledge necessary to determine the required factual state of some phenomena or notions is unavailable

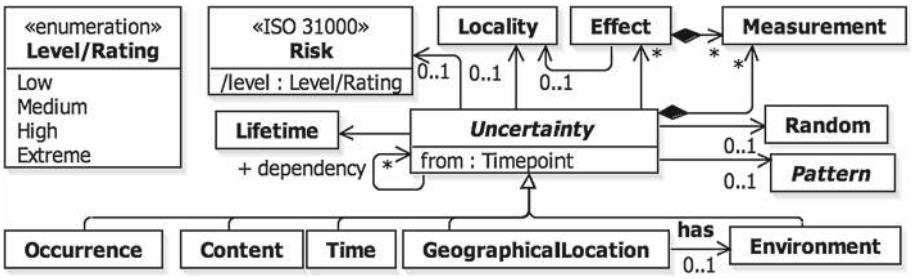


Fig. 38 The core uncertainty model of U-Model

Table 13 The definitions of concepts in the core uncertainty model

Concept	Definition
Effect	*Effect* represents the result of *Uncertainty* in the *BeliefStatement*
	Semantics: An uncertainty may result into: (1) another known *Uncertainty*, (2) something known and is not *Uncertainty*, (3) anything unknown
	Features:
	locality [0..1]—This value is used to represent that the *Locality* of the *Effect*
	measurement [*]—This set of *Measurement* represents the quantification (or qualification) of this *Effect*
Lifetime	*Lifetime* represents the duration of time for which an *Uncertainty* remains active
	Semantics: The length of time for which *Uncertainty* exists. For example, an *Uncertainty* may appear temporarily for a short period of time and disappears itself. On the other hand, an *Uncertainty* could be persistent, i.e., it stays active until appropriate actions are taken to resolve the *Uncertainty*
Locality	A particular place or a position where *Uncertainty* occurs in the *BeliefStatement*
	Semantics: A location could be a geographical location or a position where *Uncertainty* occurs. The concept of location is different than the *Uncertainty* type *GeographicalLocation*, where *Uncertainty* is due to the geographical location, however in this concept *Uncertainty* occurred at a location may not be due to the geographical location
Pattern	*Pattern* represents an intelligible way in which an *Uncertainty* appears.
	Semantics: An *Uncertainty* may occur without any *Pattern*, i.e., *Random*, or may have a pattern in which it may occur, for example, occurring at equal intervals of time, i.e., Periodic
Random	An *Uncertainty* that occurs without definite method, purpose or conscious decision
	Semantics: An *Uncertainty* occurring without any specific pattern
Risk	*Risk* measures the risk associated with *Uncertainty*
	Semantics: An uncertainty may have an associated risk and high-risk uncertainties deserve special attention
Level/Rating	*Level/Rating* is derived from *Measurement* owned by *Uncertainty* (Probability of the Occurrence of an Uncertainty) and Measurement owned by Effect (e.g., high impact), for example, using the risk matrix [74] or any other matrices
Occurrence	*Occurrence* represents a situation whereby a *BeliefAgent* lacks confidence in occurrence existing in a *BeliefStatement*
	Generalizations: *Uncertainty*
Content	*Content* represents a situation whereby a *BeliefAgent* lacks confidence in content existing in a *BeliefStatement*
	Generalizations: *Uncertainty*
Time	*Time* represents a situation whereby a *BeliefAgent* lacks confidence in time existing in a *BeliefStatement*
	Generalizations: *Uncertainty*

Table 13 continued

Concept	Definition
GeographicalLocation	*GeographicalLocation* represents a situation whereby a BeliefAgent lacks confidence in geographical location existing in a *BeliefStatement*
	Generalizations: *Uncertainty*
Environment	*Environment* represents a situation whereby a *BeliefAgent* lacks confidence in environment existing in a *BeliefStatement*
	Generalizations: *Uncertainty*
Uncertainty	Semantics: The Uncertainty Model expands on *Uncertainty* from several different viewpoints and introduces related abstractions. Notice that Uncertainty has a self-association. This self-association facilitates: (1) relating different *Application level* uncertainties to each other, (2) relating different *Infrastructure level* uncertainties to each other, (3) relating *Application level* and *Infrastructure level* uncertainties to each other, (4) relating *Integration level* uncertainties to each other, and (5) relating *Application*, *Integration*, and *Infrastructure level* uncertainties. This self-association can be specialized into different types of relationships such as ordering and dependencies. Here, we intentionally did not specialize it to keep the model general, so that it can be specialized for various purposes and contexts
	Features:
	lifetime [1]—This value is used for representing the duration of this *Uncertainty*
	pattern [0..1]—This value is used for describing whether this *Uncertainty* happens in a pattern or what kind of the pattern this *Uncertainty* occurs in
	risk [0..1]—This value is used for whether this *Uncertainty* has a risk, and what kind of risk this *Uncertainty* causes
	locality [0..1]—This value is used for representing what location this *Uncertainty* occurs
	effect [*]—This value is used for describing what effect the uncertainty may produce
	dependency [*]—The set of *Uncertainty* represents the dependency relationship with other *Uncertainty*

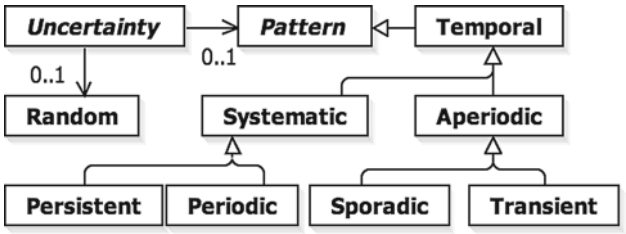


Fig. 39 The Pattern Model

A.2.1: Pattern Model

The Pattern Model in Fig. 39 shows the conceptual model for the occurrence pattern of Uncertainty [7], and the definitions of the concepts in Pattern Model are represented in Table 14.

A.3: Measure Model

The Measure Model in Fig. 40 describes the commonly known measures related to uncertainty [7], and the definitions of the concepts in Measure Model are represented in Table 15.

Table 14 The definitions of concepts in the Pattern Model

Concept	Definition
Temporal	*Uncertainty* occurring in a temporal pattern
	Generalizations: *Measure*
	Semantics: *Temporal* describes the notion of time with the occurrence of uncertainty
Systematic	*Uncertainty* occurring in a systematic pattern
	Generalizations: *Temporal*
	Semantics: *Uncertainty* occurring in some methodical pattern, i.e., a pattern that can be described in a mathematical way
Persistent	A permanent *Uncertainty*, i.e., lasting forever
	Generalizations: *Systematic*
	Semantics The definition of "forever" varies. For example, an uncertainty may exist permanently until appropriate actions are taken to deal with the uncertainty. On the other hand, an uncertainty may not be able to resolve and stays forever.
Periodic	An *Uncertainty* that occurs in repeated periods or at regular intervals
	Generalizations: Systematic

Table 14 continued

Concept	Definition
	Semantics: *Uncertainty* repeating itself after an equal interval of time
Aperiodic	An *Uncertainty* that occurs at irregular intervals of time
	Generalizations: *Temporal*
	Semantics: It is important to note that *Aperiodic* is inherited from *Temporal*; this means it has a notion of time in which the *Uncertainty* appears in an *Aperiodic* pattern
Sporadic	*Uncertainty* occurring in a sporadic pattern
	Generalizations: *Aperiodic*
	Semantics: *Uncertainty* occurring occasionally
Transient	*Uncertainty* occurring temporarily
	Generalizations: *Aperiodic*
	Semantics: *Uncertainty* that does not last long

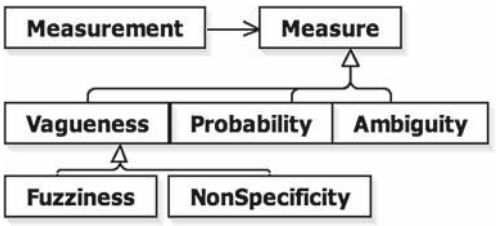


Fig. 40 The core measure model of U-Model

Table 15 The definitions of concepts in the core measure model

Concept	Definition
Vagueness	*Uncertainty* measured with the vagueness methods
	Generalizations: *Measure*
Fuzziness	*Uncertainty* measured by fuzzy methods. More details can be referred to the fuzzy logic literature [75]
	Generalizations: *Vagueness*
NonSpecificity	*Uncertainty* measured using non-specificity methods
	Generalizations: *Vagueness*
	Semantics: In certain cases, it may not be possible to measure an uncertainty using quantitative measurements and instead qualitative measurements can be used. Such qualitative measurements are classified under *Non-Specificity* methods
Probability	*Uncertainty* measured with the probability
	Generalizations: *Measure*
	Semantics: A quantitative way of measuring uncertainty
Ambiguity	*Uncertainty* in the *BeliefStatement* is measured using ambiguity way
	Generalizations: *Measure*
	Semantics: An uncertainty may be described ambiguously. For example, in the following statement: "The food is hot", the ambiguity is in the measurement, i.e., the food is either hot in terms of temperature or in terms of spices

References

1. Lee, E.A.: Cyber physical systems: design challenges. In: 2008 11th IEEE International Symposium on Object Oriented Real-Time Distributed Computing (ISORC), pp. 363–369 (2008)
2. Rawat, D.B., Rodrigues, J.J., Stojmenovic, I.: Cyber-Physical Systems: From Theory to Practice. CRC Press, Boca Raton (2015)
3. Sunder, S.: Foundations for innovation in Cyber-Physical Systems. In: Proceedings of the NIST CPS Workshop, Chicago, IL, USA
4. Geisberger, E., Broy, M.: Living in a Networked World: Integrated Research Agenda Cyber-Physical Systems (agendaCPS). Herbert Utz Verlag, Munich (2015)
5. Bammer, G., Smithson, M.: Uncertainty and Risk: Multidisciplinary Perspectives. Routledge, Abingdon (2012)

6. Lindley, D.V.: Understanding Uncertainty (Revised Edition). Wiley, Hoboken (2014)
7. Zhang, M., Selic, B., Ali, S., Yue, T., Okariz, O., Norgren, R.: Understanding uncertainty in Cyber-Physical Systems: a conceptual model. In: Proceedings of the 12th European Conference on Modelling Foundations and Applications (ECMFA), pp. 247–264
8. Ali, S., Yue, T.: U-test: evolving, modelling and testing realistic uncertain behaviours of Cyber-Physical Systems. In: Proceedings of the IEEE 8th International Conference on Software Testing, Verification and Validation (ICST), pp. 1–2
9. Future Position X. http://www.fpx.se/. Accessed 2017
10. ULMA Handling System. http://www.ulmahandling.com/en/. Accessed 2017
11. Object Management Group (OMG): UML profile for MARTE: modeling and analysis of real-time embedded systems, Version 1.1. http://www.omg.org/spec/MARTE/ (2011)
12. IBM Rational Software Architect Modeling Tool. https://www.ibm.com/developerworks/downloads/r/architect/. Accessed 2016
13. CertifyIt. http://www.smartesting.com/en/certifyit/. Accessed 2017
14. Easy Global Market. http://www.eglobalmark.com/. Accessed 2017
15. Nordic Med Test. http://www.nordicmedtest.se/. Accessed 2017
16. IK4-IKERLAN. http://www.ikerlan.es/eu/. Accessed 2017
17. Cisco. http://www.cisco.com/. Accessed 2017
18. Ali, S., Briand, L.C., Hemmati, H.: Modeling robustness behavior using aspect-oriented modeling to support robustness testing of industrial systems. Softw. Syst. Model. **11**(4), 633–670 (2012)
19. Pressman, R.S.: Software Engineering: A Practitioner's Approach, 7th edn. Palgrave Macmillan, Basingstoke (2010)
20. Object Management Group (OMG): UML testing profile, Version 1.2. http://www.omg.org/spec/UTP/ (2013)
21. Ali, S., Yue, T., Hoffmann, A., Wendland, M.F., Bagnato, A., Brosse, E., Schacher, M., Dai, Z.R.: How does the UML testing profile support risk-based testing. In: 2014 IEEE International Symposium on Software Reliability Engineering Workshops, pp. 311–316
22. UML Testing Profile™(UTP) 2.0. http://utp.zen-tools.com/
23. Object Management Group (OMG): UML testing profile, Version 2. http://utp.omg.org/, http://www.omg.org/cgi-bin/doc?ad/16-05-10 (2016)
24. IBM: UML Action Language (UAL). https://www.ibm.com/support/knowledgecenter/SS8PJ7_9.6.0/com.ibm.xtools.model.ual.doc/topics/c_umlactionlanguage.html (2017). Accessed 2017
25. Zhang, M., Ali, S., Yue, T., Nguyen, P.H.: Uncertainty Modeling Framework for the Integration Level V.1. Technical Report 2016-01 Simula Research Laboratory, 2016. https://www.simula.no/publications/uncertainty-modeling-framework-integration-level-v1 (2016)
26. Zadeh, L.A.: Fuzzy sets. Inf. Control **8**(3), 338–353 (1965)
27. Dempster, A.P.: Upper and lower probabilities induced by a multivalued mapping. Ann. Math. Stat. **38**(2), 325–339 (1967)
28. Shafer, G.: A Mathematical Theory of Evidence. Princeton University Press, Princeton (1976)
29. Hartley, R.V.L.: Transmission of information. Bell Syst. Tech. J. **7**(3), 535–563 (1928)
30. Lamata, M.T., Moral, S.: Measures of entropy in the theory of evidence. Int. J. Gen. Syst. **14**(4), 297–305 (1988)
31. Yager, R.R.: Entropy and specificity in a mathematical theory of evidence. Int. J. Gen. Syst. **9**(4), 249–260 (1983)
32. Higashi, M., Klir, G.J.: Measures of uncertainty and information based on possibility distributions. Int. J. Gen. Syst. **9**(1), 43–58 (1982)
33. Zadeh, L.A.: Fuzzy sets as a basis for a theory of possibility. Fuzzy Sets Syst. **1**(1), 3–28 (1978)
34. Smets, P., Kennes, R.: The transferable belief model. Artif. Intell. **66**(2), 191–234 (1994)
35. George, J.K., Bo, Y.: Fuzzy sets and fuzzy logic, theory and applications (2008)
36. Kosko, B.: Fuzzy entropy and conditioning. Inf. Sci. **40**(2), 165–174 (1986)
37. Didier, D., Henri, P.: Fuzzy Sets and Systems: Theory and Application. Mathematics in Science and Engineering, vol. 144. Academic Press, Cambridge (1980)
38. Zimmermann, H.-J.: Fuzzy Set Theory—And Its Applications. Springer, Berlin (2011)
39. Pawlak, Z.: Rough sets. Int. J. Comput. Inf. Sci. **11**(5), 341–356 (1982)
40. Goguen, J.A.: L-fuzzy sets. J. Math. Anal. Appl. **18**(1), 145–174 (1967)
41. Atanassov, K., Georgiev, C.: Intuitionistic fuzzy prolog. Fuzzy Sets Syst. **53**(2), 121–128 (1993)
42. Zadeh, L.A.: The concept of a linguistic variable and its application to approximate reasoning—I. Inf. Sci. **8**(3), 199–249 (1975)
43. Grattan-Guinness, I.: Fuzzy membership mapped onto intervals and many-valued quantities. Math. Logic Q. **22**(1), 149–160 (1976)
44. Jahn, K.U.: Intervall-wertige Mengen. Math. Nachr. **68**(1), 115–132 (1975)
45. Gau, W.L., Buehrer, D.J.: Vague sets. IEEE Trans. Syst. Man Cybern. **23**(2), 610–614 (1993)
46. De Luca, A., Termini, S.: A definition of a nonprobabilistic entropy in the setting of fuzzy sets theory. Inf. Control **20**(4), 301–312 (1972)
47. Feller, W.: An Introduction to Probability Theory and Its Applications. Wiley, Hoboken (2008)
48. Song, H., Rawat, D.B., Jeschke, S., Brecher, C.: Cyber-Physical Systems: Foundations, Principles and Applications. Morgan Kaufmann, Burlington (2016)
49. Talcott, C.: Cyber-Physical Systems and events. In: Wirsing, M., Banâtre, J.-P., Hölzl, M., Rauschmayer, A. (eds.) Software-Intensive Systems and New Computing Paradigms: Challenges and Visions, pp. 101–115. Springer, Berlin (2008)
50. Ali, S., Briand, L.C., Rehman, M.J.-U., Asghar, H., Iqbal, M.Z.Z., Nadeem, A.: A state-based approach to integration testing based on UML models. Inf. Softw. Technol. **49**(11–12), 1087–1106 (2007)
51. Eclipse OCL. http://www.eclipse.org/modeling/mdt/?project=ocl-ocl. Accessed 2016
52. Dresden OCL. https://marketplace.eclipse.org/content/dresden-ocl. Accessed April, 2016
53. IBM RSA Simulation Toolkit. http://www-03.ibm.com/software/products/en/ratisoftarchsimutool. Accessed 2016
54. Use Cases—Industrial Case Studies. http://www.u-test.eu/use-cases/. Accessed 2017
55. Certus. http://certus-sfi.no/. Accessed 2017
56. Schneider, M., Wendland, M.-F.: Gaining certainty about uncertainty: testing for uncertainties of Cyber-Physical Systems at the application level. In: 4th International Workshop on Risk Assessment and Risk-driven Quality Assurance (RISK), In Conjunction with 28th International Conference on Testing Software and Systems (ICTSS) (2016)
57. Kerwin, A.: None too solid medical ignorance. Sci. Commun. **15**(2), 166–185 (1993)
58. Object Management Group (OMG):Concrete syntax For A UML action language: action language for foundational UML (ALF), Version 1.0.1. http://www.omg.org/spec/ALF/ (2013)
59. Sicilia, M.-A., Mastorakis, N.: Extending UML 1. 5 for fuzzy conceptual modeling: an strictly additive approach. WSEAS Trans. Syst. **3**(5), 2234–2239 (2004)
60. Ma, Z.: Fuzzy information modeling with the UML. Advances in fuzzy object oriented databases: modeling and applications. Idea Group Inc., USA, 153–75 (2004)

61. Ma, Z.M., Zhang, F., Yan, L.: Fuzzy information modeling in UML class diagram and relational database models. Appl. Soft Comput. **11**(6), 4236–4245 (2011)
62. Yan, L., Ma, Z.M. : Extending nested relational model for fuzzy information modeling. In: 2009 WASE International Conference on Information Engineering, pp. 587–590 (2009)
63. Ma, Z.M., Zhang, F., Yan, L., Cheng, J.: Representing and reasoning on fuzzy UML models: a description logic approach. Expert Syst. Appl. **38**(3), 2536–2549 (2011)
64. Zhang, F., Ma, Z.M.: Construction of fuzzy ontologies from fuzzy UML models. Int. J. Comput. Intell. Syst. **6**(3), 442–472 (2013)
65. Haroonabadi, A., Teshnehlab, M., Movaghar, A.: A novel method for behavior modeling in uncertain information systems. World Acad. Sci. Eng. Technol. **41**, 959–966 (2008)
66. Motameni, H., Daneshfar, I., Bakhshi, J., Nematzadeh, H.: Transforming fuzzy state diagram to fuzzy Petri net. J. Adv. Comput. Res. **1**(1), 29–44 (2010)
67. Grassi, V., Mirandola, R.: UML modelling and performance analysis of mobile software architectures. In: UML 2001—The Unified Modeling Language. Modeling Languages, Concepts, and Tools, pp. 209–224. Springer (2001)
68. Garousi, V.: Traffic-aware stress testing of distributed real-time systems based on UML models in the presence of time uncertainty. In: 2008 1st International Conference on Software Testing, Verification, and Validation, pp. 92–101
69. Object Management Group (OMG): UML profile for schedulability, performance, and time, Version 1.1. http://www.omg.org/spec/SPTP/ (2005)
70. Riebisch, M., Philippow, I., Götze, M.: UML-based statistical test case generation. In: Aksit, M., Mezini, M., Unland, R. (eds.) Objects, Components, Architectures, Services, and Applications for a Networked World. NODe 2002. Lecture Notes in Computer Science, vol. 2591, pp. 394–411. Springer, Berlin (2002)
71. Salay, R., Famelis, M., Chechik, M.: Language independent refinement using partial modeling. In: de Lara, J., Zisman, A. (eds.) Fundamental Approaches to Software Engineering. FASE 2012. Lecture Notes in Computer Science, vol. 7212, pp. 224–239. Springer, Berlin (2012)
72. Famelis, M., Salay, R., Chechik, M.: Partial models: towards modeling and reasoning with uncertainty. In: 2012 34th International Conference on Software Engineering (ICSE), pp. 573–583 (2012)
73. Famelis, M., Santosa, S.: MAV-Vis: a notation for model uncertainty. In: 2013 5th International Workshop on Modeling in Software Engineering (MiSE), pp. 7–12 (2013)
74. Garvey, P.R., Lansdowne, Z.F.: Risk matrix: an approach for identifying, assessing, and ranking program risks. Air Force J. Logist. **22**(1), 18–21 (1998)
75. Klir, G.: Facets of Systems Science. Springer, Berlin (2013)

Man Zhang is a Ph.D. student at Simula Research Laboratory, Norway (2015–present), and the Department of Informatics, University of Oslo, Norway. Previously, she obtained her Master Degree in Computer Technology from Beihang University, Beijing, China (2012–2015). Her main research interests include model-based testing of Cyber-Physical Systems, uncertainty modeling, model-based engineering and requirements engineering.

Shaukat Ali is currently a senior research scientist in Simula Research Laboratory, Norway. His research focuses on devising novel methods for Verification and Validation (V&V) of large-scale highly connected software-based systems that are commonly referred to as Cyber-Physical Systems (CPSs). He has been involved in several basic research, research-based innovation, and innovation projects in the capacity of PI/Co-PI related to model-based testing (MBT), search-based software engineering, and model-based system engineering. He has rich experience of working in several countries including UK, Canada, Norway, and Pakistan. Shaukat has been on the program committees of several international conferences (e.g., MODELS, ICST, GEECO, SSBSE) and also served as a reviewer for several software engineering journals (e.g., TSE, IST, SOSYM, JSS, TEVC). He is also actively participating in defining international standards on software modeling in Object Management Group (OMG), notably a new standard on Uncertainty Modeling.

Tao Yue is a chief research scientist of Simula Research Laboratory, Oslo, Norway, and she is also affiliated to University of Oslo as an associate professor. She has received the Ph.D. degree in the Department of Systems and Computer Engineering at Carleton University, Ottawa, Canada, in 2010. Before that, she was an aviation engineer and system engineer for 7 years. She has more than 15 years of experience of conducting industry-oriented research with a focus on model-based engineering (MBE) in various application domains such as avionics, maritime and energy, and communications in several countries including Canada, Norway, and China. Her present research area is software engineering, with specific interests in requirements engineering, product line engineering, model-based engineering and testing, search-based software engineering, empirical software engineering, and uncertainty modeling. Dr. Yue has been on the program and organization committees of several international conferences (e.g., MODELS, RE, SPLC). She is also on the editorial board of Empirical Software Engineering. Tao is also actively participating in defining international standards in Object Management Group (OMG), including uncertainty modeling, SysML, and UTP.

Roland Norgren holds a Bachelor's Degree in system science. He has worked mainly as developer and project manager in both small- and large-scale organisations. Roland is today working as process manager for research and innovation at the GIS-cluster Future Position X in Gävle, Sweden.

Oscar Okariz is with ULMA since 2010. He is Computer Engineer by the Basque University (UPV) since 2003. Oscar Okariz is recognized by his experience on J2EE and .Net development as well as his experience of more than ten years as software developer analyst on several enterprises managing and directing projects. He is currently "Software technology" area responsible in charge of leading internal team work to achieve a new generation software product in Ulma Handling Systems.