An Integrated Modeling Framework to Facilitate Model-Based Testing of Cyber-Physical Systems under Uncertainty

[1]Man Zhang, [1]Shaukat Ali, [1,2]Tao Yue
[1]Simula Research Laboratory,
[2]University of Oslo
Oslo, Norway
{manzhang, shaukat, tao}@simula.no

Roland Norgre
Future Position X,
Gävle, Sweden
roland.norgren@fpx.se

ABSTRACT

Tackling inherent uncertainty of Cyber-Physical Systems (CPSs) is essential for their reliable operations. A way of ensuring the quality of CPS under uncertainty is via methodical testing techniques such as Model-based Testing (MBT). Towards this direction, we present an Uncertainty Modeling Framework (UMF) for modeling test ready models to support MBT of CPSs under uncertainty at the three testing levels: Application, Infrastructure, and Integration. UMF relies on the definition of a UML profile (named as UML Uncertainty Profile (UUP)) and an extensive set of UML model libraries extending UUP. In addition, UMF also relies on the UML profile for Modeling and Analysis of Real-Time and Embedded Systems (MARTE) and the UML Testing Profile (UTP) V.2 for developing test ready models. UMF was evaluated by modeling test ready models with uncertainty for two industrial CPS case studies 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 called *U-Model* that was published earlier and MARTE, 2) *Effort* required to model uncertainty with UMF, and 3) Assessing *Correctness* of the developed models via simulating the test ready models with executable UML, 4) Experience of using UTP V.2 to create test ready models with uncertainty.

Keywords

Uncertainty; Cyber-Physical System; UML Profile; Model-based Testing

1. INTRODUCTION

Uncertainty in Cyber-Physical Systems (CPSs) during their real operations is inevitable and hence must be properly handled with systematic verification and validation techniques during their development phases to the maximum extent. Uncertainty in CPSs is an immature area of research in general and several efforts have just begun to study uncertainty in CPSs [50]. One such effort is being done in an EU project under the Horizon2020 program called U-Test (www.u-test.eu). The U-Test project aims to devise a set of modeling and testing methodologies for explicitly modeling test ready models (with uncertainty) for CPSs under test with the ultimate

aim of automatically generating test cases from the test ready models with Model-Based Testing (MBT) techniques.

In this paper, we report the Uncertainty Modeling Framework (*UMF*) developed under the U-Test project for modeling test ready models with known uncertainty at the three CPS testing levels: Application, Infrastructure, and Integration [2]. The core of *UMF* is the UML Uncertainty Profile (*UUP*), which is defined based on the uncertainty conceptual model for CPSs (*U-Model*) [50]. 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 to *UUP*, *UMF* also defines an extensive set of UML model libraries by extending the UML profile for Modeling and Analysis of Real-Time and Embedded Systems (MARTE) [18]. The key libraries include *Uncertainty Pattern Library*, *Measure Library* and *Time Library*. In addition, *UMF* relies on the UML Testing Profile (*UTP*) V.2 to model test ready models with uncertainty. Last, *UMF* also includes a set of guidelines with recommendations and alternative scenarios for applying these modeling notations.

UMF was evaluated with two industrial case studies and one extended open source case study from the literature. The first industrial case study is GeoSports provided by Future Position X, Sweden[1], a U-Test project partner. The second industrial case study is embedded Videoconferencing Systems developed by Cisco, Norway [2] and was used in our previous work [50]. The third open source case study is SafeHome from [37]. We performed evaluation from these four perspectives: 1) Completeness and coverage of *UUP*/Model Libraries to *U-Model* and MARTE, 2) Effort required to model uncertainty using *UMF* in terms of the number of model elements and effort measured in terms of time, 3) Correctness of the developed models using simulation with executable UML, and 4) Experience of applying UTP V.2 for creating test ready models.

The rest of the paper is organized as below. Section 2 presents the background, followed by a running example (Section 3). Section 4 presents the overview of *UMF*. Section 5 discusses details of the *UUP* profile and Section 6 discusses the model libraries. Section 7 presents the guidelines for applying *UMF*. Section 8 presents the evaluation and Section 9 presents the related work. We conclude the paper in Section 10.

2. BACKGROUND

2.1 Cyber-Physical Systems and Testing Levels

A CPS is defined in [2] 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 Figure 1. As defined in [2], uncertainty can occur at the following three logical levels (Figure 1): 1) *Application level*, due to events/data originating from the application of the CPS; 2) *Infrastructure level*, due to various interactions (e.g.,

[1] www.fpx.se/geo-sports/
[2] www.cisco.com/

events/data) among physical units, networking infrastructure, and/or cloud infrastructure, 3) *Integration level*, due to either interactions between components at the application level and ones at infrastructure levels, or interactions of uncertainties across these two levels.

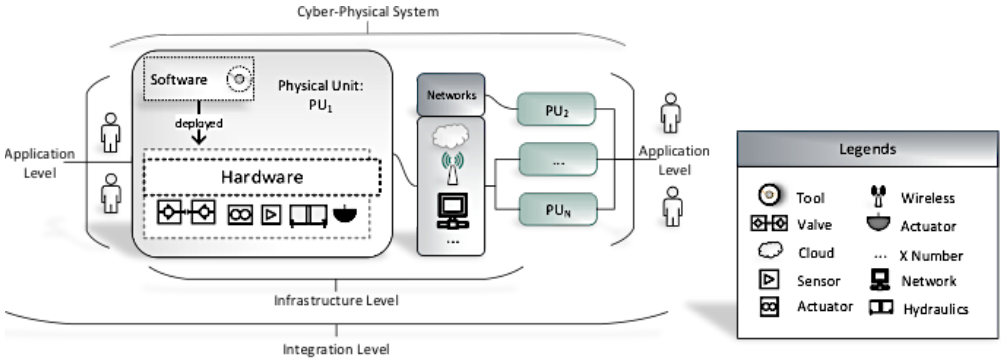

Figure 1. Conceptual model of a Cyber-Physical System [2]

2.2 U-Model

In our previous work [50], 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 further extended for supporting MBT of all the three levels of CPS under uncertainty (Section 2.1). *U-Model* was developed based on an extensive review of existing literature on uncertainty from several disciplines including philosophy, healthcare and physics, and two industrial CPS case studies from the two industrial partners of the U-Test project. In this paper, we implement *U-Model* as *UMF* (including *UUP* and a set of model libraries) to support the construction of test ready models with uncertainty. Details of *U-Model* is given in [50].

2.3 UML TESTING PROFILE

UML Testing Profile (UTP) [17] is the MBT standard at Object Management Group (OMG). With UTP, either System Under Test (SUT) is modeled using the UTP or test cases for the SUT are modeled using UML and UTP. Transformations from UTP models to executable test cases can be performed using specialized test generators. The version 2.0 of UTP is being under developed, where the second and third authors of the paper are the work group members. To enable MBT of CPSs under uncertainty, we rely on UTP V.2 to model the testing aspects of test ready models. UTP V.2 defines a set of stereotypes such as *Test Case*, *Test Data*, and *Test Design Model* and model libraries such as various types of test case *Verdict* (pass, fail).

3. RUNNING EXAMPLE

To illustrate *UMF* throughout the paper, we present a running example in this section. It is about a simplified security function of the SafeHome system described in [37]. The test ready models of the running example were developed as a class diagram, a composite structure diagram and a set of state machine using IBM Rational

Software Architect (RSA) 9.1 [24]. For the sake of simplicity, we only show one security function related to intrusion detection.

In general, the security system controls and configures the *Alarm* and related *Sensor*s through their corresponding interfaces (class diagram in Figure 2). In Figure 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 interface 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 Figure 6).

Behaviors of the system, alarm and sensors are specified as three state machines shown in Figure 8, Figure 5 and Figure 6 respectively. The alarm state machine has two states as shown in Figure 5: *AlarmDeactivated* and *AlarmActivated*. *AlarmDeactivated* represents the state that the alarm is not ringing, whereas the *AlarmActivated* state denotes that the alarm is ringing. The sensor state machine has two states (Figure 6): *SensorDeactivated* denoting 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 (Figure 8) 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* representing 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 (*Timer Stopped*), the timer is activated to wait for 3 minutes before notifying the police (*TimerStarted*), and the police is notified (*PoliceNotified*).

Table 1. Details on Transition «BeliefElement» IntrusionOccurred

Transition	***Normal→IntrusionDetected* (Figure 8, B.1)**
trigger*	<SignalEvent>*IntrusionOccurred(sensorID:String)*
effect*	*activateAlarm()* the body of this operation is as below: portSecurity.send(new StartAlarm())
UUP::Stereotype	**«BeliefElement»**
agent	«BeliefAgent» *SafeHomeSoftwareEngineer*
beliefDegree	Measurement -measureInDTViaClass: «BeliefMeasure»*ReceiveIntrusionOccurred* -measurementInVS:<InstanceValue>*VeryLikely*
Uncertainty	Uncertainty -kind: *UncertaintyKind::Occurrence* -referredCause: «IndeterminacySource» *notifyIntrusion* (the effect of *IntrusionDetected* transition, see Figure 6, **A**) -referredEffect: «BeliefElement, Effect» *AlarmActivated* (**C**)

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

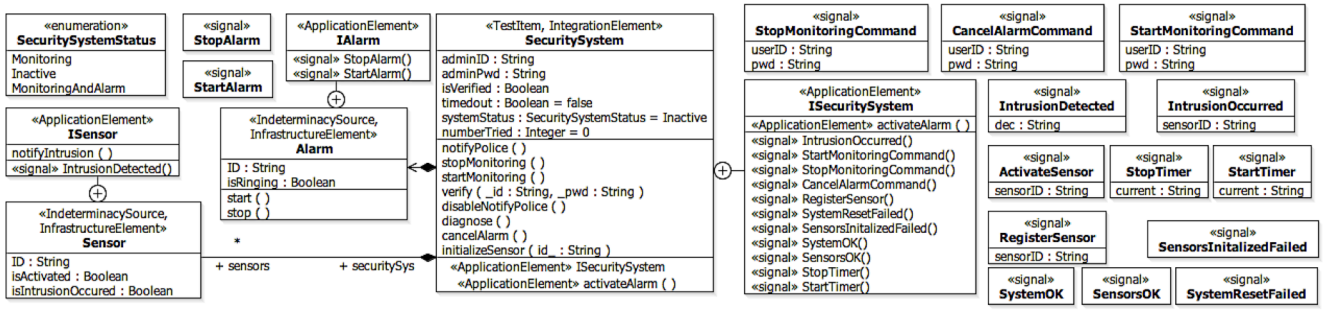

Figure 2. Class diagram of the Simplified *Security System*

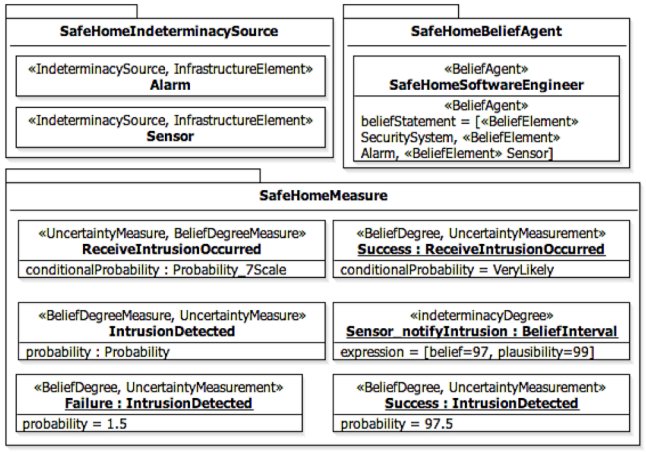

Figure 3. The Example of Applying UUP with Model Libraries

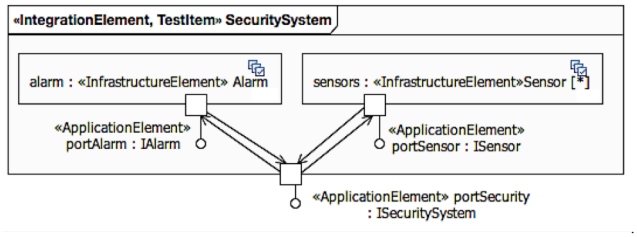

Figure 4. Composite Structure diagram of the *Security System*

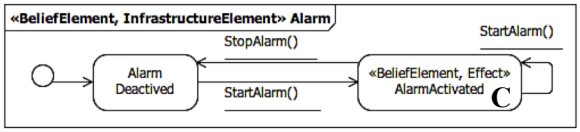

Figure 5. State Machine of *Alarm*

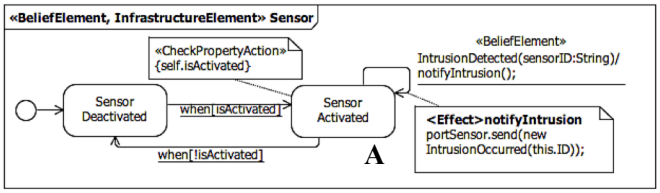

Figure 6. State Machine of *Sensor*

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)
```

Figure 7. StateInvariant (in OCL) of *IntrusionDetected* (B.2)

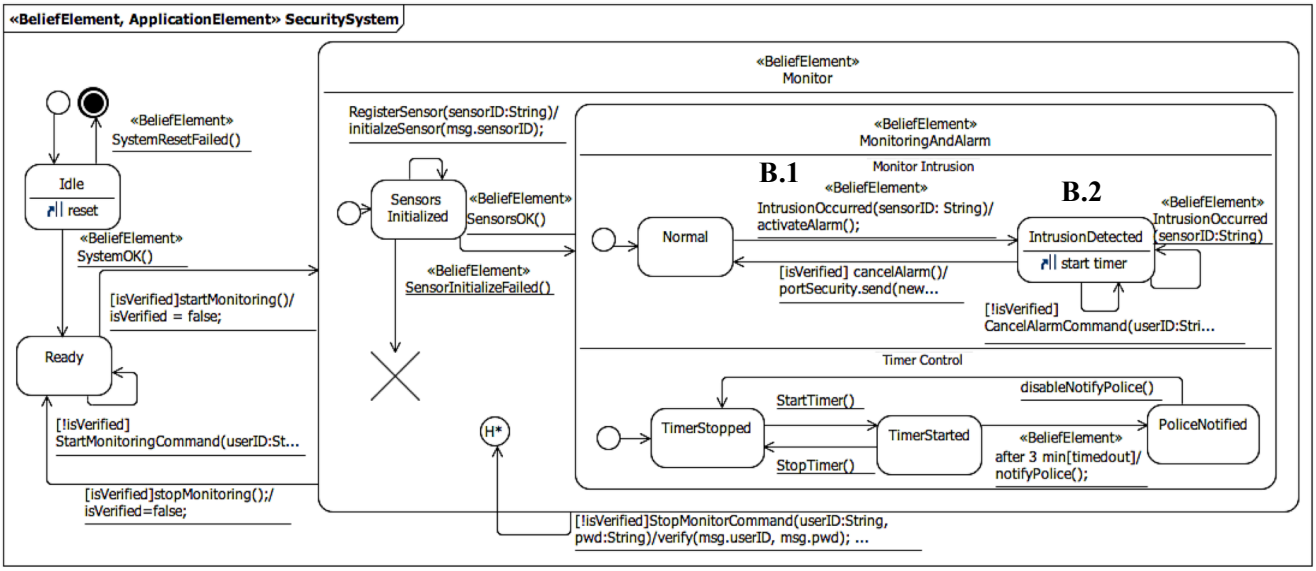

Figure 8. State Machine of the Security System

These three state machines communicate via signals using the ports defined in the composite structure (Figure 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 (UAL code in the comment in Figure 6) once the intrusion is detected via the effect *notifyIntrusion* defined in the self transition (Figure 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 (Figure 8, B.1). In parallel, as one can see from the bottom region (*Timer Control*) of the *MonitoringAndAlarm* composite state of the system (Figure 8), the system sends the *StartAlarm* signal to the *Alarm* state machine via effect *activateAlarm* (Figure 8 and effect* in the Table 1) and triggers the *StartTimer()* when entering *IntrusionDetected* state (Figure 8, B.2), which leads to the transition from *TimerStopped* to *TimerStarted* (Figure 8). From *TimerStarted*, after 3 minutes (time event), the system notifies the police and transits to *PoliceNotified*; 3) The *Alarm* state machine receives the *StartAlarm* signal in the *AlarmDeactivated* state and activates the alarm and transits to *AlarmActivated*.

To illustrate how to model uncertainty using *UUP* combined with Model library, CPS Testing Levels profile and UTP V.2, we also apply these profiles in the running example, and more details are presented in the following sections.

4. OVERVIEW OF UNCERTAINTY MODELING FRAMEWORK

An overview of *UMF* is presented in Figure 9. Notice that our framework is exclusively developed to develop test ready models with uncertainty to facilitate MBT of CPS under uncertainty. This means that *UMF* is not proposed for supporting modeling of CPSs and uncertainty from the design and development perspectives.

The core of *UMF* is to implement *U-Model* [50]. More specifically, the core part of *U-Model* is implemented as *UUP* comprising of three parts: *Belief*, *Uncertainty* and *Measurement*. All these profiles import the *Internal_Library* that defines the necessary Enumerations required in the profiles. The framework also consists of a small *CPS* profile that permits modeling specific aspects of the three testing levels of CPS, i.e., *Application*, *Infrastructure*, and *Integration*, exclusively for facilitating MBT.

The framework also consists of three UML model libraries: *Measure Library, Pattern Library*, and *Time Library* (that extend MARTE [18]). The framework relies on UTP V.2 to bring necessary MBT concepts to test ready models. Finally, the framework provides a set of guidelines to use its modeling notations to construct test ready models for enabling MBT of CPSs under uncertainty.

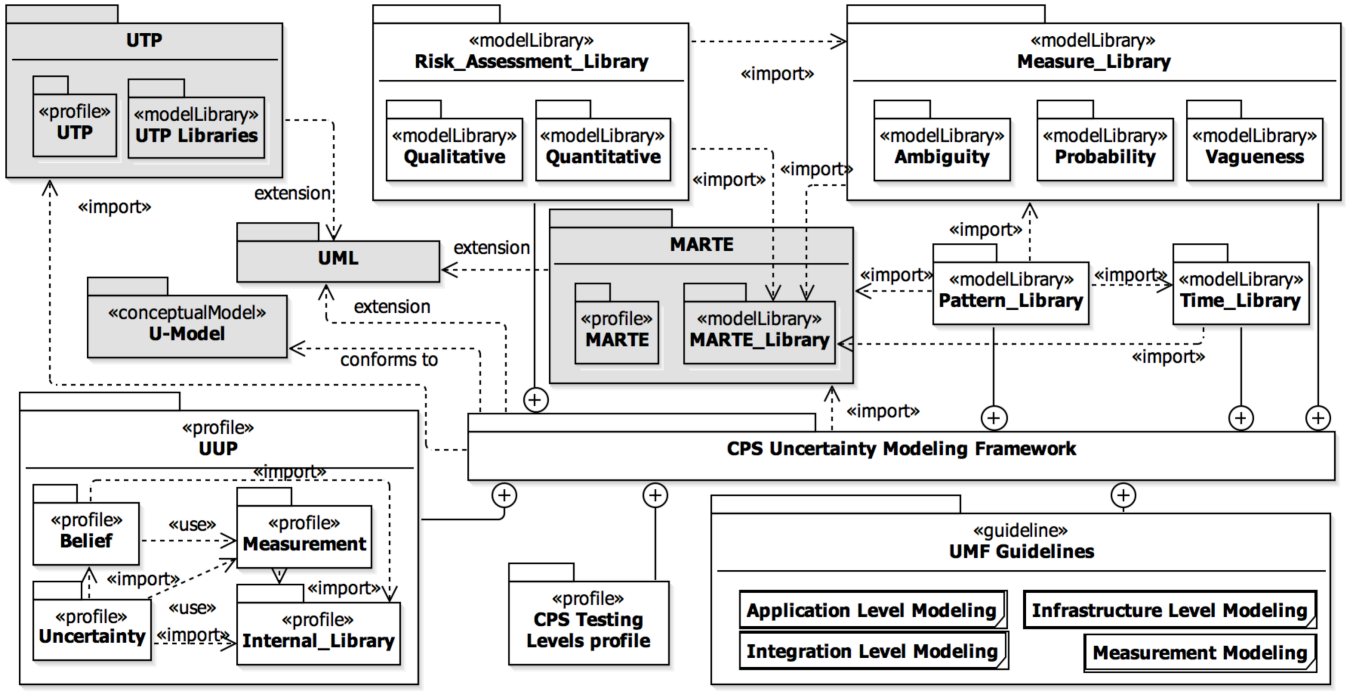

Figure 9. Overview of Uncertainty Modeling Framework (*UMF*)

5. UUP AND CPS TESTING LEVELS PROFILES

This section presents *UUP*, whose modeling notations are composed of stereotypes and classes for *Belief*, *Uncertainty* and *Measurement*, as shown in Figure 10, Figure 11 and Figure 12.The definitions of the profile modeling elements are provided in [49] for reference.

There are two approaches discussed in the literature for defining UML profiles [29]. One approach directly defines a UML profile for a specific purpose. The second approach is a more systematic as it starts with a conceptual model defining detailed concepts and relationships among them independent of any detail on for example which UML metaclass to extend. A UML profile is then developed based on the conceptual model. Note that not all the concepts defined the conceptual model are transformed into model elements in the UML profile. We opted for the second approach to create the *UUP* profile, as it has been used to define several well-known standardized UML profiles such as MARTE [18] and UTP V.2.

5.1 UUP Belief

The *Belief* part of UUP (Figure 10) implements concepts in *U-Model:BeliefModel* [50] (the mapping is provided in Table 2 and further discussion is provided in Section 8.1). As shown in Figure 10 and Table 2, five stereotypes are defined, among which «BeliefElement» is the key stereotype associated to various UML metaclasses. For example, a *StateMachine* (subtype of metaclass *Behavior*) itself can be a belief element such that «BeliefElement» can be applied on it to characterize it with additional information such as to which extent a modeler (stereotyped with «BeliefAgent») is confident about the state machine (i.e., *beliefDegree* of *BeliefStatement*), all uncertainties (i.e., *Uncertainty*) associated to the state machine, and their *Measurement*s. In the context of the U-Test project,

we extend UML state machines. However, it is worth mentioning that «BeliefElement» can extend other UML modeling notations such as activity and sequence diagrams if needed. We intentionally kept the profile generic such that different MBT techniques based on different diagrams can be defined when needed.

«Evidence» 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. Each uncertainty is also associated to a set of indeterminacy sources, which can be explicitly specified using «IndeterminacySource» and classified with enumeration *IndeterminacyNature* (Figure 10).

The profile also implements OCL constraints defined in *U-Model*. For example, as shown in Figure 10, each *beliefDegree* (an instance of *Measurement*) of a «BeliefStatement» must have exactly one measure associated to 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 below:

```
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, the belief agent (Figure 3) is *SafeHomeSoftwareEngineer* (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, the «BeliefElement» stereotype is applied *IntrusionOccurred* transition from state *Normal* to *IntrusionDetected* (Figure 8, B.1). The belief agent of this belief element is class *SafeHomeSoftwareEngineer* (stereotyped with «BeliefAgent» shown in Figure 3). 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 «IndeterminacySource» *notifyIntrusion* of *Sensor* (Table 1).

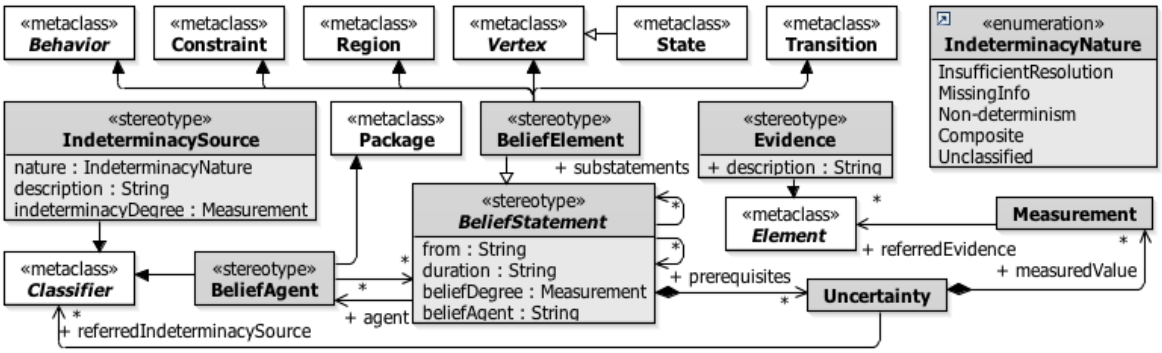

Figure 10. The *Belief* Profile

5.2 UUP Uncertainty and Measurement

The Uncertainty and Measurement parts of the *UUP* Profile are presented in Figure 11 and Figure 12. 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 *kind* and *field*) are optional. For example, an uncertainty might or might not have an indeterminacy source (i.e., *indeterminacySource*).

The *U-Model* concepts of *Effect*, *Pattern*, *Lifetime*, *and Risk* can be specified with *UUP* in difference ways. For example, one can specify the effect 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 associate to it (i.e., *referredEffect*). More details regarding the possible alternatives can be found in Section 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». The abstract stereotype of «Measurement» is further classified into five subtypes, corresponding to the five elements that need to be measured.

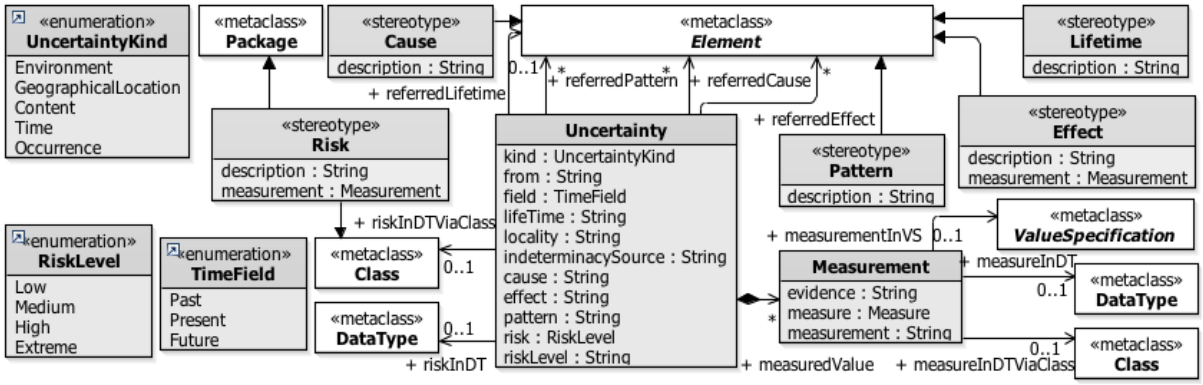

Figure 11. The *Uncertainty* Profile

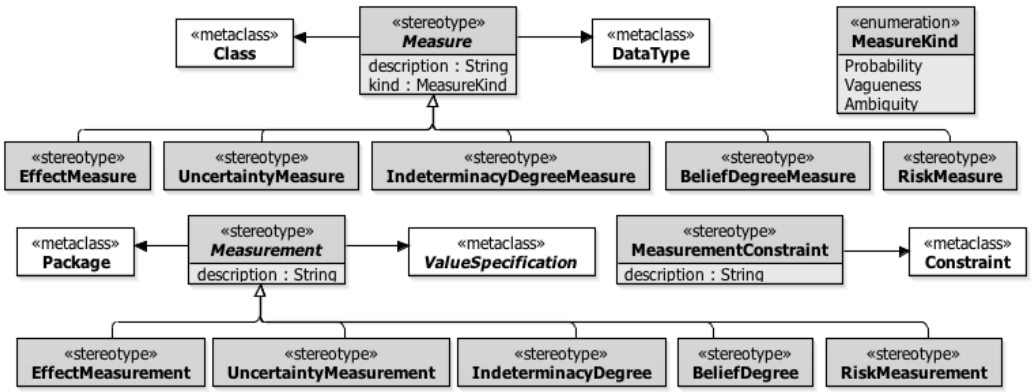

Figure 12. The *Measurement* Profile

Abstract stereotype «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 example is that Element with applied «Effect» stereotype 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 "referredEffect" attribute (Table 1).

Table 2. Definitions of the Stereotypes and Classes in *UUP*

Profile	**Stereotype/Class**	**U-Model** [50]	**Definition** (Most of concepts are defined in the U-Model [50])
Belief	«BeliefStatement»	BeliefModel:: BeliefStatement	As defined in [50], "A BeliefStatement is a concrete and explicit specification of some Belief held by a BeliefAgent about possible phenomena or notions belonging to a given subject area."
	«BeliefElement»		This stereotype is specialization of «BeliefStatement» that is more relevant in the context of UML and modeling in general.
	«BeliefAgent»	BeliefModel:: BeliefAgent	As defined in [50], "A BeliefAgent is a physical entity owning one or more Beliefs about phenomena/notion".
	«Indeterminacy Source»	BeliefModel:: IndeterminacySource	As defined in [50], "It represents a situation whereby the information required ascertaining the validity of a BeliefStatement is indeterminate in some way, resulting in Uncertainty being associated with that statement."
	Uncertainty	BeliefModel::Uncertainty	As defined in [50], "uncertainty is a state (i.e., worldview) of some agent or agency – henceforth referred to as a BeliefAgent – that, for whatever reason, is incapable of possessing complete and fully accurate knowledge about some subject of interest."
	Measurement	BeliefModel::Measurement	As defined in [50], "Measurement represents the result of measuring stated in the BeliefStatement related to the existing Measure."
	«Evidence»	BeliefModel::Evidence	As defined in [50], "Evidence is either an observation or a record of a real-world event occurrence or, alternatively, the conclusion of some formalized chain of logical inference that provides information that can contribute to determining the validity (i.e., truthfulness) of a BeliefStatement."
Uncertainty	«Cause»		Anything from which an Uncertainty occurs in the BeliefStatement.
	«Effect»	UncertaintyModel::Effect	Effect represents the result of Uncertainty in the BeliefStatement.
	«Lifetime»	UncertaintyModel:: Lifetime	As defined in [50], "Lifetime represents an interval of time, during which an Uncertainty exists".
	«Risk»	UncertaintyModel::Risk	The risk associated with the uncertainty
	«Pattern»	UncertaintyModel::Pattern	A pattern represents a particular pattern in which an uncertainty can occur.
Measurement/ Measure	«Measurement»	BeliefModel::Measurement	Please see the definition of Measurement under Belief Category
	«BeliefDegree»	"beliefDegree" attribute of Belief	As defined in [50], "This Measurement is used for representing confidence degree from BeliefAgent held this Belief."
	«Indeterminacy Degree»	"indeterminacyDegree "attribute of IndeterminacySource	As defined in [50], "This set of Measurement represents the quantification (or qualification) of this IndeterminacySource."
	«EffectMeasurement»	"measurement" attribute of Effect	This value is used for representing what kind of measurement may be used to measure the Effect.
	«RiskMeasurement»		This value is used for representing what kind of measurement may be used to measure the Risk
	«Uncertainty Measurement»	"measuredValue" attribute of Uncertainty	As defined in [50], "This value is used for representing confidence degree of uncertainty by the agent making the BeliefStatement."
	«Measure»	MeasureModel::Measure	As defined in [50], "Measure is an objective concept specifying method of measuring uncertainty."
	«BeliefDegree Measure»	"measure" attribute of Measurement	BeliefDegreeMeasure represents the measure specifying method of measuring belief degree.
	«IndeterminacyDegree Measure»	"measure" attribute of Measurement	IndeterminacyDegreeMeasure represents the objective concept specifying method of measuring indeterminacy degree.
	«RiskMeasure»		RiskMeasure represents the objective concept specifying method of measuring risk.
	«UncertaintyMeasure»	"measure" attribute of Measurement	UncertaintyMeasure represents the objective concept method of measuring uncertainty.
	«EffectMeasure»	"measure" attribute of Measurement	EffectMeasure represents the objective concept specifying method of measuring effect.

5.3 CPS Testing Levels Profile

We define a small CPS Testing Levels profile with the three stereotypes, as shown in Figure 13, to denote which model element belongs to which of the three CPS testing levels: *Application*, *Infrastructure* and *Integration*. All the three stereotypes extend UML metaclass Element, as one can apply them to a class, a state, a state machine and many other model elements.

It is important to point it out that this CPS profile is defined for the purpose of 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 Figure 2 is stereotyped with «IndeterminacySource» and «InfrastructureElement», meaning that sensors are infrastructure elements. As shown in Figure 4, the composite structure of the system describes the interactions between the infrastructure elements (*alarm* and *sensors*) and the application level elements: *portSensor*, *portAlarm* and *portSecurity*, which are typed by three interfaces (i.e., *ISensor*, *IAlarm* and *ISecuritySystem*) as shown in Figure 2. This is the reason that the composite structure is stereotyped as «IntegrationElement».

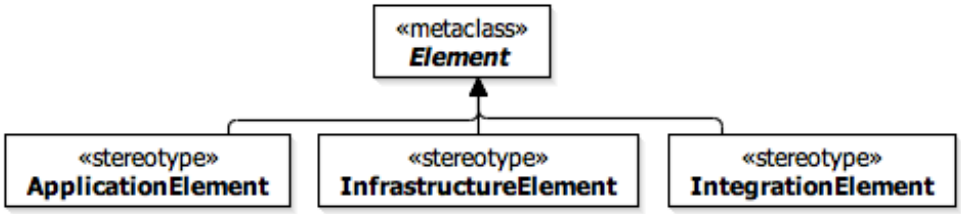

Figure 13. The CPS Testing Level Profile

6. MODEL LIBRARIES

To model uncertainty with advanced modeling features, we define three model libraries (Figure 14, Figure 15, Figure 16) that can be used together with *UUP* (Figure 9): model libraries for modeling uncertainty *Pattern*s, uncertainty *Measurement*s (corresponding to *Probability*, *Vagueness*, and *Ambiguity*) and *Time* related properties. *Measure*, *Pattern* and *Time* libraries import the MARTE_PrimitiveTypes library [18] to faciliate the expression of data in the domain of CPSs, e.g. *Real*. Respectively, the *Measure* library adapts the operation of NFP_CommonType of MARTE [18] related to probability distributions. The *Pattern* library imports elements related to *Pattern* from the BasicNFP_Types library of MARTE [18] (e.g., AperiodicPattern) and further extends them. For example, the *Transient* pattern does not exist in MARTE [18] and has been newly defined. The *Time* library imports the MARTE_DataTypes library [18] to facilate the expression of time, e.g., *Duration* and *Frequency*.

6.1 Measure Libraries

We define the three measure packages (*Probability*, *Ambiguity*, and *Vagueness*) to facilitate modeling different uncertainty measures as shown in Figure 9.

In the *Ambiguity* library, we define the data types corresponding to the relevant *Ambiguity* measures published in the literature as shown in Figure 14. Since these measures are well-known, we do not provide further details in this paper; however, interested readers may consult the provided references for more details. The *BeliefInterval* is adopted from [5], *Belief* from [5], *Plausibility* from [5], *Conflict* from [40], *ShannonEntropy* from [5], *HartleyMeasure* from [20], *AlternativeMeasure* from [30], *DissonanceMeasure* from [43], *U_Uncertainty* from [21], *PossibleDistribution* from [47], and *PignisticDistribution* from [42]. Some further details of these data types including their attributes are provided in the technical report corresponding to this paper [49].

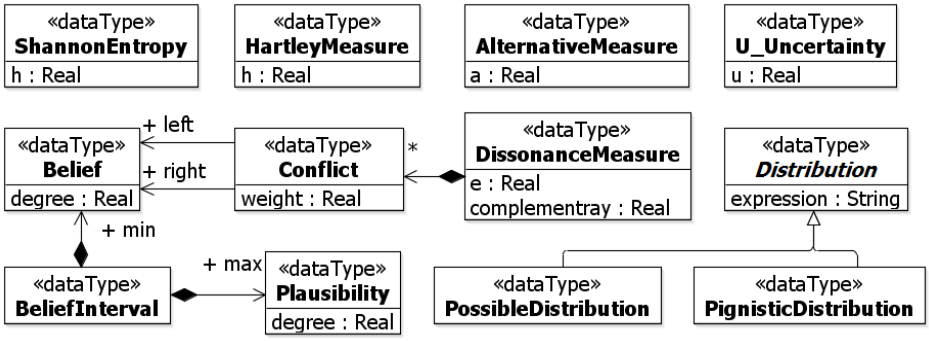

Figure 14. The Ambiguity Model Library

The concepts of the fuzzy theory [45] are defined in the *Vagueness* library (Figure 15). *HedgeKind* is adopted from [35], *MembershipDegree* and *FuzzySet* (*FuzzySetOperationKind*, *FuzzyLogicOperation*, *FuzzyLogic* and *FuzzyNumber*) from [45], *FuzzySetCut* from [11], *FuzzyEntropy* from [28], *Fuzziness* (further classification *EuclidFuzziness*, *HammingFuzziness*, and *MinkowskiFuzziness*) from [6; 51], *Roughness* and *RoughSet* from [36], *LFuzzySet* from [12], *IntuitionisticFuzzySet* from [3], *IntervalValuedFuzzySet* from [14; 26; 46], *VagueSet* from [10], and *Sharpness* from [4]. For example, as shown in Figure 3, the *IndeterminacyDegree* of *SensorNotifyIntrusion* which is used to measure the occurrence of *notify Intrusion*, the effect of self-transition of *SensorActivated* (Figure 6), is expressed by BeliefInterval[3] [5] that is composed of belief (97%) and plausibility (99%), which are pre-defined in the *Ambiguity* library (Figure 14). Some further details of these data types are provided in the technical report corresponding to this paper [49] and interested readers may consult the provided references for more details.

[3] This concept borrows from *Dempster-Shafer* is used to specify belief interval that is a boundary of probability.

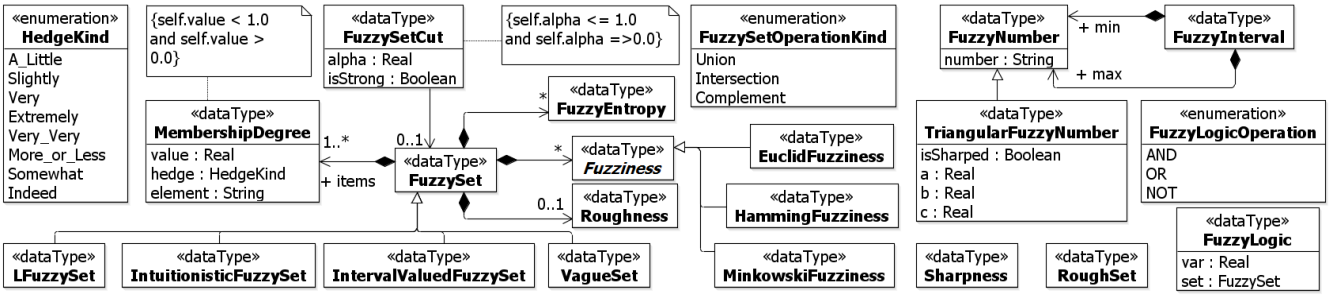

Figure 15. The Vagueness Model Library

Various data types related to the probability are defined in the *Probability* library (Figure 16). 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 [49]. 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.

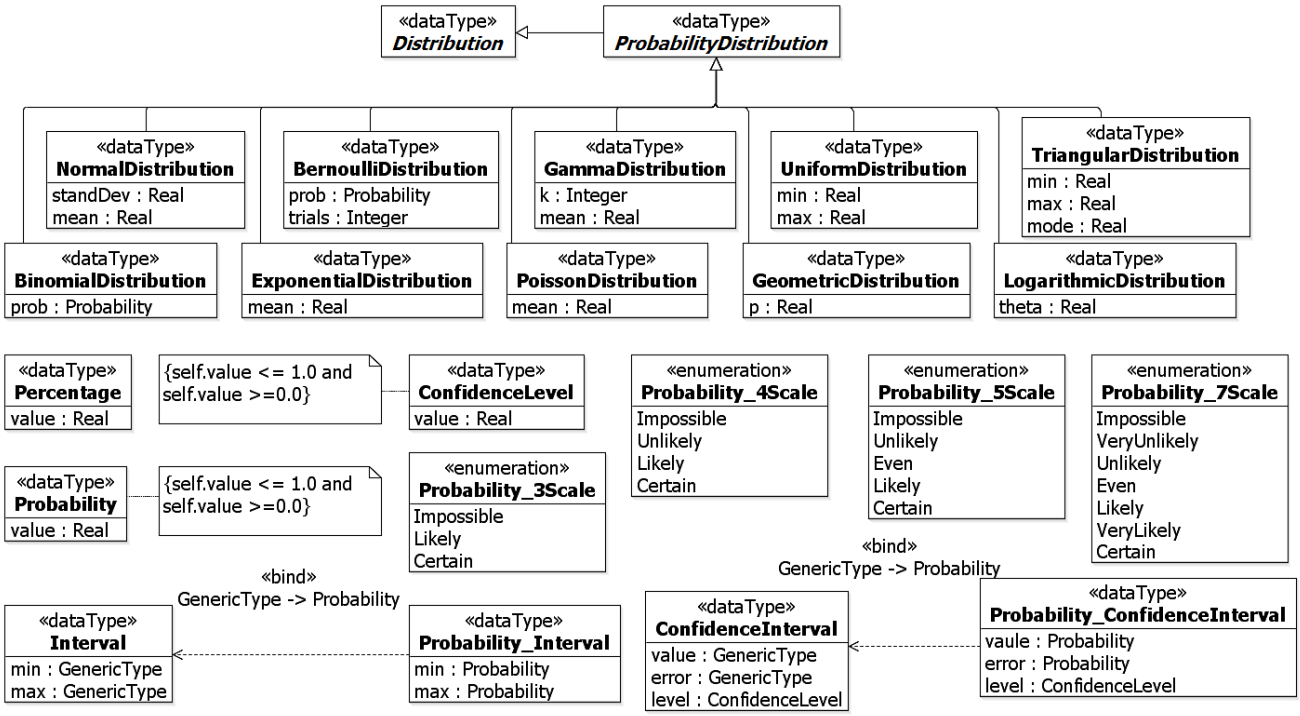

Figure 16. The Probability Model Library

6.2 Pattern Library

This section presents the *Pattern Library* shown in Figure 17. All the patterns except *Transient and PersistentPattern* are imported from MARTE [18] . Details of these patterns can be consulted from the MARTE specification and the technical report corresponding to this paper [49]. The definition of transient is adopted from

[50], i.e., *"Transient is the situation whereby an uncertainty does not last long"*. *Transient* inherits from *IrregularPattern*. The newly introduced attributes are: *minDuration* and *maxDuration* describing the duration for which the uncertainty lasts. The definition of *PersistentPattern* is adopted from [50], 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.

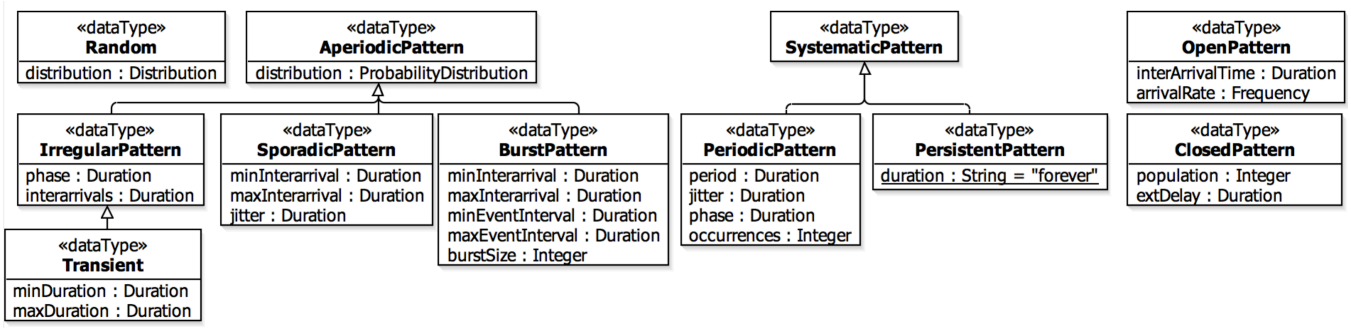

Figure 17. Pattern Library

6.3 Time Library

This section presents the necessary time concepts borrowed from the MARTE Library [18]. An overview diagram is shown in Figure 18. Some further details of these data types are provided in the technical report corresponding to this paper [49] and interested readers may also consult the MARTE specification [18].

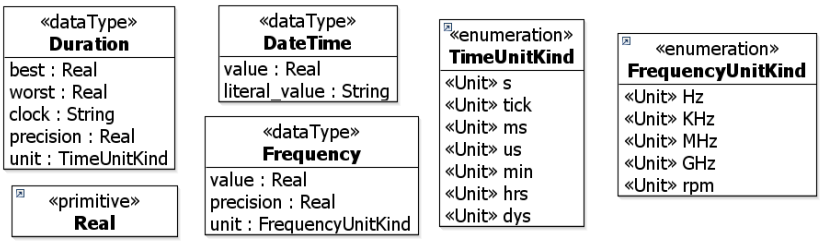

Figure 18. Time Library

7. UMF MODELING METHODOLOGY

In this section, we present a modeling methodology for the *U-Model* notations. The rest of this section is organized as follows: Section 7.1 presents the overview of modeling activities, Section 7.2 presents modeling activities at *Application Level*, Section 7.3 presents modeling activities at *Infrastructure Level*, Section 7.4 presents modeling activities at *Integration level*, and Section 7.5 presents the modeling activities of *applying UUP* which is invoked at above three level.

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 Figure 19. 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.

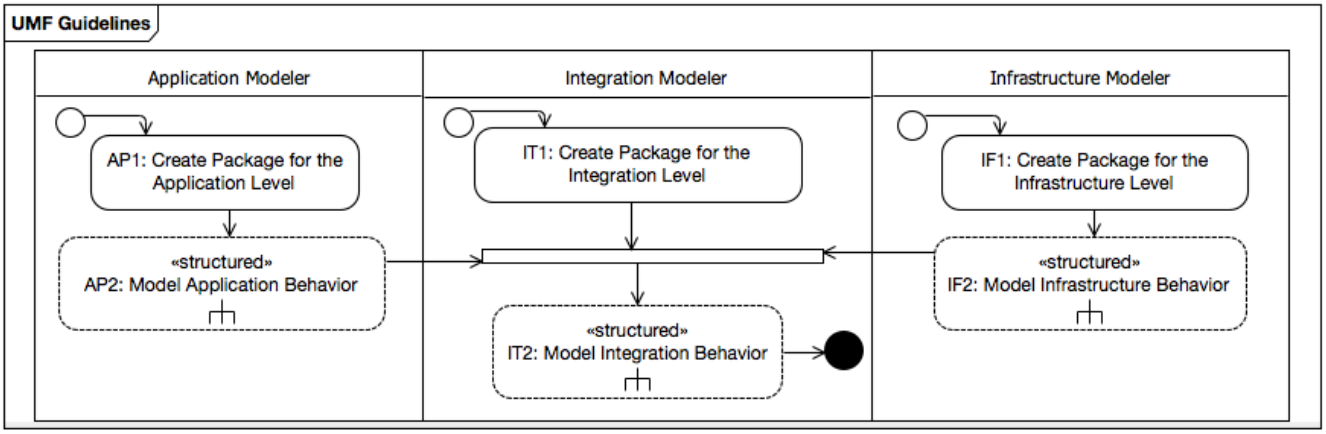

Figure 19. The Top Level Guideline

As shown in Figure 19, 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. Next, application and infrastructure modelers apply the *U-Model* 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 Sections 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 Section 7.4.

7.2 Application Level Modeling

The application level modeling activities include three sequential steps: creating application level class diagrams (AP2.1), creating application level state machines (AP2.2) and applying the *UUP* notations on the created class and state machines (AP2.3).

A class diagram created for the application level should capture domain concepts that are needed for specifying the API information to gain access to the data and behavior of the system. 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 objects. 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 (Section [49]) or MARTE_Library [18]. 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 the API of the application software or an action physically performed by an operator (e.g., switching on or off of an X4 device).

Each state in a state machine created for the application level is defined, with an OCL constraint specifying its state invariants. 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 have its guard condition modelled as an OCL constraint on the input parameters of the trigger of the transition.

Next, application modelers need to apply *UUP* on state machines (AP2.3) to specify uncertainties and apply the UTP profile to add testing information (e.g., indicating *TestItem*). The application of the UUP is the same for the three levels and thus we only it under the Integration Level Modeling section (Section 7.4.).

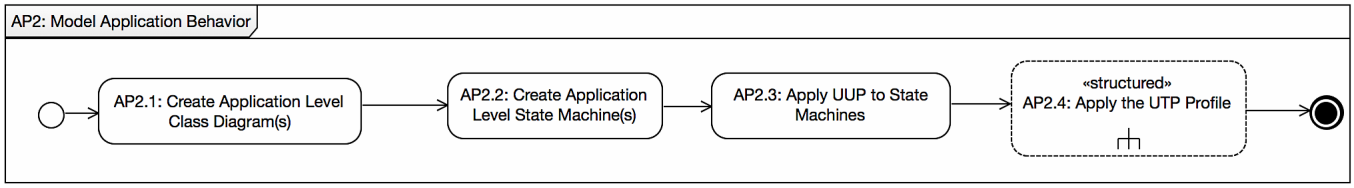

Figure 20. Application Level Guideline

7.3 Infrastructure Level Modeling

For the infrastructure level, a similar modeling procedure as the one defined for the application level should be followed to derive class diagrams and state machines, apply *UUP* and the UTP profile (further details in Section 7.4), as shown in Figure 21. One difference is that attributes of the infrastructure level class diagrams should capture observable infrastructure attributes. For example, an attribute can be the percentage of data loss between an X4 device and the Radio Antenna. Operations of the 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.

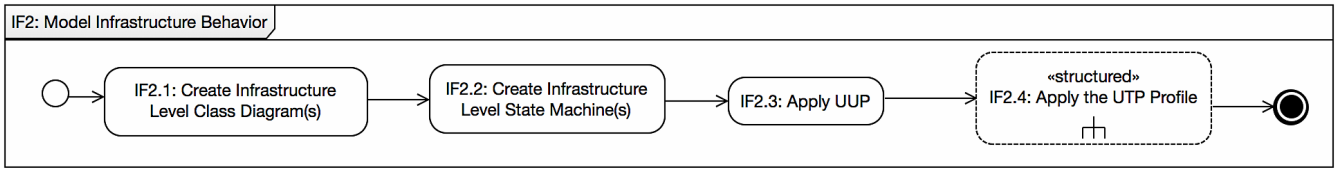

Figure 21. Infrastructure Level Guideline

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 Figure 22, the IT2 activity starts from creating integration level class diagrams (IT2.1) and state machines (IT2.2), followed by applying *UUP* and the UTP profile.

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. Each signal reception corresponds to an instance of UML *Signal* defined in a created integration level class diagram. Notice that creating class diagrams for the integration level is not mandatory. 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 the 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.

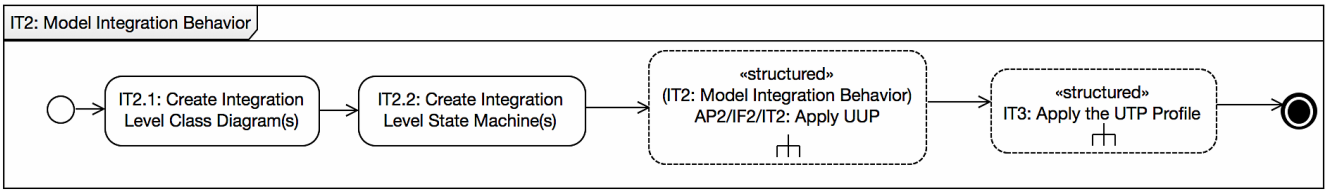

Figure 22. Integration Level Guideline

7.5 Apply UUP (AP2/IF2/IT2)

Recall that the activity of applying UML Uncertainty Profile (*UUP*) is invoked at all the three levels. We tag each type of activities of the activity diagrams from Figure 23 to Figure 31 with S, C and A to represent structured activities, call behavior and normal activity nodes. As shown in Figure 23, applying *UUP* starts from applying the «BeliefElement» stereotype on any allowed state machine model element according to *UUP*. Then a modeler can optionally specify values for the "from" and "duration" attributes of the stereotype, model belief agents, model belief degree, and/or model uncertainties (Figure 23).

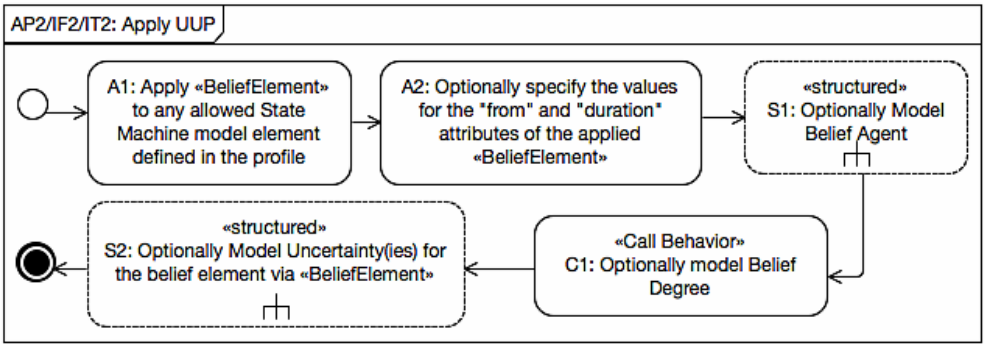

Figure 23. Applying UUP

As shown in Figure 24, there are two ways 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 create a package to organize all the belief agents (S1.2). In this case, each belief agent can be modelled 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).

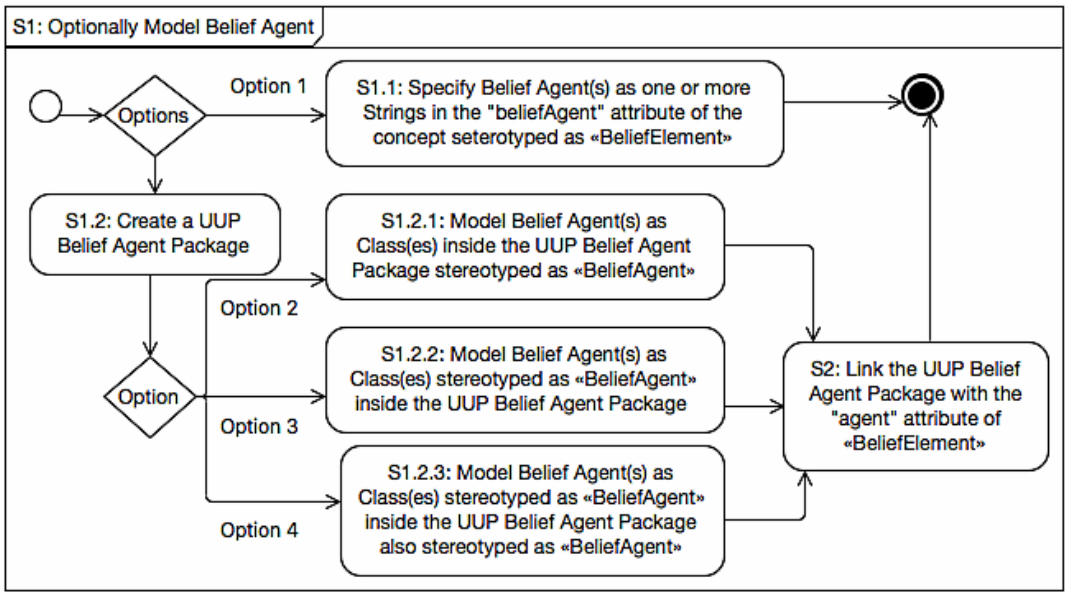

Figure 24. Model «BeliefAgent»

Modeling BeliefDegree is presented in Section 7.5.1 and modeling uncertainties is discussed in Section 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 Figure 25, 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 optionally specify Evidence (S1), followed by the specification of each measurement instance and its corresponding measure (S3 and S2).

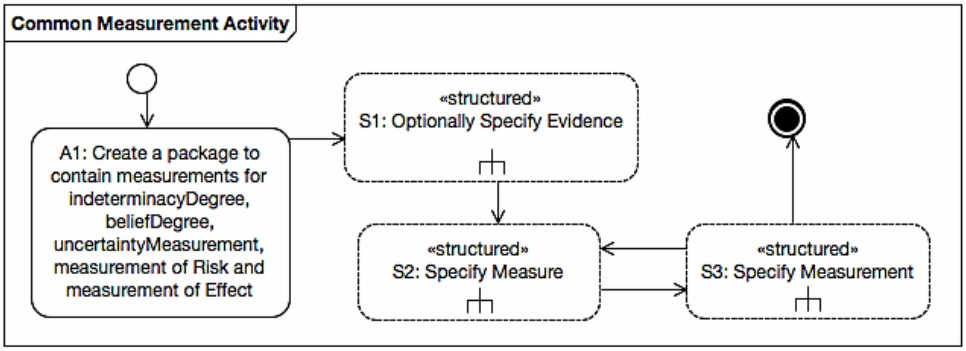

Figure 25. Common Measurement Modeling Activity

A. Specify Evidence

As shown in Figure 26, there are two ways 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 representing evidence to the "referredEvidence" attribute of Measurement (S1.2.3).

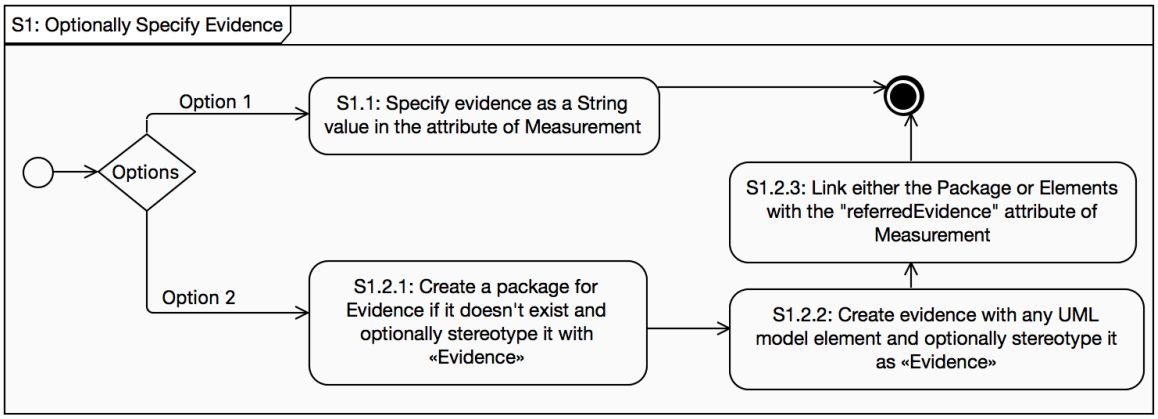

Figure 26. Specify Evidence

B. Specify Measure

As shown in Figure 27, 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 datatypes (A2). One

then needs to add attributes to these classes or datatypes by using the datatypes defined in the Measure Libraries. 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).

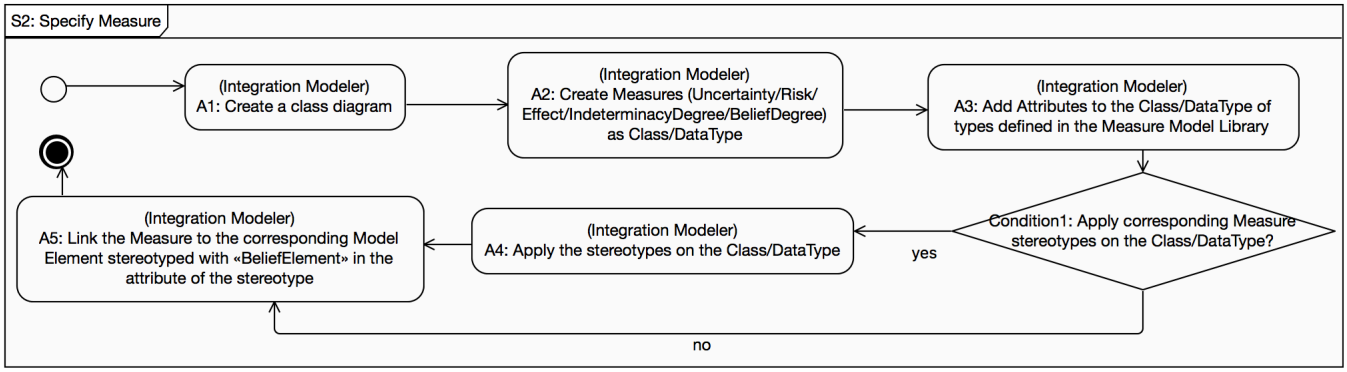

Figure 27. Specify Measure

C. Specify Measurement

There are three ways to specify measurements, as shown in Figure 28: 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).

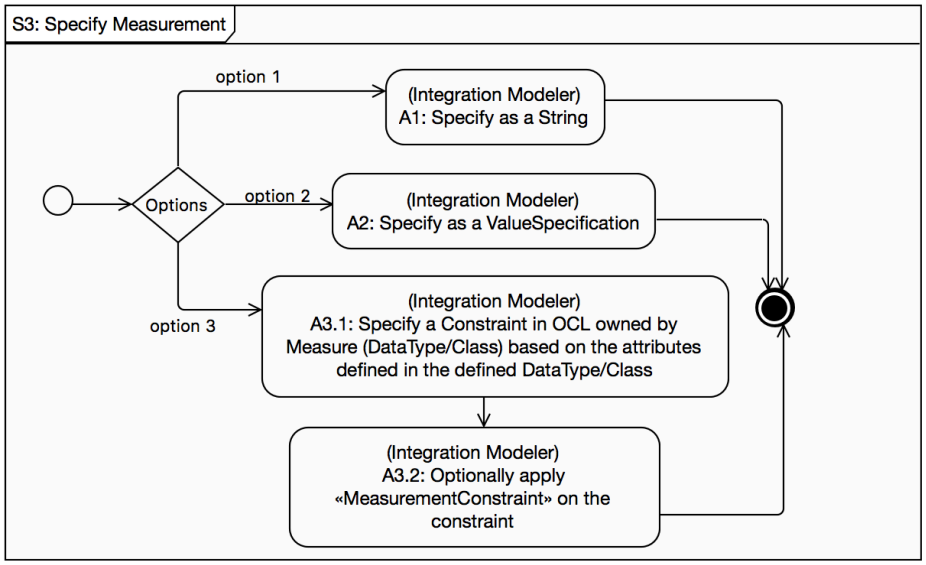

Figure 28. Specify Measurement

7.5.2 Uncertainty Modeling

As shown in Figure 29, 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.

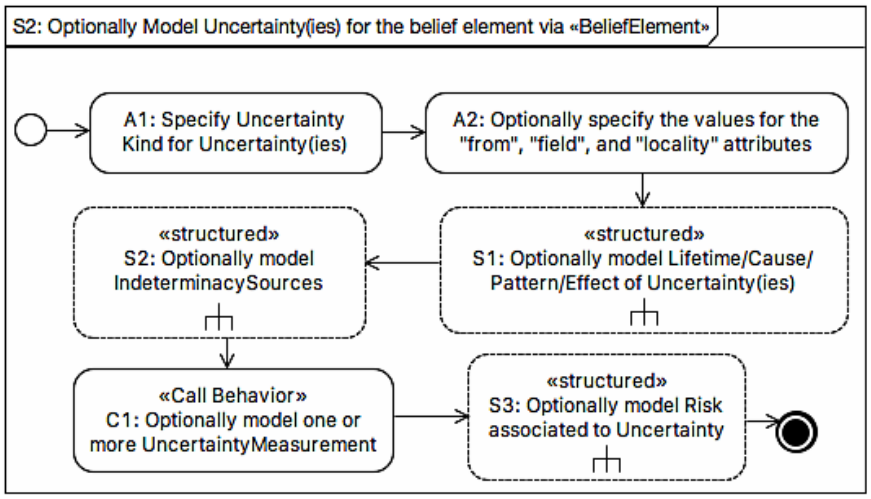

Figure 29. Model Uncertainty(ies)

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

A modeler has two options to specify Lifetime/Cause/Pattern/Effect of an uncertainty, as shown in Figure 30. 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 (Section 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".

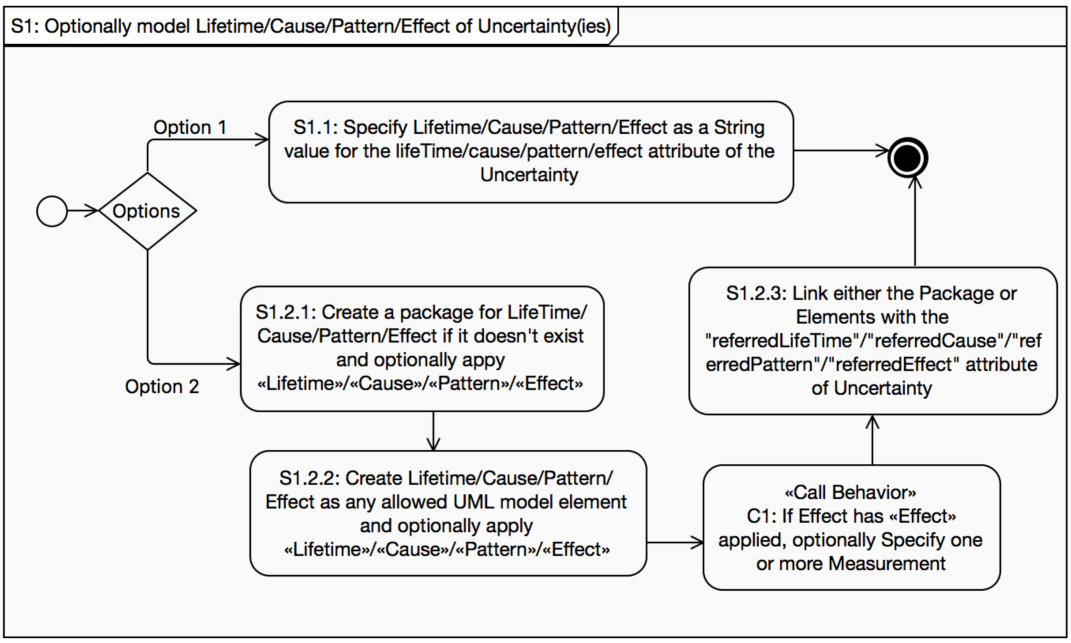

Figure 30. Model Lifetime/Cause/Patten/Effect of Uncertainty

B. Model IndeterminacySource

As shown in Figure 31, a modeler can simply specify an indeterminacy source as a String value of attribute "indeterminacySource" of Uncertainty (Option 1). Alternatively, one can create a package 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.

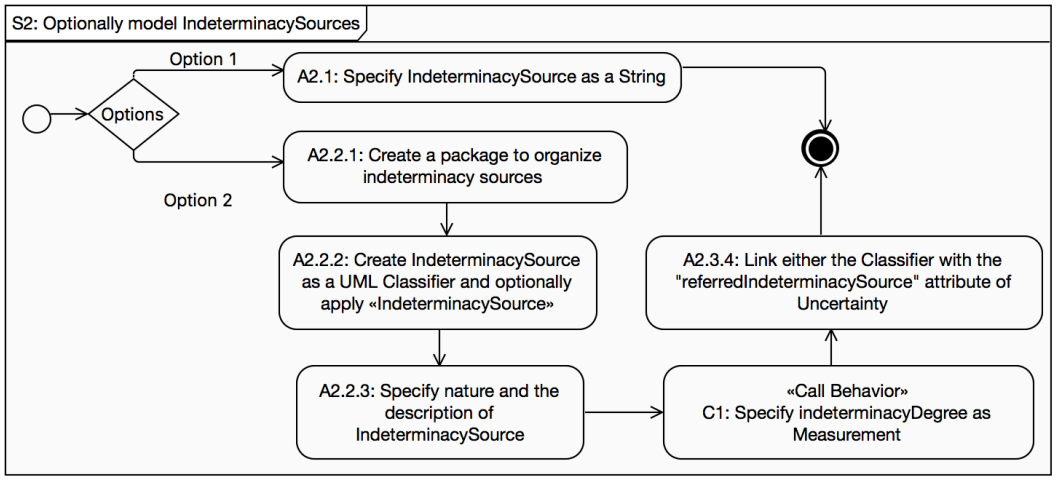

Figure 31. Model IndeterminacySource

C. Model Risk

A modeler can also optionally associate an uncertainty to Risk. As shown in Figure 32, 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 datatypes to represent Risks and optionally applying «Risk» (A4.3.2). Afterwards, 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).

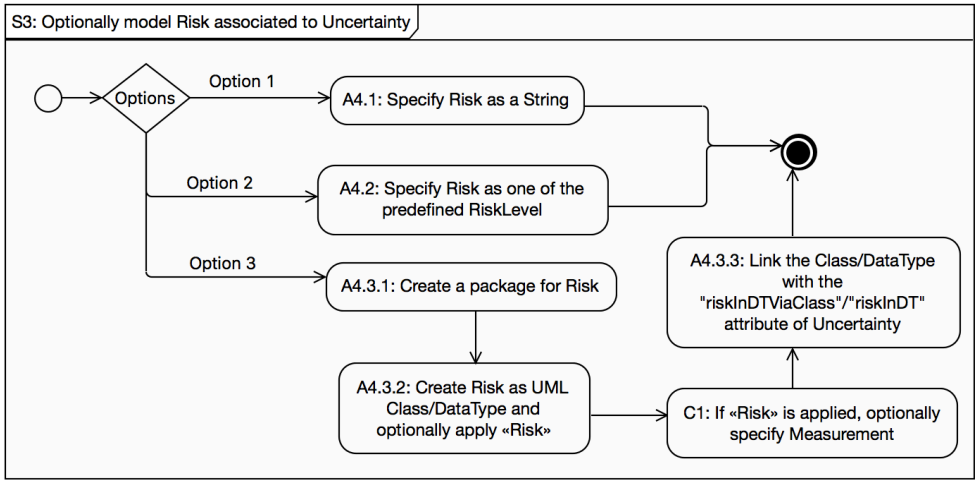

Figure 32. Model Risk

8. EVALUATION

For evaluating *UMF*, we used three case studies. The first case study is a modified version of the SafeHome case study provided in [37]. This case study implements various security and safety features in smart homes including intrusion detection, fire detection, and flooding. We extended the original case study by defining a set of executable UML models. The second case study is a Videoconferencing System (VCS) developed by Cisco, Norway. This case study has been used in our previous work [1] and was extended for evaluating *UMF*. The third case study is GeoSports provided by Future Position X (FPX) Sweden as part of the U-Test project. Some descriptive statistics of the test ready models developed for these case studies are provided in Table 3. 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 modeled for each case study.

Note that the second and third authors of the paper are working with the working group of UTP V.2 and have implemented it in IBM RSA. This implementation was used to model the case studies.

Table 3. Characteristics of the Case Studies

	CPS Profile	**Class Diagram**	**State Machine**	***Total***
SafeHome	# of diagrams	2	15	**17**

	# Application Elements	15	10	
	# Infrastructure Elements	16	19	
	# Integration Elements	3	3	**66**
	# Uncertainties	9	10	**19**
VCS	# of diagrams	6	12	**18**
	# Application Elements	92	59	
	# Infrastructure Elements	103	67	
	# Integration Elements	51	70	**442**
	# Uncertainties	41	83	**124**
GeoSports	# of diagrams	3	4	**7**
	# Application Elements	31	99	
	# Infrastructure Elements	36	34	
	# Integration Elements	6	20	**226**
	# Uncertainties	12	29	**41**

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

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

Table 4 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 MARTE. For example, 10 data types in the *Measure* library can be mapped to MARTE.

As we can see from Table 4 (last row), 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 are newly introduced by extending *U-Model* concepts. 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 4 shows 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* [27], but are not required to be implemented in *UUP* and the model libraries.

Table 4. Mapping UUP/Model Libraries to U-Model and MARTE

		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.*

8.2 Application of *UUP*/Model Libraries

We discuss the application of *UUP*/Model Libraries 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 three case studies, and 2) the effort required to apply *UUP*/Model Libraries.

As shown in Table 5, for the SafeHome case study, in total we modeled 21 classes in the class diagrams, seven out of which have *UUP* stereotypes applied (e.g., the «IndeterminancySource» sensor is applied to *Sensor*, see Figure 2). For the modeled state machines, three states out of 17 require the application of *UUP*/Model Libraries, whereas seven out of 29 transitions required 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%) of the modeling elements for the VCS (GeoSoports) case study required the application of *UUP*/Model Libraries. For all the three case studies, on average 16% of the model elements require applying *UUP*/Model Libraries.

Table 6 summarizes effort (measured in time (hours)) spent on constructing the test ready models (i.e., UML class diagrams, state machines, with *UUP*/Model Libraries applied) for the three 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 6, for *SafeHome*, it took us 4.5 hours for modeling UML class diagrams, whereas additional 0.5 hour was spent on applying *UUP/Model Libraries*. For UML state machines, it took 22.5 hours, whereas additional 7.5 hours were spent on applying *UUP/Model Libraries*. For SafeHome, as shown in the last column (%Time) of Table 6, it took additional 22% of time to apply *UUP*/Model Libraries.

Similarly, for VCS it took additional 23% of time and 15% of additional time for GeoSports. On average for all the three case studies together, on average modeling with *UUP/*Model Libraries required additional 20% of the total modeling effort.

Table 5. Percentage of *UUP*/Model Libraries Concepts to UML Concepts

Case Study	Class Diagram		State Machine		%
	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
Average Percentage: 16%					

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

Table 6. Effort (Time in Hours) of Applying *UUP*/Model Libraries

Case Study	Class Diagram		State Machine		% Time
	UML (hr.) Modeling	UUP/Model libraries (hr.) Modeling	UML (hr.) Modeling	UUP/Model libraries (hr.) 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
Average Percentage of Time: 20%					

8.3 Verification of Test Ready Models via Simulation using Executable UML

In this section, we present the results of the verification of the test ready models developed with *UMF* for the three 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 the UML Action Language (UAL, a implementation of the Action Language For Foundational UML, Alf [16])—a formal language supported in IBM RSA [24] for executing UML models implemented in Java. UML models with UAL can be executed with the IBM RSA Simulation Toolkit [25].

Table 7 shows the results of the verification. We classified identified problems during the verification process into two main categories: Incorrect and incomplete model elements for each case study. In Table 7, we classified the model elements into two categories states and transitions. For *State*, more specifically, we report problems identified in state invariants and «BeliefElement». For *Transition*, more specifically, we report problems identified in *Guard*, *Trigger*, *Effect*, and «BeliefElement». For *State*, in total, 56 problems (10+46) were identified across the three case studies, where 10 problems were related to *Incorrectness* and 46 were related to *Incompleteness*. For «BeliefElement» related to *State*, we identified 24 missing stereotypes. For transition, we discovered 85 problems (9+76), where 9 problems were related to *Incorrectness* and 76 problems were related to *Incompleteness*. For «BeliefElement» related to *Transition*, we identified 25 missing stereotypes.

The typical problems identified included: 1) transition happening between the states without any event, 2) even after firing a transition the state change didn't happen or state changed to an unexpected one, 3) failure to send signals across concurrent state machines, 4) non-deterministic transitions from a state, 5) unexpected exit, block, or deadlock observing in a state machines, 6) unreachable states, 7) guard conditions that are always true. Notice that these problems are not comprehensive set of problems, but provide most commonly observed problems. After simulating the test ready models, we ensure that our models are correct and complete and hence can be used for facilitating MBT.

Table 7. Results of Verification of Test Ready Models

Case Study		State		Transition			
		StateInvariant	«BeliefElement»	Guard	Trigger	Effect	«BeliefElement»
Incorrect	SafeHome	1	0	0	0	1	0
	VCS	6	0	0	5	0	0
	GeoSports	3	0	2	1	0	0
Incomplete	SafeHome	5	2	0	7	2	3
	VCS	30	13	15	23	21	18
	GeoSports	11	9	2	4	2	4
Total		**56 (10, 46)**	**24**	**85(9, 76)**			**25**

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

8.4 Application of UTP V.2

Applying UTP V.2 is the last step of *UMF* modeling as shown in Figure 22 to facilitate testing. In the running example, the «TestItem» stereotype from the *Test Context* package of UTP V.2 is applied on SecuritySystem as shown in the Figure 2 and «CheckPropertyAction» from the *Arbitration Specification* package of UTP V.2 is applied to the StateInvariant of *IntrusionDetected* (Figure 8) shown in Figure 7.

Table 8 reports the results of the application of UTP V.2 to the models of the case studies. Notice that we only report the statistics for the high level packages (e.g., Arbitration Specification) of UTP V.2 instead of providing the number of applications of each stereotype due to space limitations. Notice that each high level package contains a set of related stereotypes. For SafeHome, in total 54 stereotypes from UTP V.2 were applied, whereas 551 for VCS, and 209 for GeoSports.

Based on our experience of applying UTP V.2, we discovered that it is a generic UML profile for MBT and does not cater for 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.

Table 8. Application of UTP V.2

Category	SafeHome	VCS	GeoSports
Arbitration Specification	20	246	92
Test Data	29	278	106
Test Configuration	2	15	7

Test Context	3	12	4
Total	**54**	**551**	**209**

9. RELATED WORK

There are some works in the literature that attempt to deal with modeling uncertainty with UML. For example, the authors of [41] 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 weren't 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 [31; 32; 44], 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 [33] 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 [48] to transform fuzzy UML data models into web ontologies to support automated reasoning on fuzzy properties in the context of web services.

In [19], the UML profile (named as fuzzy UML) was proposed to model uncertainty on use case diagrams, sequence diagrams, and state machines. Another work in [34] formalizes UML state diagrams with fuzzy information and transforms them into fuzzy petri nets for supporting automated verification and performance analysis. In [13], 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 to these works, our *UMF* 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 [31; 32; 44] for specifying fuzzy UML data can easily integrated to our model libraries when needed. Notice that *UMF* is proposed to explicitly capture 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 [9] is the closest to our work, where uncertainty in time is modeled in UML sequence diagrams applied with the UML-SPT profile [15]. 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, *UMF* covers other types of uncertainties, in addition to time, such as content and environment. Moreover, the work doesn't cater for sources of time uncertainties that are essential to be explicitly captured in order to introduce uncertainties for test execution.

In [38], 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 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 dose not support other types of uncertainty measures such as ambiguity as supported in *UMF*. In addition, the work only supports modeling application level uncertainties and cannot be used to model uncertainties in the other two CPS levels as *UMF*.

In [39], 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 [7; 8]. The approach was demonstrated on UML class and sequence diagrams [39]. This work is related to our work in terms of expressing 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 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.

10. 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 (*UMF*). *UMF* allows creating test ready models with uncertainty at three logical testing levels of CPS: *Application*, *Infrastructure*, and *Integration*. The core of *UMF* is UML Uncertainty Profile (*UUP*), which implements an existing uncertainty conceptual model for CPSs, called *U-Model*. In addition, *UMF* 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*. *UMF* also relies on the UML Testing Profile (UTP) V.2 to construct test ready. Finally, *UMF* defines concrete guidelines for supporting the use of *UMF* for creating test ready models with uncertainty. We evaluated *UMF* with two industrial and one open source case studies. As a future work, we are implementing test generators that can take test ready models created with *UMF* as input and generate executable test cases.

ACKNOWLEDGMENT

This research was supported by the EU Horizon 2020 funded project U-Test (Testing Cyber-Physical Systems under Uncertainty). Tao Yue and Shaukat Ali are also supported by RCN funded Zen-Configurator project, RFF Hovedstaden funded MBE-CR project and RCN funded MBT4CPS project.

REFERRENCE

[1] Ali, S., Briand, L.C., and Hemmati, H., 2012. Modeling robustness behavior using aspect-oriented modeling to support robustness testing of industrial systems. *Software & Systems Modeling 11*, 4, 633-670.

[2] Ali, S. and Yue, T., 2015. U-Test: Evolving, Modelling and Testing Realistic Uncertain Behaviours of Cyber-Physical Systems IEEE, 1-2.

[3] Atanassov, K. and Georgiev, C., 1993. Intuitionistic fuzzy prolog. *Fuzzy Sets and Systems 53*, 2, 121-128.

[4] De Luca, A. and Termini, S., 1972. A definition of a nonprobabilistic entropy in the setting of fuzzy sets theory. *Information and control 20*, 4, 301-312.

[5] Dempster, A.P., 1967. Upper and lower probabilities induced by a multivalued mapping. *The annals of mathematical statistics*, 325-339.

[6] Didier, D. and Henri, P., 1980. Fuzzy sets and systems: Theory and Applcation. *Mathematics in Scince and Engineering 144*.

[7] Famelis, M., Salay, R., and Chechik, M., 2012. Partial models: Towards modeling and reasoning with uncertainty. In *Software Engineering (ICSE), 2012 34th International Conference on* IEEE, 573-583.

[8] Famelis, M. and Santosa, S., 2013. MAV-Vis: a notation for model uncertainty. In *Modeling in Software Engineering (MiSE), 2013 5th International Workshop on* IEEE, 7-12.

[9] Garousi, V., 2008. Traffic-aware stress testing of distributed real-time systems based on UML models in the presence of time uncertainty. In *Software Testing, Verification, and Validation, 2008 1st International Conference on* IEEE, 92-101.

[10] Gau, W.L. and Buehrer, D.J., 1993. Vague sets. *Systems, Man and Cybernetics, IEEE Transactions on 23*, 2, 610-614. DOI= http://dx.doi.org/10.1109/21.229476.

[11] George J, K. and Bo, Y., 2008. Fuzzy sets and fuzzy logic, theory and applications. -.

[12] Goguen, J.A., 1967. L-fuzzy sets. *Journal of mathematical analysis and applications 18*, 1, 145-174.

[13] Grassi, V. and Mirandola, R., 2001. UML modelling and performance analysis of mobile software architectures. In *≪ UML≫ 2001—The Unified Modeling Language. Modeling Languages, Concepts, and Tools* Springer, 209-224.

[14] Grattan‐Guinness, I., 1976. Fuzzy Membership Mapped onto Intervals and Many‐Valued Quantities. *Mathematical Logic Quarterly 22*, 1, 149-160.

[15] Group, O.M., 2005. UML Profile For Schedulability, Performance, And Time.

[16] Group, O.M., 2013. Concrete Syntax For A UML Action Language: Action Language For Foundational UML (ALF).

[17] Group, O.M., April, 2013. UML Testing Profile.

[18] Group, O.M., June, 2011. UML Profile For MARTE: Modeling And Analysis Of Real-Time Embeded Systems.

[19] Haroonabadi, A., Teshnehlab, M., and Movaghar, A., 2008. A novel method for behavior modeling in uncertain information systems. *World Academy of Science, Engineering and Technology 41*, 959-966.

[20] Hartley, R.V.L., 1928. Transmission of information. *Bell System Technical Journal*, 535-563.

[21] Higashi, M. and Klir, G.J., 1982. Measures of uncertainty and information based on possibility distributions. *International Journal of General Systems 9*, 1, 43-58.

[22] Höhle, U., 1981. Fuzzy plausibility measures. In *Proceedings of the 3th International Seminar on Fuzzy Set Theory, Johannes Kepler University, Linz*, 7-30.

[23] Höhle, U., 1982. Entropy with respect to plausibility measures. In *Proceedings of the 12th IEEE International Symposium on Multiple-Valued Logic*, 167-169.

[24] Ibm. *IBM Rational Software Architect Modeling Tool*2016. https://http://www.ibm.com/developerworks/downloads/r/architect/.

[25] Ibm. *IBM RSA Simulation Toolkit*2016. http://www-03.ibm.com/software/products/en/ratisoftarchsimutool.

[26] Jahn, K.U., 1975. Intervall - wertige Mengen. *Mathematische Nachrichten 68*, 1, 115-132.

[27] Kerwin, A., 1993. None Too Solid Medical Ignorance. *Science Communication 15*, 2, 166-185.

[28] Kosko, B., 1986. Fuzzy entropy and conditioning. *Information sciences 40*, 2, 165-174.

[29] Lagarde, F., Espinoza, H., Terrier, F., André, C., and Gérard, S., 2008. Leveraging patterns on domain models to improve UML profile definition. In *Fundamental Approaches to Software Engineering* Springer, 116-130.

[30] Lamata, M.T. and Moral, S., 1988. Measures of entropy in the theory of evidence. *International Journal of General System 14*, 4, 297-305.

[31] Ma, Z., 2005. Fuzzy information modeling with the UML. *Idea*.

[32] Ma, Z.M., Zhang, F., and Yan, L., 2011. Fuzzy information modeling in UML class diagram and relational database models. *Applied Soft Computing 11*, 6, 4236-4245.

[33] Ma, Z.M., Zhang, F., Yan, L., and Cheng, J., 2011. Representing and reasoning on fuzzy UML models: A description logic approach. *Expert Systems with Applications 38*, 3, 2536-2549.

[34] Motameni, H., Daneshfar, I., Bakhshi, J., and Nematzadeh, H., 2010. Transforming fuzzy state diagram to fuzzy Petri net. *Journal of Advances in Computer Research 1*, 1, 29-44.

[35] Negnevitsky, M., 2005. *Artificial intelligence: a guide to intelligent systems*. Pearson Education.

[36] Pawlak, Z., 1982. Rough sets. *International Journal of Computer & Information Sciences 11*, 5, 341-356.

[37] Pressman, R.S., 2010. *Software engineering: a practitioner's approach 7th edition*. Palgrave Macmillan.

[38] Riebisch, M., Philippow, I., and Götze, M., 2002. UML-based statistical test case generation. In *Objects, Components, Architectures, Services, and Applications for a Networked World* Springer, 394-411.

[39] Salay, R., Famelis, M., and Chechik, M., 2012. Language independent refinement using partial modeling. In *Fundamental Approaches to Software Engineering* Springer, 224-239.

[40] Shafer, G., 1976. *A mathematical theory of evidence*. Princeton university press Princeton.

[41] Sicilia, M.-A. and Mastorakis, N., 2004. Extending UML 1. 5 for fuzzy conceptual modeling: An strictlyadditive approach. *WSEAS Transactions on Systems 3*, 5, 2234-2239.

[42] Smets, P. and Kennes, R., 1994. The transferable belief model. *Artificial intelligence 66*, 2, 191-234.

[43] Yager, R.R., 1983. Entropy and specificity in a mathematical theory of evidence. *International Journal of General System 9*, 4, 249-260.

[44] Yan, L. and Ma, Z.M., 2009. Extending nested relational model for fuzzy information modeling. In *2009 WASE International Conference on Information Engineering* IEEE, 587-590.

[45] Zadeh, L.A., 1965. Fuzzy sets. *Information and control 8*, 3, 338-353.

[46] Zadeh, L.A., 1975. The concept of a linguistic variable and its application to approximate reasoning—I. *Information sciences 8*, 3, 199-249.

[47] Zadeh, L.A., 1978. Fuzzy sets as a basis for a theory of possibility. *Fuzzy Sets and Systems 1*, 1, 3-28.

[48] Zhang, F. and Ma, Z.M., 2013. Construction of fuzzy ontologies from fuzzy UML models. *International Journal of Computational Intelligence Systems 6*, 3, 442-472.

[49] Zhang, M., Ali, S., Yue, T., and H. Nguyen, P., 2016. *Uncertainty Modeling Framework for the Integration Level V.1.* https://www.simula.no/publications/uncertainty-modeling-framework-integration-level-v1

[50] Zhang, M., Selic, B., Ali, S., Yue, T., Okariz, O., and Norgren, R., 2016. Understanding Uncertainty in Cyber-Physical Systems: A Conceptual Model. In *Proceedings of the ECMFA* (2016), Simula Laboratory Research. https://www.simula.no/publications/understanding-uncertainty-cyber-physical-systems-conceptual-model-0

[51] Zimmermann, H.-J., 2011. *Fuzzy set theory—and its applications*. Springer Science & Business Media.