Uncertainty-wise Cyber-Physical Systems Testing

Man Zhang

Thesis submitted for the degree of Ph.D.

Department of Informatics
Faculty of Mathematics and Natural Sciences
University of Oslo
2018

Abstract

A Cyber-Physical Systems (CPS), as an integration of computing, communication, and control for making intelligent and autonomous systems, has been widely applied in various safety-critical domains, e.g., avionics and automotive. However, uncertainty is inherent in CPSs due to various reasons such as unpredictable environment under which the CPSs are operated. And, uncertainties may cause irreparable accidents once they cannot be handled properly by CPSs. Therefore, it is crucial to identify uncertainties in CPSs and test CPSs under the uncertainties, to ensure that CPSs are capable of handling the uncertainties during their actual operations, i.e., making CPSs *less* uncertain.

Towards this direction, five contributions were made in the thesis corresponding to five papers respectively: (**C1**) a conceptual model, named as *U-Model*, for helping develop a systematic and comprehensive understanding of uncertainty in CPSs; (**C2**) an use case modeling methodology, named as *U-RUCM*, for identifying, qualifying, and, where possible, quantifying uncertainty in requirements engineering; (**C3**) a test modeling methodology, named as *UncerTum*, for supporting the construction of test ready models with the explicit representation of uncertainties in CPSs; (**C4**) an evolution framework, named as *UncerTolve*, for interactively evolving test ready models specified with *UncerTum* based on real operational data; and (**C5**) a testing framework, named as *UncerTest*, for testing CPSs in the presence of uncertainties in their operating environments in a cost-effective manner using model-based and search-based testing techniques.

Based on our evaluations of the five contributions with the industrial CPS case studies, we observed that *U-Model*, as the foundation for this research, is sufficiently complete for characterizing and classifying uncertainties in CPSs. Then, the *U-Model* based modeling methodologies *U-RUCM* and *UncerTum* offer solutions to enable the identification and specification of uncertainties at two critical phases of a system development lifecycle: requirements engineering and testing. Furthermore, *UncerTolve* can successfully evolve model elements of the test ready models specified with *UncerTum* and calculate objective uncertainty measurements based on real operational data. Last, *UncerTest* managed to cost-effectively test CPSs in the presence of uncertainties and proactively identify unknown uncertainties by introducing the sources of the uncertainties into the test environments during test case execution.

Acknowledgments

When writing the last part of the thesis, a feeling of gratitude welled up in my heart for the people who helped me to make the thesis possible.

First and foremost, I would like to express my deepest gratitude to my supervisors: Shaukat Ali and Tao Yue. They have both consciously and unconsciously taught me how to think, how to express and how to do research. I appreciated their immense knowledge, professional guidance, continues supports, patience and time to make my *uncertain* Ph.D. pursuit be towards *certain*. In addition, the enthusiasm and enjoy they presented for their research motived me when I met the bottle-neck in my research.

I would like to thank all the members of our group for their caring and help in my research and daily life. Without them, I cannot have this great time after I moved away from my country for the first time. Also, I would like to thank the members of the U-Test project, who provided me invaluable suggestions and case studies for conducting the evaluation of the solutions proposed in this thesis. Moreover, I would like to thank all colleagues at Simula Research Laboratory, who altogether create a pleasant research environment.

Last, I would like to express my special gratitude to my parents for their encouragement and support. Whenever I need help, you are always there. Thank you.

List of papers

The following papers are included in this thesis:

Paper A. Understanding Uncertainty in Cyber-Physical Systems: A Conceptual Model

M. Zhang, B. Selic, S. Ali, T. Yue, O. Okariz and R. Norgren.

In: Proceedings of the 12th European Conference on Modelling Foundations and Applications (ECMFA 2016), pp. 247-264, 2016. DOI: 10.1007/978-3-319-42061-5_16

Paper B. Specifying Uncertainty in Use Case Models

M. Zhang, T. Yue, S. Ali, B. Selic, O. Okariz, R. Norgren and K. Intxausti.

Journal paper that has been submitted to the Journal of Systems and Software (JSS), second revision.

Paper C. Uncertainty-Wise Cyber-Physical System Test Modeling

M. Zhang, S. Ali, T. Yue, R. Norgren and O. Okariz

Journal of Software & Systems Modeling (SOSYM). DOI: 10.1007/s10270-017-0609-6

Paper D. Uncertainty-Wise Evolution of Test Ready Models

M. Zhang, S. Ali, T. Yue and R. Norgren.

Journal of Information and Software Technology (IST). DOI: 10.1016/j.infsof.2017.03.003

Paper E. Uncertainty-wise Test Case Generation and Minimization for Cyber-Physical Systems: A Multi-Objective Search-based Approach

M. Zhang, S. Ali and T. Yue.

Journal paper that has been submitted to ACM Transactions on Software Engineering and Methodology (TOSEM).

Contents

Abstract ..... **i**

Acknowledgments ..... **ii**

List of papers ..... **iii**

Contents ..... **iv**

Part I Summary **1**

1 Introduction 2

2 Background 7
2.1 Cyber-Physical System and its uncertainty 7
2.2 Restricted Use Case Modeling (RUCM) 7
2.3 Model-based Testing (MBT) 8
2.4 Search-based Software Testing 9
2.5 Uncertainty Theory 10
2.5.1 Probability Theory vs. Uncertainty Theory. 10
2.5.2 Uncertainty Measure and Uncertainty Space 10

3 Research Methodology 12
3.1 Research activities 12
3.2 Implementations 13

4 Uncertainty-wise CPSs Testing Methodologies 15
4.1 U-Model 16
4.2 U-RUCM 17
4.3 UncerTum 20
4.4 UncerTolve 21
4.5 UncerTest 23

5 Evaluation 25
5.1 Case Study 25
5.1.1 GeoSports 25
5.1.2 Automated Warehouse 26
5.2 U-Model (Paper A) 27
5.3 U-RUCM (Paper B) 29
5.4 UncerTum (Paper C) 30
5.5 UncerTolve (Paper D) 32
5.6 UncerTest (Paper E) 33

6 Discussion 35

7 Conclusion and Future Work 38

Reference 39

Part II **46**

Paper A **47**

Abstract 48

1 Introduction 48

2 Background and Running Example 49
3 Uncertainty Conceptual Model 51
3.1 Belief Model 51
3.1.1 Belief, BeliefAgent and BeliefStatement 53
3.1.2 Evidence, EvidenceKnowledge, IndeterminacySource and IndeterminacyKnowledge. 55
3.1.3 Measurement and Measure. 57
3.2 Uncertainty Model 57
3.2.1 Uncertainty, Lifetime and Pattern. 58
3.2.2 Locality and Risk. 60
3.3 Measure Model 61
4 Evaluation 62
4.1 Development and Validation of Uncertainty Requirements and U-Model 62
4.2 Evaluation Results 64
5 Related Work 65
6 Conclusion 67
References 68

Paper B 72

Abstract 73
1 Introduction 74
2 Background and Running Example 76
2.1 U-Model 76
2.2 Running Example 77
2.3 Restricted Use Case Modeling (RUCM) 77
3 U-RUCM Templates and Keywords 80
4 U-RUCM Formalization 83
4.1 Relationships of BeliefUCMeta with UCMeta and U-Model 83
4.2 Belief Use Case Model, Element, and Classifier 84
4.3 Belief Use Case Specification 85
4.4 Belief Flow of Events 89
4.5 Belief Sentence 92
4.6 Uncertainty 94
4.6.1 Uncertainty in Belief Sentences (NLUncertainty) 95
4.7 Branch Uncertainty 96
4.8 Measurement 97
5 Tool Support and Methodology 99
5.1 Tool Support 99
5.2 Methodology 99
6 Evaluation 104
6.1 Case Studies 104
6.2 Context, Design, and Execution of Evaluation 105
6.3 Results 107
6.4 Experience, Lessons Learned, and Future Challenges 111
7 Related Work 113
8 Conclusion and Future Work 115

Acknowledgment 116
References 116

Paper C 122
Abstract 123
1 Introduction 123
2 Background 127
2.1 Cyber-Physical Systems and Testing Levels 127
2.2 U-Model 128
2.3 UML Testing Profile (UTP) 128
3 Running Example 129
4 Overview of UncerTum 134
5 UUP and CPS Testing Levels Profile 136
5.1 UUP Belief 137
5.2 UUP Uncertainty and Measurement 139
5.3 CPS Testing Levels Profile 141
6 Model Libraries 141
6.1 Measure Libraries 142
6.2 Pattern Library 145
7 UncerTum Modeling Methodology 145
7.1 Overview 147
7.2 Application Level Modeling 148
7.3 Infrastructure Level Modeling 149
7.4 Integration Level Modeling 150
7.5 Apply UUP (AP2/IF2/IT2) 151
7.5.1 Measurement Modeling 153
7.5.2 Uncertainty Modeling 155
8 UncerTum Validation Process 158
8.1 UAL Executable Modeling Guidelines 159
8.2 Recommendations to Fix Problems in Test Ready Models 160
9 Evaluation 161
9.1 Development and Validation of UncerTum and Test Ready Models 162
9.2 Evaluation Results 165
9.2.1 Mapping UUP/Model Libraries to U-Model and MARTE 166
9.2.2 Application of UUP/Model Libraries 169
9.2.3 Validation of Test Ready Models via Model Execution 170
9.2.4 Application of UTP V.2 171
9.3 Overall Discussion and Limitations 172
10 Related Work 173
11 Conclusion and Future Work 175
Acknowledgment 176
References 177

Paper D 183

Abstract 184

1 Introduction 185
1.1 Challenges and Objectives 186
1.2 Context, Scope and Overview 187
1.3 Contributions 190
1.4 Results and the Structure of the Paper 191

2 Related Work 192
2.1 Comparison with Existing Works 192
2.2 Comparison with Our Previous Works 195

3 Background 196
3.1 Cyber-Physical Systems and Uncertainty Levels 196
3.2 UML Testing Profile 197
3.3 U-Model 198
3.4 UncerTum 198
3.5 UncerTest 199

4 Terminologies And Running Example 200
4.1 Belief Test Ready Model 200
4.2 Executable Belief Test Ready Model 203
4.3 Driver Model 203

5 Architecture and Current Implementation of *UncerTolve* 205
5.1 Architecture 205
5.2 Current Implementation of UncerTolve 208

6 Recommended Methodology 211
6.1 Creating BM and Driver Model (S1) 212
6.2 Validate BM and Driver Model, and Evolve Objective Uncertainty Measurements (S2) 214
6.2.1 Analysis of Errors and Fixing Models (S2A2, S2A3, and S2M1) 215
6.2.2 Identifying Objective Uncertainty Measurements (S2A4) 216
6.3 Evolve Belief State Machines with Dynamic Invariant Analysis (S3) 217

7 Evaluation 219
7.1 Results of Creating BM and DM (S1) 220
7.2 Results of Validation and Evolution via Model Execution (S2) 220
7.3 Results of Dynamic Inference (S3) 221
7.4 Overall Validation 222
7.5 Effort to Build Belief Test Ready Models and Adoption of UncerTolve 223
7.6 Discussion and Experiences 224
7.7 Threats to Validity 225

8 Conclusion 226

Acknowledgment 226

References 227

Paper E 232

Abstract 233

1 Introduction 233

2 Background 235
2.1 Uncertainty Modeling Framework (UncerTum) 236

2.2 Uncertainty Theory 236
2.2.1 Probability Theory vs. Uncertainty Theory 236
2.2.2 Summary of Uncertainty Theory 237
2.3 Example of the Application of UncerTum and Uncertainty Theory 237
3 Overview 239
4 Test Case Generation and Minimization 240
4.1 Abstract Test Case Generation 240
4.1.1 Definitions 240
4.1.2 Strategies 243
4.2 Uncertainty-Wise Test Case Minimization 245
4.2.1 Problem Representation 245
4.2.2 Definitions and Functions of the Six Minimization Objectives 246
4.2.3 Uncertainty-wise Test Case Minimization Problems 247
4.3 Executable Test Case Generation 248
4.3.1 Enabling Indeterminacy 248
4.3.2 Test Setup and Test Data Generation 251
4.4 Test Execution and Reporting 252
5 Evaluation 253
5.1 Case Study 254
5.2 Research Questions 256
5.3 Design of the Evaluation 257
5.4 Results and Analyses 259
5.4.1 Results for RQ1 259
5.4.2 Results for RQ2 260
5.4.3 Results for RQ3 261
5.5 Discussion 263
5.6 Threats to Validity 264
6 Automation 265
7 Related Work 267
8 Conclusion 269
Acknowledgment 270
References 270

Appendixes 277

Appendix A. Definitions of U-Model Concepts 277
A.1 Belief Model 277
A.2 Uncertainty Model 280
A.3 Measure Model 283
Appendix B. An Example of Questionnaire of the AW Case Study 284
Appendix C. An Example of BUCS Specified with the U-RUCM Editor 289
Appendix D. tolveR-E 290
Appendix E. tolveR-D 291

Part I
Summary

Summary

1 Introduction

A Cyber-Physical Systems (CPS), as an integration of computing, communication, and control for making intelligent and autonomous systems [1], has been widely applied in various safety-critical domains, e.g., avionics and automotive [1-3]. However, uncertainty is inherent in CPSs, due to various reasons such as unpredictable operating environments of CPSs [1-3]. And, uncertainties may cause irreparable accidents once they cannot be handled properly by CPSs. Therefore, it is crucial to identify the uncertainties and test CPSs under the uncertainties, to ensure that CPSs are capable of handling the uncertainties during their actual operations, i.e., making CPSs *less* uncertain.

In the thesis, we investigated uncertainty in CPSs from the subjective view. This means that *uncertainty* is considered as a state of affairs whereby, for whatever reason, participants who are involved in the phase of CPSs lifecycle (e.g., designer) lack perfect knowledge about some interests of CPSs (e.g., a state of CPSs during the actual operations). Thus, the identification of the uncertainties in CPSs originates from the participants according to their knowledge at a given point of time. More specifically, the identification is about sufficiently and explicitly capturing uncertainties known by the participants. In addition, the identification is also about to exploring unknown uncertainties, which might be known at some point in the future. However, the identified uncertainties in CPSs may be not valid. Therefore, it is also required to take an objective approach (e.g., testing) to validate the identified uncertainties based on CPSs. Towards this direction, a series of systematic,

uncertainty-wise, model-based methodologies (Figure 1) was proposed with respect to five contributions (C1-- C5).

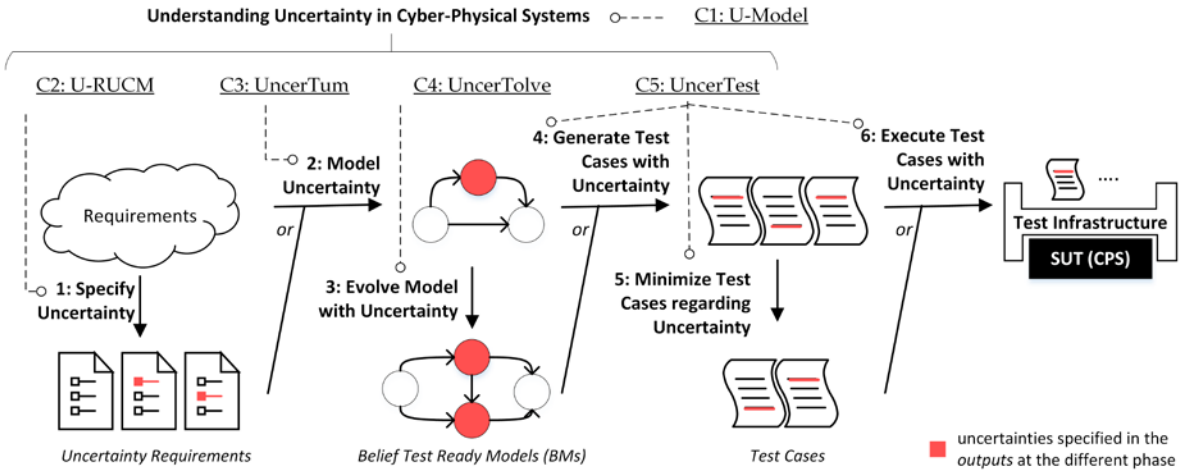


Figure 1. The scope of the *Uncertainty-wise Cyber-Physical Systems Testing*

As shown in Figure 1, the first contribution (C1) is *U-Model*, which is a conceptual model for understanding uncertainty in CPSs. The conceptual model takes a subjective approach to represent uncertainties, i.e., lack of knowledge about some interests of CPSs held by some agents or agencies (i.e., *belief agents*). In addition, the classification and abstractions of uncertainties in CPSs were defined in *U-Model*. Furthermore, a set of the commonly known measures of uncertainty were also introduced in *U-Model*. Note that all proposed uncertainty-wise methodologies share the same definition of uncertainty at the conceptual level based on *U-Model*.

To handle uncertainty at the early stage of the development of CPSs, we developed *U-RUCM* (C2 in Figure 1) for specifying uncertainty as a part of requirements of CPSs, i.e., *uncertainty requirements*. In terms of testing CPSs under uncertainty using MBT as shown in Figure 1, *U-RUCM* also aims to provide the precise uncertainty requirements for supporting further modeling activities (e.g., constructing test ready models with uncertainties). In *U-RUCM*, two templates were introduced for structuring and specifying the uncertainty requirements. Such requirements can be automatically formalized as instances of a formal *U-RUCM* metamodel.

To enable MBT of CPSs under uncertainty, an uncertainty modeling methodology, named as *UncerTum*, was developed (C3 in Figure 1), for constructing belief test ready models (BMs). BMs annotate the test ready models with uncertainties, describing the uncertain

behaviors of CPSs and/or their uncertain operating environments. Moreover, the methodology also allowed to specify the measurements and characteristics of the uncertainties as parts of BMs. The core of *UncerTum* is the UML Uncertainty Profile (*UUP*), which was implemented based on *U-Model*. In addition, we also developed three model libraries (i.e., time library, pattern library and measure library) in *UncerTum* for constructing BMs with the advanced modeling features.

To evolve and validate BMs, we developed an uncertainty-wise evolving framework (C4 in Figure 1), called *UncerTolve*. As shown in Figure 1, since BMs are produced by modelers, BMs may be not complete and correct due to their mistakes made accidentally. Therefore, it is vital to validate the BMs based on some objective evidence. In addition, the BMs are the inputs of MBT of CPSs, which decides whether CPSs can be tested sufficiently with uncertainties. Thus, the sources of uncertainty in the BMs are required to be identified sufficiently and correctly before testing. Otherwise, it might be time-consuming to observe the occurrence of uncertainty during testing, due to the randomness of the appearance of the sources. As such, *UncerTolve* was proposed to be interactive to evolve BMs based on available evidence, i.e., real operational data collected from real applications of CPSs. To achieve the evolution, three main features were implemented: 1) validating BMs based on the data via model execution; 2) deriving objective uncertainty measurements (i.e., frequency) based on the result of the model execution; 3) evolving state invariants and guards of transitions of BMs together with the data using a dynamic invariant detector.

Last but not least, we developed a search-based and model-based framework to test CPSs in the presence of uncertainty, called *UncerTest* (C5 in Figure 1). To be uncertainty oriented and cost-effective, we implemented three features in *UncerTest* for supporting three testing activities shown as 4-6 steps in Figure 1: (1) *Test Case Generation*: Two test case generation strategies were proposed in *UncerTest* corresponding to two coverage criterions, All Simple Belief Path Coverage (*ASiBP*) and All Specified Length Belief Path Coverage (*ASlBP*); (2) *Test Case Minimization*: To reduce the number of the generated test cases, we defined four test cases minimization strategies based on multi-objective search; and (3) *Test Case Execution*: A set of an uncertainty-wise test verdicts were defined for evaluating the occurrence of uncertainty during test case execution.

Through the entire process (1-6 steps) as shown in Figure 1, uncertainties in CPSs can be gained step by step. More concretely, the modeling methodologies (i.e., *U-RUCM* and *UncerTum*) is potential to help modelers to identify more known uncertainties by characterizing and structuring the uncertainties explicitly. In addition, the evolution provided by *UncerTolve* helps to validate the known uncertainties and identify new uncertainties which are not obvious to modelers but exist according to available evidence, i.e., real operational data. Last, *UncerTest* enables to test CPSs under uncertainty by introducing various sources of uncertainty into test environments, and thus we are also able to observe new (i.e., previously unknown) uncertainties during testing.

The structure of this thesis is presented in Figure 2, which includes two parts.

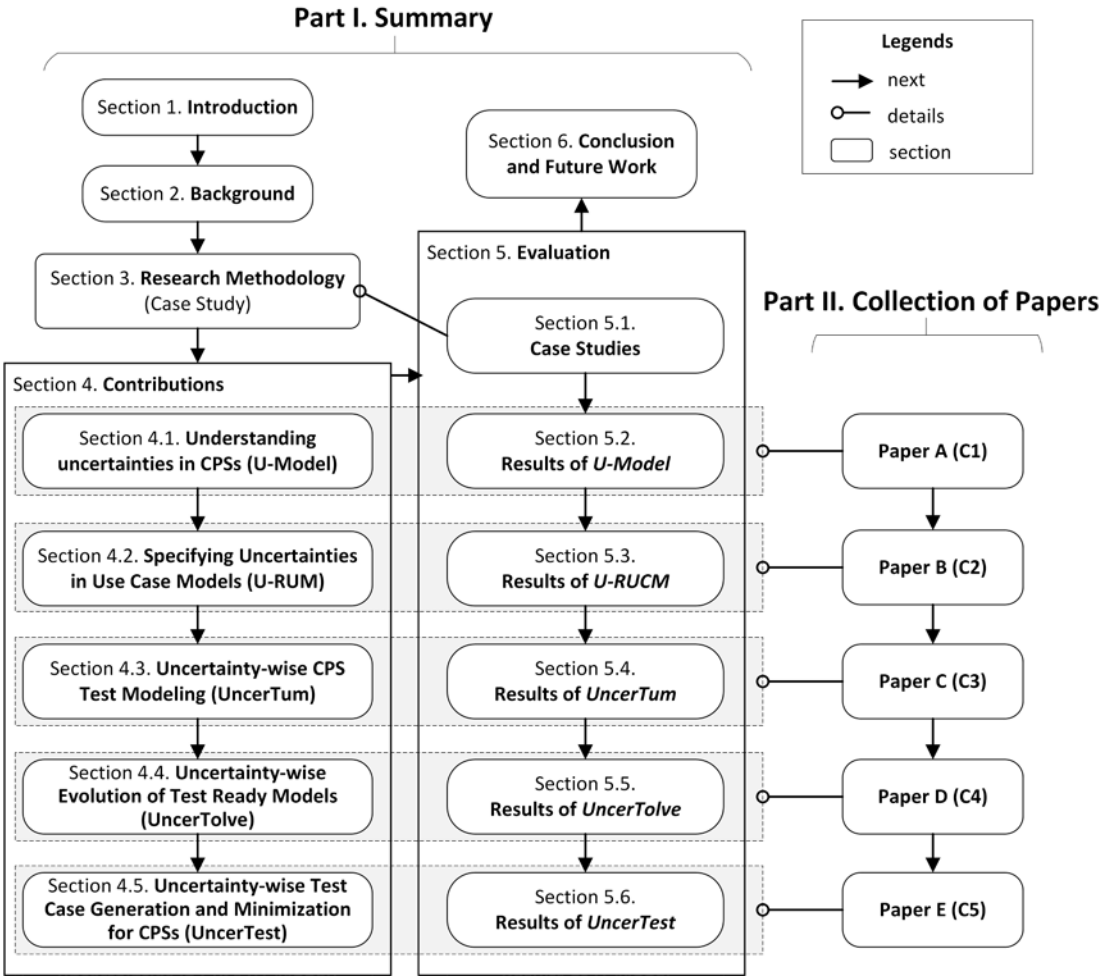


Figure 2. The structure of the thesis

The first part (Part I) is about summarizing the research work for the entire thesis, which is organized as the following: Section 2 is the background that provides the necessary information for understanding the thesis; The research methodology is presented in Section 3; Section 4 briefly presents the contributions of the thesis, followed by the summary of the key results (Section 5); and Section 6 concludes the thesis. The second part (Part II) is about collecting the related papers with respect to the five contributions (*C1*-- *C5*). The mapping between the summary and the collection of papers is shown in Figure 2.

2 Background

2.1 Cyber-Physical System and its uncertainty

As defined in [4], a CPS is composed of *"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."*

According to the definition, uncertainty in CPSs can be categorized as the three logical levels [4] as shown in Figure 3. First, the uncertainty at the application level originates from an application (one or more software components) of a physical unit of CPS, which can be one software component, or the interaction between human being and applications, or the interaction among software components within one physical unit. Second, the uncertainty at the infrastructure level originates from the hardware of a physical unit, or networking infrastructure and/or cloud infrastructure built on the set of physical units, or data transmission via information network enabled through the infrastructure. Last, the uncertainty at the integration level originates from either the interaction of applications across physical units at the application level or the interactions of physical units across the application and infrastructure levels.

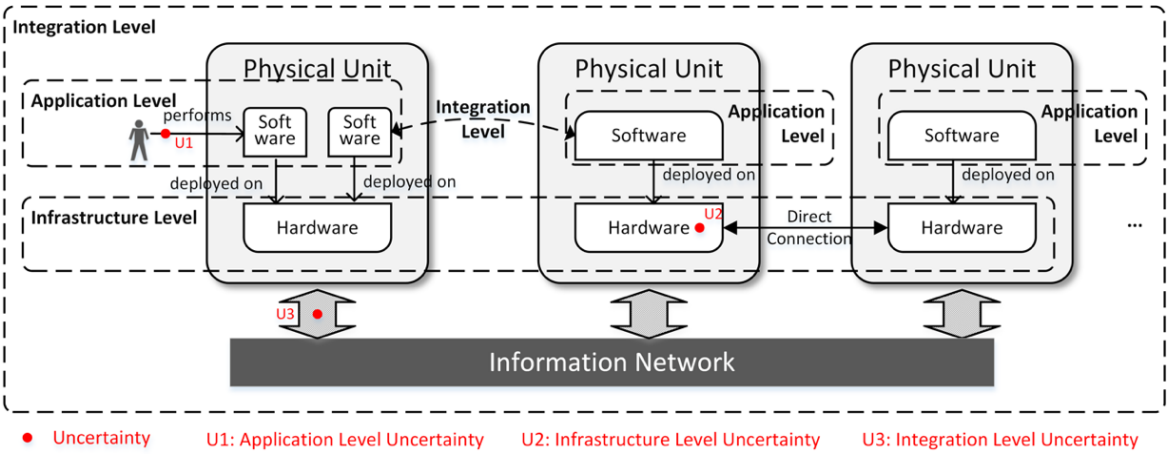


Figure 3. Uncertainty in Cyber-Physical Systems (CPSs)

2.2 Restricted Use Case Modeling (RUCM)

RUCM is a methodology for specifying textual use case specifications [5, 6], aiming at being easy to use, reducing the ambiguity of the textual specification, and supporting further

automated analyses. Two controlled experiments were conducted to evaluate RUCM, and the results showed that it is overall easy to use and achieved a significant improvement of the understandability of use case specifications [5, 6].

RUCM is composed of a use case template, a set of keywords, and a formalization mechanism. A use case specification structured with the RUCM use case template has a mandatory basic flow and optional alternative flow(s). An alternative flow always depends on a condition occurring in a specific step of another flow (reference flow). Accordingly, alternative flows are classified into three types: a *specific alternative flow* refers to one specific step of a reference flow; a *bounded alternative flow* refers to more than one specific steps of reference flow(s); a *global alternative flow* refers to any step in any other flow. By such template, the interactions among flows of events in the system are able to be precisely defined. In RUCM, a set of keywords were proposed for specifying control structures, such as IF-THEN-ELSE-ELSEIF-ENDIF for conditional logic, DO-UNTIL for iteration, and VALIDATE THAT for the condition check. The introduction of these keywords helps to reduce ambiguities in the use case specifications and facilitates the automated analysis model transformation. In addition, a metamodel, UCMeta, was developed for formalizing RUCM using the OMG's standard Meta-Object Facility (MOF) [7]. The metamodel not only covers all concepts in the RUCM use case template and keywords, but also captures the natural-language (NL) concepts in a sentence, e.g., subject, object, and verb. Moreover, an automated solution was provided in the RUCM framework, called aToucan [5], to automatically extract the NL information and generate three types of UML analysis models: class, sequence and activity diagrams.

Several extension works [8-11] have been developed since RUCM was initially introduced by Yue et al. [12] in 2009. In the thesis, an extension of RUCM (Section 4.2) for explicitly specifying uncertainty in use case specifications was proposed.

2.3 Model-based Testing (MBT)

Model-based Testing (MBT) is a technique for performing software system testing using models [13]. In MBT, models can be used to express the expected behavior of the system under test, and/or its environment to be tested.

MBT has been in use since the 1980s [14]. With the rapid expansion of the interest in MBT from the industry and academia, the feasibility and cost-effectiveness of MBT have been demonstrated by the intensive research work and industrial practices [14, 15].

The typical process of MBT includes five steps [15]: (1) construct models with respect to the system under tested and/or its environment; (2) generate a set of abstract test cases based on the constructed models according to the defined test selection criteria; (3) concretize the abstract test cases to executable ones; (4) execute the test cases on test infrastructures and assign verdicts; and (5) analyze execution results of the test cases. The constructed models can be reused for generating abstract test cases according to the different criteria. In addition, the same set of abstract test cases can be made executable in different environments by changing the adaptation layer for converting abstract test cases into executable ones.

In the thesis, we developed a modeling framework (Section 4.3) for constructing test ready models with uncertainty. Such models can be used to test CPSs under uncertainty by the proposed testing framework (Section 4.5).

2.4 Search-based Software Testing

Search-based Software Testing (SBST) is about applying a meta-heuristic optimizing search technique to tackle software testing problems, such as automatic generation of test data [16-20]. In SBST, these testing problems are normally reformulated as search problems for seeking optimal or near-optimal solutions in a search space. The process is guided by fitness functions that are defined to evaluate the sought solution for seeking the better ones. The applicability and effectiveness of SBST can be demonstrated by many successful works and related surveys [21-25].

Recently, multi-objective approaches are increasingly applied for optimizing the test process, such as test set selection, minimization, and prioritization [20, 25]. One possible reason is that the problem in software testing faces multiple objectives in nature [20], and it may be conflict among the objectives, such as time budget and the coverage of selected test cases. In SBST, Pareto optimality is one commonly applied approach to deal with such situation by outputting a set of trade-off optimal solutions [18, 20, 26, 27]. A multi-objective optimization problem can be formulated as a set of fitness functions corresponding to the objectives being achieved. In Pareto optimality, a solution *S1* is said to *dominate* (implies

better) other solution *S2* if *S1* is strictly better than *S2* in at least one fitness function and *S1* is no worse than *S2* in all other fitness functions. Then a solution is called Pareto optimal if no existing another solution can *dominate* it. To solve a multi-objective problem using search algorithms based on Pareto optimality, a set of Pareto optimal solutions are produced.

In the thesis, a set of multi-objective problems for minimizing the automatically generated test cases were defined with considerations of the cost-effectiveness and uncertainty during testing CPSs (Section 4.5). In addition, eight Pareto-based search algorithms were selected for assessing the performances of the algorithms in solving the defined test case minimization problems.

2.5 Uncertainty Theory

2.5.1 Probability Theory vs. Uncertainty Theory.

To measure uncertainty, *Probability Theory* is commonly applied method (e.g., [28, 29]) in the practice to treat the measurement of uncertainty as a *frequency*. But, the application of this theory is built on an amount of data collected from the long-run experiment (i.e., being "close enough to the long-run frequency" [30]). However, in the context of software testing, it is quite common that data is not able to be obtained at the startup phase of the test design for enabling MBT, due to, e.g., budget issues [30]. Therefore, it is not ideal to apply *Probability Theory* to obtain the *frequency* in such context with its usage of guiding the test phases, e.g., test design and test optimization.

Uncertainty Theory defined by Liu [31] is *"a branch of mathematics for modeling human uncertainty"*. *Uncertainty Theory* is to measure uncertainty as a *belief degree* from the subjective perspective, heavily depending on the knowledge and experience of the domain expert. It is a natural fit for our context, i.e., handling uncertainty even lacking observed data and treating uncertainty from the subjective perspective (*U-Model* [4]). It is also important to note that *Uncertainty Theory* has been applied to solve various problems from different domains, e.g., [32-35]. In the thesis, an application of *Uncertainty Theory* for obtaining the uncertainty measurement of a test case is presented in Section 4.5, and the related definitions are described in Section 2.5.2.

2.5.2 Uncertainty Measure and Uncertainty Space

Uncertainty Measure (UM) is defined in *Uncertainty Theory* that is a specific value (i.e., a number) assigned to the *belief degree* of an event [31] by a *belief agent*, indicating his/her/its confidence about the occurrence of the event [4]. In *Uncertainty Theory*, *UM* is represented as the $\mathcal{M}$ symbol. As Liu suggested in [31], $\mathcal{M}$ respects the following three axioms:

Axiom 1. (Normality) $\mathcal{M}(\Gamma) = 1$, (Γ is the universal set).

Axiom 2. (Duality) $\mathcal{M}\{\Lambda\} + \mathcal{M}\{\Lambda^c\} = 1$, where Λ shows a particular event, whereas Λ^c shows all the elements in the universal set excluding Λ.

Axiom 3. (Subadditivity) $\mathcal{M}\{\bigcup_{i=1}^{\infty} \Lambda_i\} < \sum_{i=1}^{\infty} \mathcal{M}\{\Lambda_i\}$ (every countable sequence of events Λ_1, Λ_2, ...).

In addition, *Uncertainty Space* and its related theorem which are relevant to the thesis are presented as below. For more details of *Uncertainty Theory*, readers may consult [31].

Uncertainty Space: A triplet $(\Gamma, \mathcal{L}, \mathcal{M})$, where Γ is the universal set, $\mathcal{L}$ is a σ-algebra [36] over Γ, and $\mathcal{M}$ is UM.

Theorem: Let $(\Gamma_k, \mathcal{L}_k, \mathcal{M}_k)$ be uncertainty spaces and $\Lambda_k \in \mathcal{L}_k$, for $k = 1, 2, \ldots n$. Then $\Lambda_1, \Lambda_2, \ldots \Lambda_n$ are always independent of each other if they are from different uncertainty spaces.

3 Research Methodology

This section presents the research method we applied in the entire thesis. This research work was funded by an EU project, U-Test, which involved two use cases providers from two distinct domains of CPSs and two testbed providers. Such participants provide a rare opportunity to develop and evaluate the proposed approaches with the support of the industry. From the research perspective, the industrial partners and industrial case studies help to identify the research problems based on their needs. In our case, it is particularly helpful since the existing study about uncertainty in CPSs is not mature [4, 37, 38]. From the engineering perspective, the developed approaches may work in theory, but it might be not applicable in the industry due to various factors, such as its usability and scalability. Thus, by conducting the research with the industrial case studies, such problems can be revealed and might be solved further.

3.1 Research activities

The research activities are shown in Figure 4, which has two parallel processes conducted by the researchers and U-Test industrial partners respectively.

The overall research problem was initially defined based on the objectives of the U-Test project (A1 in Figure 4), i.e., *testing CPSs with uncertainties for ensuring that CPSs can operate properly in the presence of uncertainties*.

In order to solve it, the five approaches were proposed in the following activities (A2-A6 in Figure 4). Overall, the activities can be divided into two phases (i.e., the development phase and evaluation phase) for each proposed approach. For the development phase, all *developing* actions (i.e., A2.1, A3.1, A4.1, A5.1, A6.1 in Figure 4) took inputs from the industrial partners whether directly or indirectly. Based on the domain knowledge of the industrial partners, some feedbacks regarding their understanding and representations of uncertainty in CPSs could be collected in the forms of meetings or questionnaires. So, the process of the development of *U-Model, U-RUCM,* and *UncerTum* is iterative, by refining the approaches with the accumulated feedbacks from the industrial partners, shown as A2.2 & B2, A3.2 & B3, and A4.3 & B4 in Figure 4. For the evaluation phase, all approaches were evaluated with at least one industrial case study to assess the performances, shown as A2.3,

A3.4, A4.4, A5.3, and A6.3 in Figure 4. The design and key results of the evaluations with the industrial case studies are reported in Section 5.

The last activity (A7 in Figure 4) is to conclude the research work in the forms of this thesis.

3.2 Implementations

Finally, we produced the five uncertainty-wise approaches with tool supports, denoted by the underlined text in Figure 4. The implementations of all approaches are described in Table 1, together with the accessible links to view or download the implementations.

Table 1. Implementations of the proposed approaches

Approach	**Techniques/tools/languages**	**Implementations**
U-Model	UML [39], OCL [40], IBM Rational Software Architect (RSA)	*U-Model* was implemented by UML class diagram with constraints using RSA, which is available in [41].
U-RUCM	Eclipse [42], LMF, Java	*U-RUCM* was implemented as an eclipse plugin, and the metamodel (*BeliefUCMeta*) is available in [43]. In addition, a video to demonstrate the U-RUCM editor and the formalization from *U-RUCM* to instances of metaclass can be found in [44].
UncerTum	UML [39], RSA	A prototype implementation of *UncerTum* by RSA can be found in [45], which includes a set of UML profiles and model libraries. The detailed specification and guidelines are available in [46].
UncerTolve	RSA, IBM Simulation Toolkit [47], Eclipse OCL [48], Java, Daikon Invariant Detector	An implementation of *UncerTolve* was built on IBM Simulation Toolkit, by integrating eclipse OCL for evaluating the state invariant and daikon for detecting the invariants. More details about the implementation are described in Paper D and [37].
UncerTest	Eclipse, EMF, Java, jMetal [49, 50], Eclipse OCL and JUnit	A prototype implementation of *UncerTest* by java is available in [51]. Note that we used multi-objective search algorithms implemented by jMetal for performing the test case minimization.

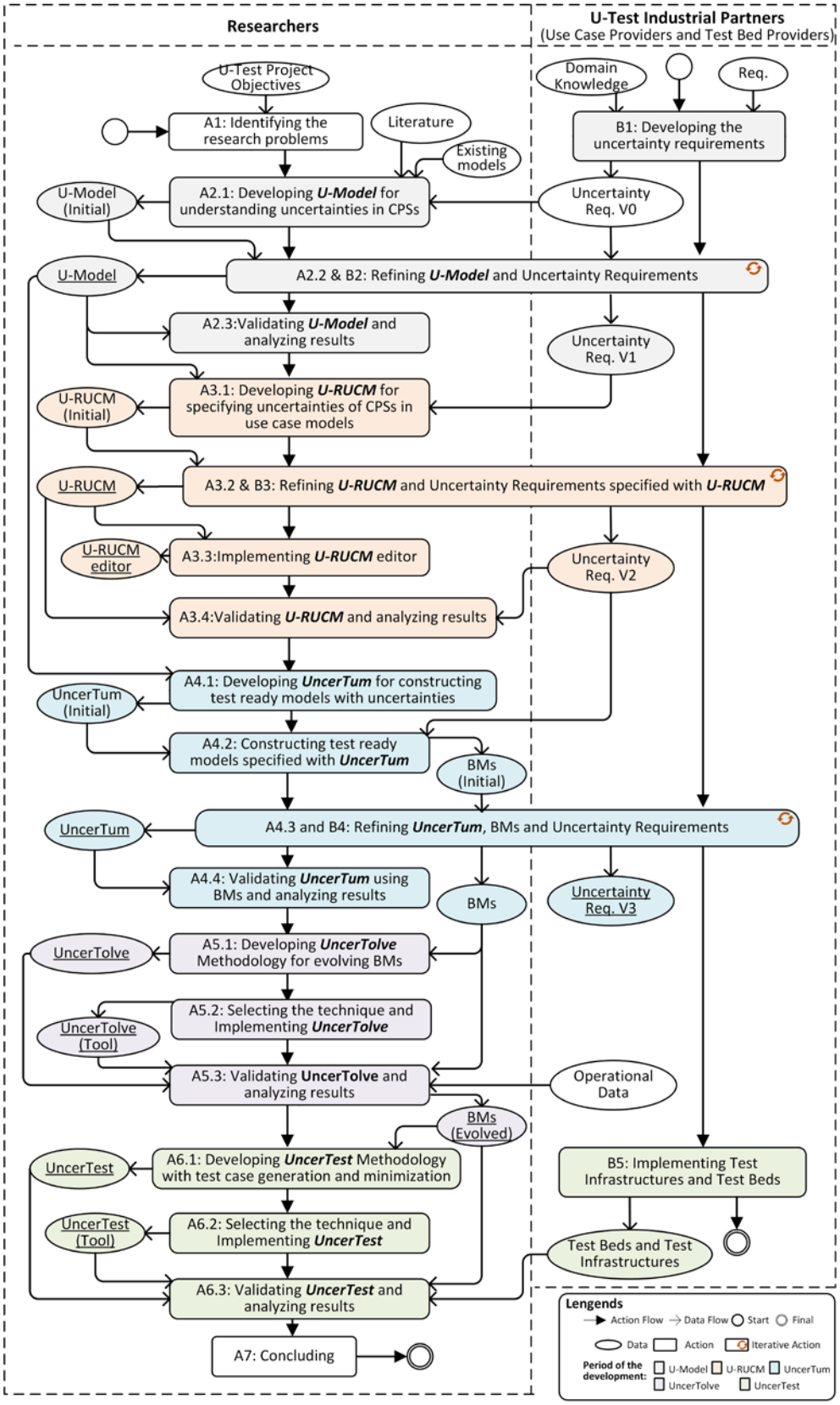


Figure 4. Overview of the research activities in the thesis

4 Uncertainty-wise CPSs Testing Methodologies

Figure 5 presents an overview of the uncertainty-wise CPSs testing by integrating the five approaches: *U-Model* (Section 4.1), *U-RUCM* (Section 4.2), *UncerTum* (Section 4.3), *UncerTolve* (Section 4.4), and *UncerTest* (Section 4.5).

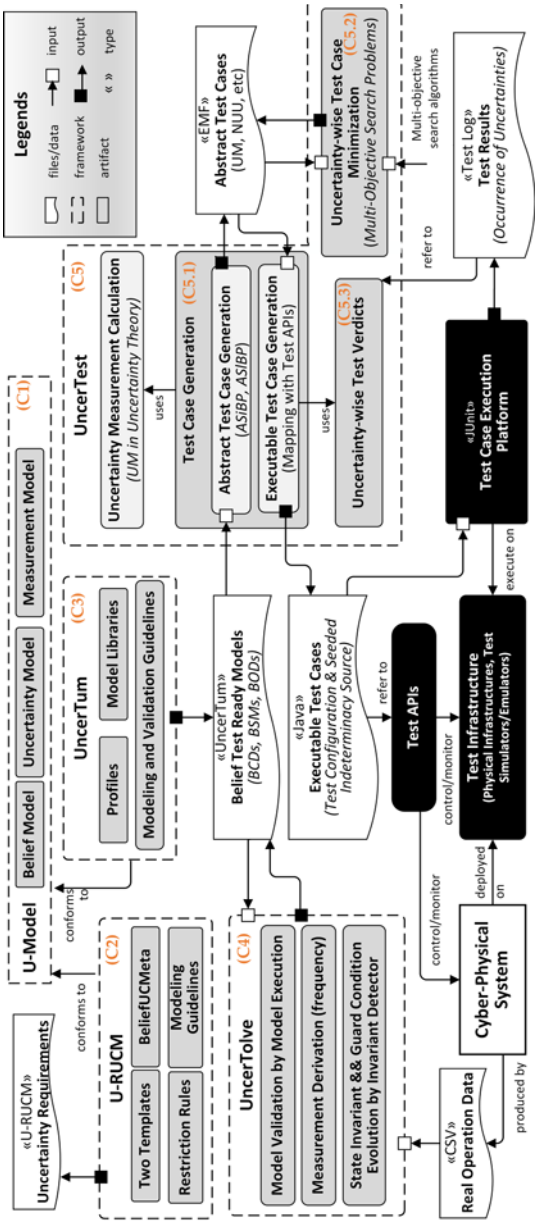


Figure 5. Overview of the Uncertainty-wise Cyber-Physical Systems Testing

To be consistent and systematic to define and identify uncertainties in CPSs, we developed *U-Model* (C1 in Figure 5) for providing a unified conceptual understanding of uncertainty that is the base of the entire research work. Based on *U-Model*, two modeling methodologies, *U-RUCM* (C2 in Figure 5) and *UncerTum* (C3 in Figure 5), were developed

for annotating requirements and test ready models of CPSs with the explicit representation of uncertainty, denoted as *uncertainty requirements* and *belief test ready models* respectively in Figure 5. Moreover, the *belief test ready models* are able to be evolved by *UncerTolve* (C4 in Figure 5) based on real operational data collected from the real applications of CPSs. Furthermore, by taking the *belief test ready models* as input, *UncerTest* can produce a set of *test cases* (i.e., *abstract test cases* and *executable test cases*) embedded with uncertainty information, e.g., the source of uncertainty, the uncertainty measure (measurement) of uncertainty, by the test case generation (C5.1 in Figure 5) and/or uncertainty-wise test case minimization (C5.2 in Figure 5). The *executable test cases* can be executed on the test infrastructure for testing CPSs with the sources of uncertainties seeded in the test environment. After the execution, the occurrences of the uncertainties are allowed to be observed using the *test results* specified with the uncertainty-wise verdicts (C5.3 in Figure 5).

4.1 U-Model

To investigate uncertainty in CPSs, a unified and comprehensive uncertainty conceptual model should be derived. As such, we developed *U-Model* based on the accessible CPSs industrial case studies and a thorough literature review of existing uncertainty models from various domains [52-55], e.g., physics and statistics.

Figure 6 presents the top-level model of *U-Model*, which is composed of three packages: *Belief Model*, *Uncertainty Model*, and *Measure Model*.

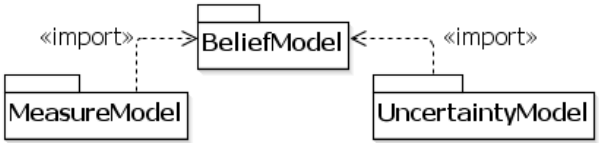


Figure 6. The top-level model of U-Model

U-Model was proposed for taking a subjective approach to representing uncertainty. This means that *uncertainty* is modeled as a state (i.e., worldview) of some agent or agency (referred to as a *BeliefAgent*) that lacks perfect knowledge about some subject of interest. A *BeliefAgent* holds a set of subjective *Belief*s about the subject. A *Belief* is an abstract concept, but it can be expressed in concrete form i.e., an explicit specification (*BeliefStatement*). Thus, *Uncertainty* represents a state of affairs whereby a *BeliefAgent* does not have full

confidence in a *BeliefStatement* that it holds. Note that all subjective concepts mentioned above are represented by the grey-filled boxes in Figure 7. In addition, some objective concepts were also defined, reflecting objective reality, e.g., *Evidence* and *IndeterminacySource*. *Evidence* is inherently an objective phenomenon (e.g., an observation of a real-world event occurrence) that provides information for supporting a *BeliefStatement*. *IndeterminacySource* represents a situation whereby the information required to ascertain the validity of a *BeliefStatement* is indeterminate, resulting in *Uncertainty* being associated with that statement. Moreover, we defined the concept of *Measurement*, representing measured values of the associated *Belief*, *Uncertainty* or *IndeterminacySource*.

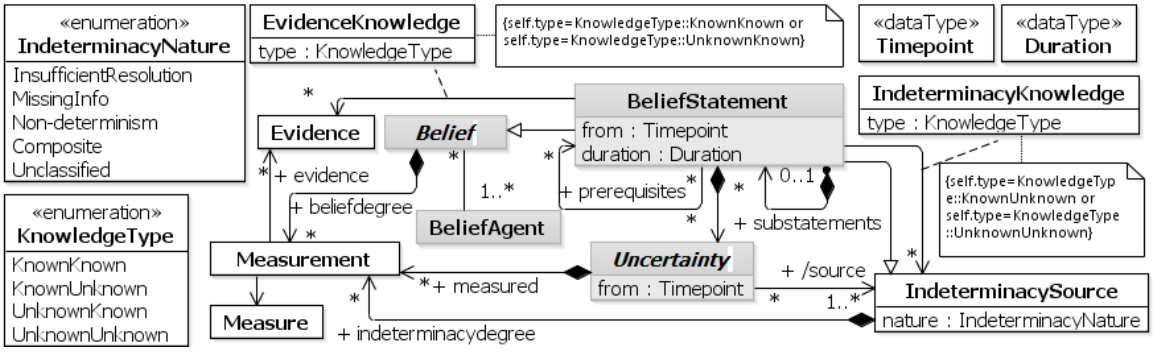


Figure 7. The Core Belief Model

To expand on *Uncertainty* from several different viewpoints and introduce the related abstractions (e.g, *Risk*, *Pattern*), we proposed *Uncertainty Model*, inspired by the concepts defined in the literature on uncertainty [56-60]. Besides, we defined *Measure Model* for introducing the commonly known ways of measuring uncertainty, inspired by the concepts reported in [57-59] and by no means complete. More details are represented in Paper A.

4.2 U-RUCM

Given the complexity and intrinsic uncertainty of CPSs, it is best to address *uncertainty* at the early stage of the software development. Therefore, we developed *U-RUCM* by extending RUCM (Section Section 2.2) for specifying uncertainty in use case models.

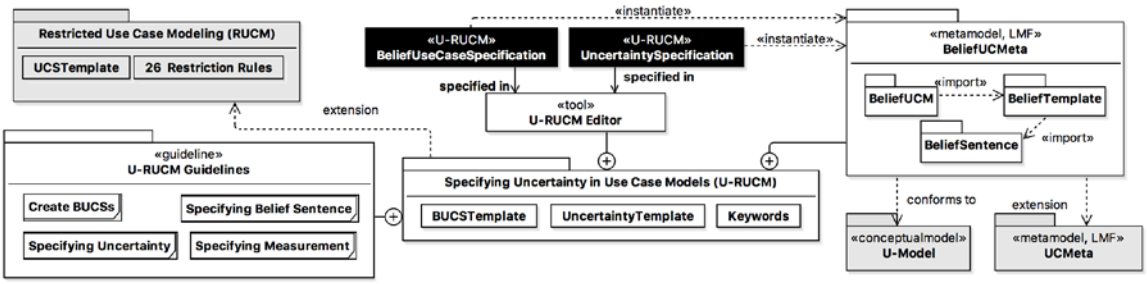


Figure 8. Overview of U-RUCM

Figure 8 shows an overview of *U-RUCM*. In *U-RUCM*, we proposed two new templates (i.e., *BUCSTemplate* and *UncertaintyTemplate*) and two keywords (i.e., *REF* and *URFS*) for stakeholders (i.e., *BeliefAgent*) to specify *belief use case specification*s (BUCSs) and *uncertainty specification*s using natural language as shown in Table 2. *BUCSTemplate* inherits the key heading fields of the RUCM template (grey in Table 2) and introduces six new fields to denote belief and uncertainty information (white in the top of Table 2), e.g., *Indeterminacy Source(s)* indicates a set of indeterminacy sources which resulted in the uncertainties of the BUCS. In addition, *BUCSTemplate* extends all types the flows of events of *RUCM* (*basic flow*, *specific althernative flow*, *bounded althernative flow* and *global althernative flow* as described in Section 2.2) by introducing: (1) a *belief degree*, which measures the degree to which the belief agent(s) believes a specific flow; (2) a new keyword, *URFS*, from which step(s) of a reference flow branches out; (3) the new concept of *alternative steps*, which enables the specification of uncertainties for alternative steps across flows of events; (4) the concepts, *belief sentence* (*BS*) and *belief postcondition*, which provides the capability to annotate sentences in steps of flows and postconditions with belief and uncertainty information. Moreover, we developed *UncertaintyTemplate* to specify the details of an uncertainty in the BS as shown in the bottom of Table 2.

As discussed in Section 2.2, *UCMeta* which formalizes RUCM covers not only all concepts in the RUCM use case template and keywords but also the NL concepts in a sentence. The formalization of *U-RUCM* was implemented as a metamodel, called *BeliefUCMeta*, which extends UCMeta [6, 61] based on *U-Model* [4]. *BeliefUCMeta* imports all elements of UCMeta, thus, *BeliefUCMeta* can naturally benefit from the existing capability of UCMeta for formalizing sentences into instances of metaclass [5], e.g., formalizing sentence constructs such as *noun phrase*, *subject*. As shown in Figure 8, the *U-*

RUCM editor was implemented for providing a graphical user interface to specify *BUCS*s and uncertainty specifications along with the automatic formalization from *U-RUCM* to *BeliefUCMeta*. Besides, a set of guidelines on the usage of *U-RUCM* and the editor were proposed (Figure 8).

Table 2. The U-RUCM templates for specifying BUCS and uncertainty

The template for specifying a BUCS

Use Case Name	The name of the use case. It usually starts with a verb.	
Brief Description	Summarizes the use case in a short paragraph.	
Primary Actor	The actor who initiates the use case.	
Secondary Actor(s)	Other actors the system relies on to accomplish the services of the use case.	
Dependency	Include and extend relationships to other use cases.	
Generalization	Generalization relationships to other use cases.	
Belief Agent(s)	One or more agents who hold a belief about this BUCS.	
Time Point and Duration	The time point when the BUCS/BS is specified and the duration in which the belief agent(s)'s belief on the BUCS holds.	
Belief Degree	The degree to which the belief agent(s) believe the BUCS.	
Indeterminacy Source(s)	The set of indeterminacy sources related to the BUCS (REF is used).	
Evidence	Evidence to support the BUCS, and its contained belief and uncertainty elements (REF is used).	
Belief Precondition	Belief agent(s)' belief on the precondition of the BUCS, which describes what should be true before the use case is executed.	
Belief Basic Flow (*Belief degree*)	Specifies the main successful path, also called "happy path".	
	Steps (numbered)	A set of ordered **belief sentence**s.
	Belief Postcondition	Belief agent(s)' belief on what should be true after the basic flow executes.
Belief Specific Alternative Flow (*Belief degree*)	Applies to one specific step of the reference flow.	
	URFS	The reference flow step where the belief agent(s) believe there are uncertainties.
	Alternative Step	An alternative to the reference flow step.
	Steps (numbered)	A set of ordered belief sentences.
	Belief Postcondition	Belief agent(s)' belief on what should be true after the specific alternative flow executes.
Belief Bounded Alternative Flow (*Belief degree*)	Applies to more than one step of the reference flow, but not all of them.	
	URFS	A list of reference flow steps where the belief agent(s) believe there are uncertainties.
	Alternative Steps	A set of alternatives to the reference flow steps.
	Steps (numbered)	A set of ordered **belief sentences**.
	Belief Postcondition	Belief agent(s)' belief on what should be true after the bounded alternative flow executes.
Belief Global Alternative Flow (*Belief degree*)	Applies to all the steps of the reference flow.	
	Belief Branching Condition	Belief agent(s)' belief on the condition, which describes what should be true when branching from any of the steps of the reference flow.
	Steps (numbered)	The set of ordered **beliefs sentences.**
	Belief Postcondition	Belief agent(s)' belief on what should be true after the global flow executes.

The template of specifying an uncertainty in a belief sentence

Uncertainty Details	Specifies the details of the **uncertainty** in the **belief sentence**.	
	Type	The type of this uncertainty (Occurrence/Content/Time/Environment/GeographicalLocation)
	Indeterminacy Source(s)	The set of indeterminacy sources related to this Uncertainty (REF is used).
	Measure Value	The measurement of this uncertainty.
	Risk	The possible risk led by this uncertainty.
	Pattern	The pattern of the occurrence of this uncertainty

4.3 UncerTum

To enable MBT of CPSs with uncertainty, an uncertainty-wise modeling framework (*UncerTum*) was proposed based on *U-Model* for constructing test ready models with uncertainty.

An overview of *UncerTum* is represented in Figure 9. The *UML Uncertainty Profile* (*UUP*) is the core of *UncerTum*, which defined a set of modeling notations based on *U-Model*. To adopt *U-Model* into *UUP* from the modeling perspective, we made three types of decisions: 1) Some concepts from *U-Model* can be incorporated into *UUP* as it is, e.g., *IndeterminacySource* for specifying the source of uncertainty; 2) Some concepts from *U-Model* do not need to be implemented in *UUP*, e.g., *Belief* is an abstract concept to implicitly describe some phenomena or notions, which is not necessary to be implemented since model can be regarded as the explicit description by modeler ; 3) Some concepts from *U-Model* need to be refined in *UUP*, e.g., *BeliefStatement* was implemented as «BeliefElement» in *UUP* for adjusting to an explicit representation of model elements in the modeling context. Thus, the modeling notations of *UUP* are composed of stereotypes and classes for *Belief*, *Uncertainty*, and *Measurement* (Figure 9) corresponding to *U-Model*. In addition, an *Internal_Library* was implemented to define the necessary enumerations required in *UUP*. Besides, *UncerTum* also consists of a small *CPS Testing Levels* profile which allows the modeler to label the testing level of CPSs (Section 2.1), i.e., *Application*, *Infrastructure*, and *Integration*, just for MBT.

Three model libraries, *Measure Library*, *Pattern Library*, and *Time Library* (Figure 9), were implemented in *UncerTum*, which defined a set of the reusable data types that are commonly used for specifying the characteristics of uncertainty and measuring uncertainty.

Finally, *UncerTum* provides a set of the step-wise guidelines on how to use the modeling notions (*UUP* and *CPS Testing Levels profile*) and datatype (*model libraries*) to construct test ready models with uncertainty, named Belief Test Ready Models (BMs), to enable MBT of CPSs with uncertainty (Figure 9).

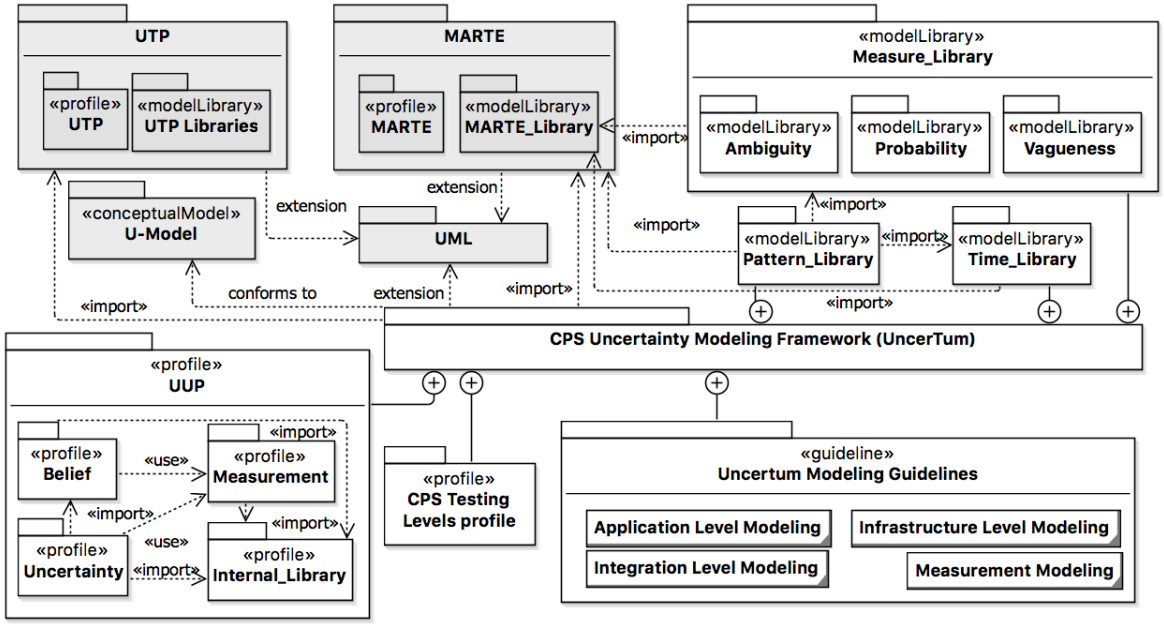


Figure 9. Overview of *UncerTum*

4.4 UncerTolve

UncerTolve is an uncertainty-wise evolution framework that can interactively evolve BMs of CPSs based on real operational data collected from real CPS applications. Thus, the uncertainties in the BMs can be systematically validated and explicitly identified.

A CPS may be deployed to more than one applications from the same or different application domains. One example can be illustrated by the industrial case study of CPSs used in the thesis, GeoSports (Section 5.1.1). GeoSports was deployed on a variety of sports, e.g., Bandy and soccer. Each application regarding a different type of sports corresponds to a different deployment, and real operational data can be collected from already developed applications of CPSs, represented as *D1*, *D2*, … *Dn* (Figure 10). Thus, the collected operational data (i.e., objective evidence) are a valuable resource to validate BMs, including uncertainty information, test oracles, and test data specifications. Such evolved models can be used to generate new test cases to test future developments of the CPS with the explicitly identified subjective and objective uncertainties. *UncerTolve* was so designed to evolve BMs with the real operational data as shown in Figure 10. In *UncerTolve*, three activities and four components were implemented (Figure 10).

The first activity is to construct initial BMs, which contain known subjective uncertainties specified by modelers (i.e., *belief agents* [4]). Moreover, to make BMs executable for the

next activity, a modeling methodology (which extends *UncerTum*) was proposed as a part of *UncerTovle*, particularly for constructing executable features in BMs.

The second activity is to execute (initial) BMs against the real operational data. By the execution, syntactic and semantic errors may be identified by checking the execution logs based on a set of heuristics defined in *UncerTolve*. In addition, *UncerTolve* also calculates the frequency of traversing a state or transition (i.e., objective measurement) based on the execution logs. As shown in Figure 10, (initial) BMs are updated with the removal, modification, and addition of model elements by the model execution component (blue), and the objective uncertainty measurements are appended in BMs by the derivation of measurements (orange).

The third activity is about the invariant inference using dynamic invariant detection techniques [62-64]. In *UncerTolve*, we used Daikon [62] that enables to produce a set of invariants (i.e., test oracle and test data specification) by executing BMs together with the real operational data. To merge the invariants into BMs, *UncerTolve* defined a set of heuristics, for providing recommendations to modeler about restructuring test oracles and test data specifications in BMs.

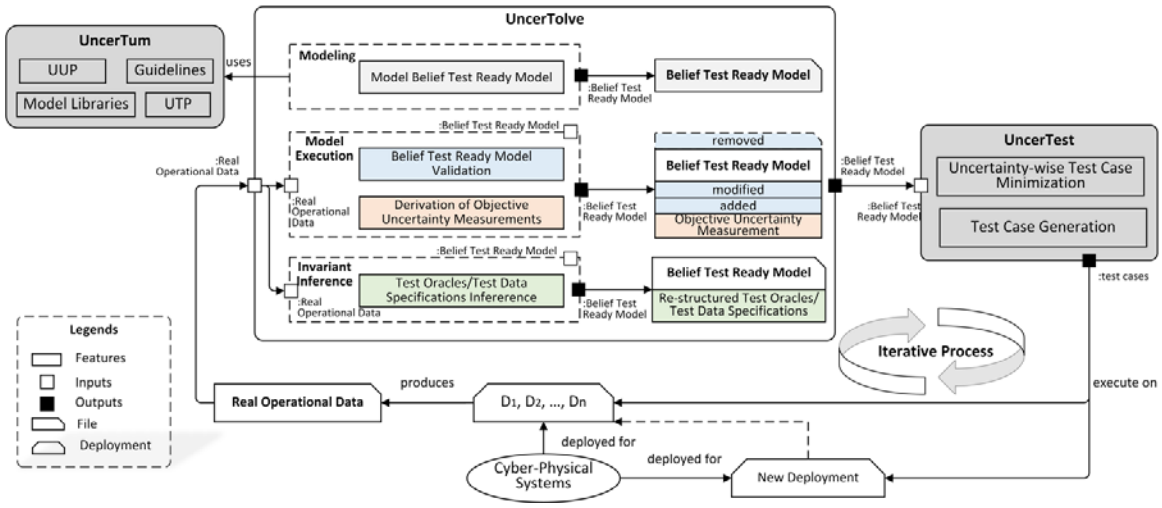


Figure 10. Overview of *UncerTolve*

Note that the model execution activity and invariant inference activity are independent of each other, by evolving BMs from different aspects. But we recommend to apply them sequentially, resulting in the improvement of the overall quality of evolved BMs by avoiding

the syntactic errors in the invariant inference activity. This is also how the industrial case study was conducted in the thesis.

4.5 UncerTest

To perform an automated testing of CPSs with uncertainty, uncertainty is required to be considered in the test case generation, test optimization and test case execution. Driven by such needs, an uncertainty-wise testing framework, named as *UncerTest*, was developed using model-based and search-based software testing techniques. An overview of *UncerTest* is presented in Figure 11, which is composed of three components: *test case generation*, *uncertainty-wise test case minimization* and *uncertainty-wise test verdicts*.

As shown in Figure 11, *UncerTest* facilitates to generate *abstract test cases* and *executable test cases* based on BMs. In *UncerTest*, two strategies, All Simple Belief Paths (*ASiBP*), All Specified Length Belief Paths (*ASlBP*), were proposed for deriving abstract test cases based on BMs, inspired by *Prime Path Coverage* [65] and *Specified Path Coverage* presented in [65]. In addition, *UncerTest* also calculates uncertainty measurements (i.e., *Uncertainty Measure*) for all the generated test cases using *Uncertainty Theory*. Note that each generated test cases contain uncertainty information, e.g., the number of uncertainties, uncertainty measure. Moreover, the *executable test case generation* enables to seed the *executable test cases* with *indeterminacy source*s which might lead to the occurrence of the uncertainties specified in BMs.

To reduce the number of automatically generated abstract test cases when needed, an uncertainty-wise test minimization approach was proposed in *UncerTest* using multi-objective search algorithms (e.g., NSGA-II). As shown in Figure 11, four uncertainty-wise minimization strategies were defined, which were formulated as multi-objective search problems. All of these four problems aim to minimize the number of test cases and maximize the transition coverage. But the problems distinguish themselves by four uncertainty related objectives for different purposes of testing CPSs under uncertainty: (1) *Prob. 1* aims to observe the reaction of CPSs in the presence of maximum uncertainties with minimum possible test cases by covering the maximum number of known uncertainties possible (2nd objective of *Prob. 1* in Figure 11); (2) *Prob. 2* aims to observe the reaction of CPSs in the presence of uncertainties from all the known uncertainty spaces with the minimum possible

test cases by covering at least one uncertainty from each uncertainty spaces (2nd objective of *Prob. 2* in Figure 11); (3) *Prob. 3* aims to test the parts of the system with high degree of confidence by selecting the high value of the uncertainty measure of test cases (2nd objective of *Prob. 3* in Figure 11); (4) *Prob. 4* aims to test the behavior of CPSs under diverse uncertainties with the minimum number of test cases by maximizing the coverage of the different uncertainties (2nd objective of *Prob. 4* in Figure 11).

In *UncerTest*, a set of the uncertainty-wise test verdicts (Figure 11) were defined to assign the result of occurrences of uncertainties during test execution. For instance, we are able to identify the situation whereby known uncertainty occurred under the occurrence of a specified indeterminacy source, based on the test results.

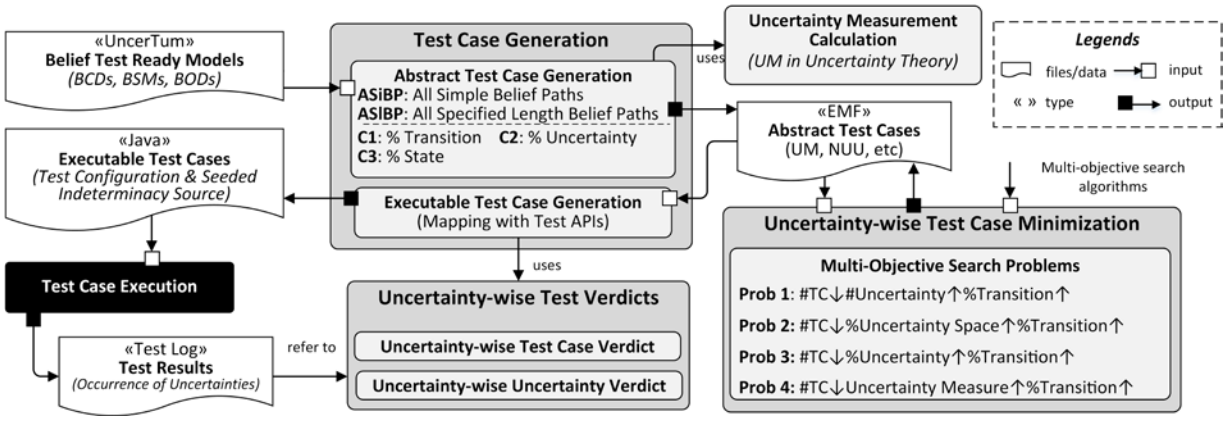


Figure 11. Overview of *UncerTest*

5 Evaluation

This section describes the process and key results of the evaluation for the entire thesis. First, two industrial case studies are described in Section 5.1. Following, the summaries of key results of the evaluation of the five proposed approaches with the industrial case studies are reported from Sections 5.2 to 5.6.

5.1 Case Study

To take the benefits of the U-Test project, two industrial CPS case studies with the available testbeds are accessible for developing and evaluating the research works in the thesis. One is GeoSports system from the healthcare domain described in Section 5.1.1, and another is an automated warehouse system from the automation domain described in Section 5.1.2. Note that each of the proposed approaches was evaluated with at least one of the industrial case studies from the U-Test project.

5.1.1 GeoSports

GeoSports system (GS) by Future Position X (FPX) [66] is to monitor performances and health conditions of each player and the whole team in the game. As shown in Figure 12, the system integrates a set of endpoint devices (i.e., tag), a set of receiver stations (i.e., locator), a set of servers (i.e., QPE server), and a set of applications. Each tag embeds a set of various types of sensors (e.g., accelerometer and gyroscope) for collecting data regarding the individual performance. A locator communicates with surrounding tags by collecting the data at runtime, and the data allows to be visualized using applications of Quuppa, e.g., visualizing a position of a tag. All the data will be maintained in QPE servers.

The case study of GS involved in this thesis is about a sport, Bandy, using the Quuppa system [67]. The testbed provider of the U-Test project, Nordic Med Test [68] developed the testing infrastructure for testing GS with uncertainties for the Bandy setting as shown in Figure 12. Instead of using real player to perform test case execution, the test rig is used to carry the Quuppa tag (Figure 12). In addition, a set of REST APIs was developed for controlling the test rigs and accessing the status of GS.

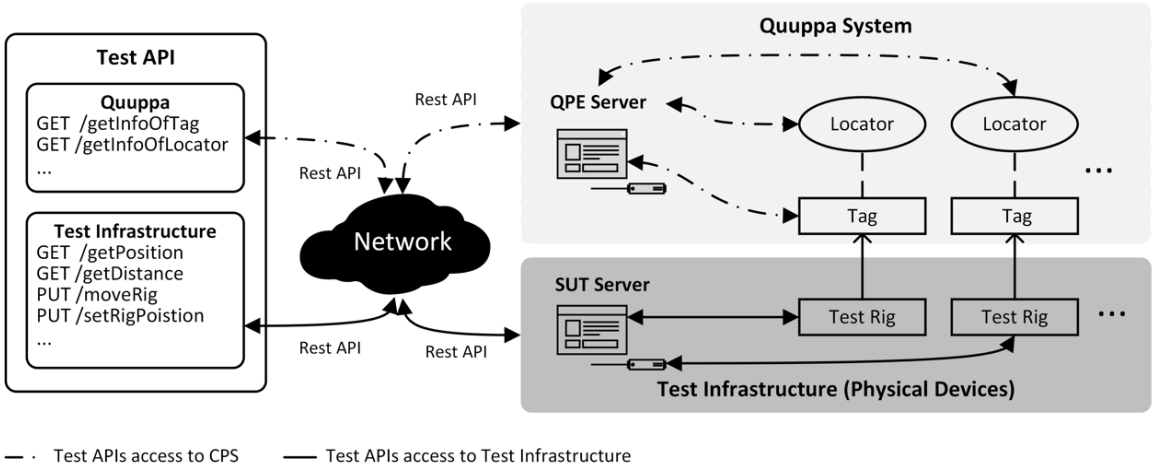


Figure 12. The Test Execution Solution for GS

5.1.2 Automated Warehouse

Automated Warehouse (AW) system by ULMA [69] provides an automated solution to monitor, control, and manage warehouses. Each handling facility (e.g., cranes, conveyors) performs as a physical unit, and together they are deployed to one handling system application.

The case study of AW involved in this thesis includes several key industrial scenarios of an automated warehouse system, e.g., introducing a large number of pallets to the warehouse. Instead of using real devices to test the scenarios, ULMA [69] and IK4-Ikerlan [70] developed relevant simulators and emulators (Figure 13). As shown in Figure 13, the test infrastructure includes two handling systems that are deployed at two different sites (Site 1 and Site 2). In each site, a local superior collects data by monitoring all types of devices and services (e.g., WMS), and uploads the data to a cloud superior through the network. Each physical device is developed as a simulator where services, i.e., WMS and MFC, are deployed on. Also, a set of emulators are developed for manipulating the real physical environment, e.g., putting a pallet on the conveyor. To access the devices, software, and environment, a set of Testing APIs were implemented.

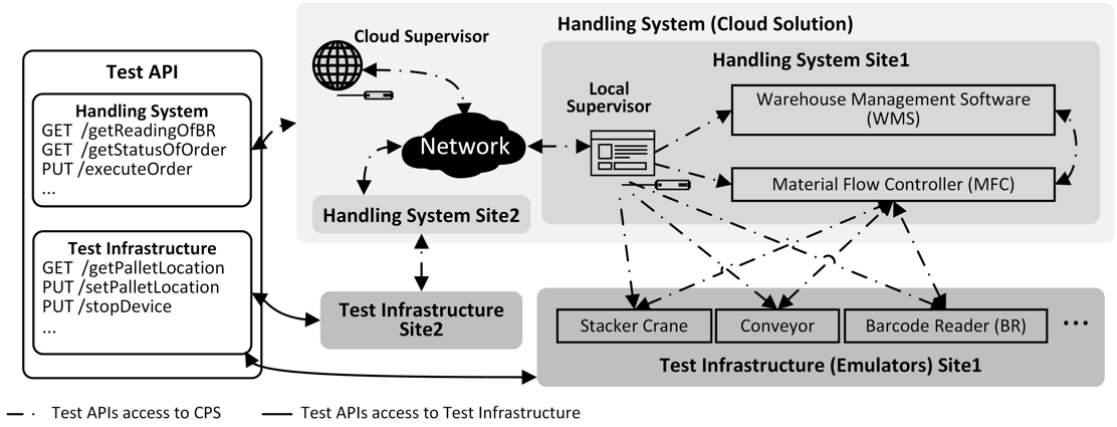


Figure 13. The Test Execution Solution for AW

5.2 U-Model (Paper A)

Understanding Uncertainty in Cyber-Physical Systems: A Conceptual Model. M. Zhang, B. Selic, S. Ali, T. Yue, O. Okariz and R. Norgren. In: Proceedings of the 12th European Conference on Modelling Foundations and Applications (ECMFA 2016), pp. 247-264, 2016. DOI: 10.1007/978-3-319-42061-5_16

In this paper, *U-Model* was proposed with aims of identifying, defining, and classifying uncertainties at the three-logical level of CPSs, i.e., *Application*, *Infrastructure*, and *Integration*.

Figure 14 shows the overall process of the development and validation of *U-Model* and uncertainty requirements. An initial uncertainty conceptual model (*U-Model V.1*) was developed incrementally (A1 and A2 in Figure 14) based on existing models in the literature and other related examples. The activities were mainly conducted by researchers. In parallel, the U-Test industrial partners developed the initial uncertainty requirements (*Reqs V.1*), denoted as B1 in Figure 14. The researchers took these initial uncertainty requirements as inputs for refining the *U-Model* (A3 in Figure 14) by outputting *U-Model V.2*, and further provided some comments on how to improve the requirements using a requirements inspection checklist [71]. Based on the comments, the industrial partners refined the uncertainty requirements (*Reqs V.2*). Then, *Reqs V.2* and *U-Model V.*2 were used as inputs for the onsite workshops conducted by both the researchers and the two industrial partners (A4/B3 in Figure 14). These workshops aimed at discussing the uncertainty requirements,

presenting *U-Model* to the industrial partners and collecting their feedback. The outputs were *U-Model V.3* (the final version of *U-Model*) and *Reqs. V.3*. Using the *U-Model V.3* as a reference model, we identified all uncertainties in uncertainty requirements (*Reqs V.3*) that produced the final version of uncertainty requirements (*Reqs V.4*).

To assess the performance of *U-Model* in terms of identifying uncertainties in CPSs, we compared the identified uncertainties among *Reqs V.1*, *Reqs V.2*, and *Reqs V.4* with GS and AW as discussed in Section 5.1. On average, *U-Model* was managed to identify 61.5% of unknown uncertainties in *Reqs V.4* that weren't explicitly specified in uncertainty requirements (*Reqs V.1*) collected from the two case studies.

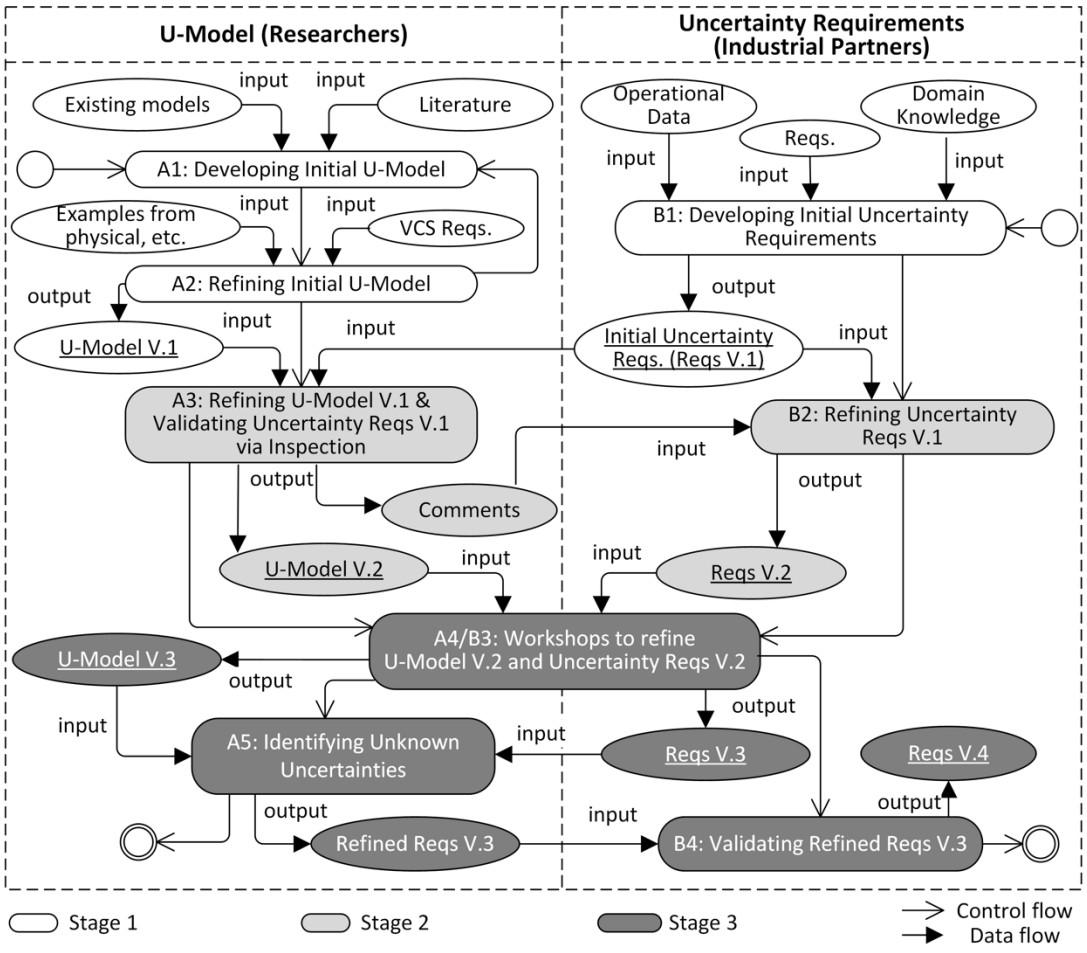


Figure 14. The process of development and validation of *U-Model* and uncertainty requirements

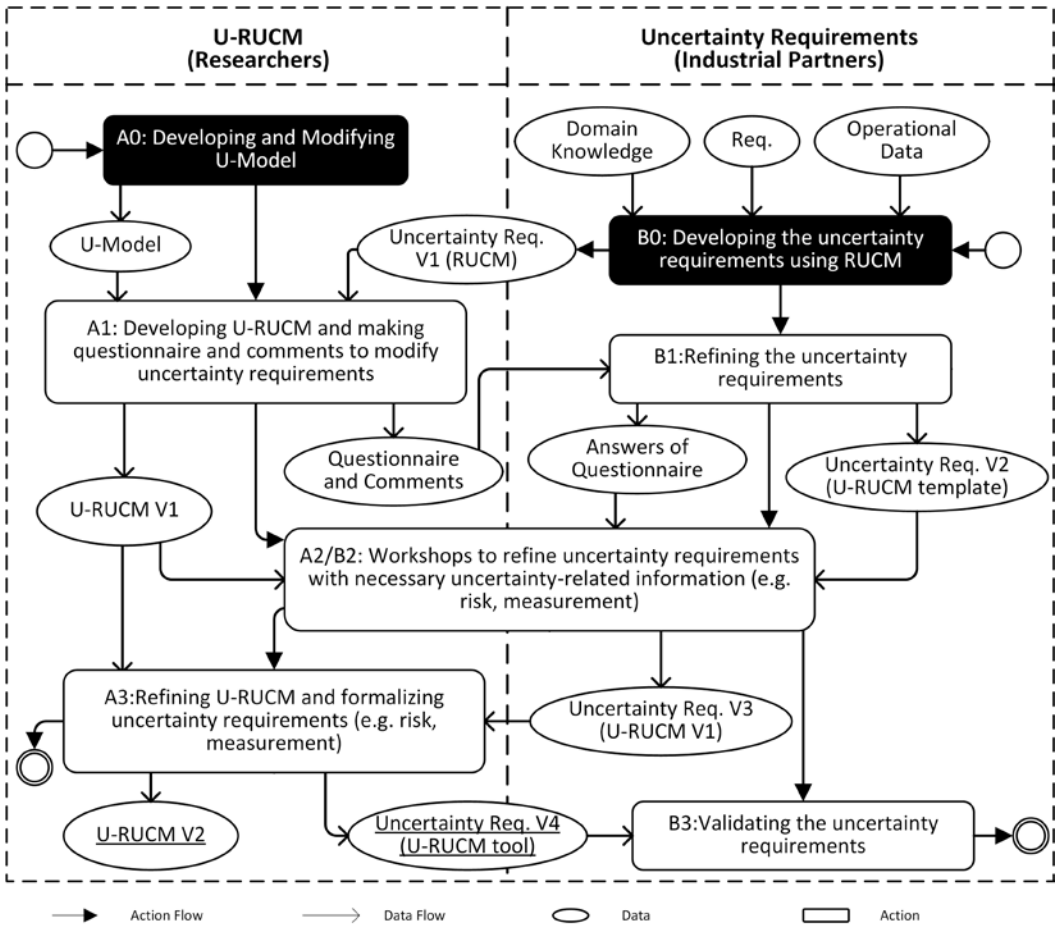
5.3 U-RUCM (Paper B)

"Specifying Uncertainty in Use Case Models". M. Zhang, T. Yue, S. Ali, B. Selic, O. Okariz, R. Norgren and K. Intxausti. Journal paper that has been submitted to the Journal of Systems and Software (JSS), second revision.

In this paper, *U-RUCM* was designed for explicitly specifying uncertainty as a part of requirements of CPSs, which extends a practical use case modeling solution, RUCM. The process of the development of *U-RUCM* is presented in Figure 15.

First, *U-RUCM* was developed based on *U-Model* and existing uncertainty requirements of AW and GS (A1 in Figure 15). In addition, a questionnaire was conducted for collecting information about detailing and quantifying known uncertainties in the requirements with the current development (*U-RUCM V1*). The questionnaire was derived by reviewing use case specifications specified by industrial partners using RUCM (*Uncertainty Req.V1*). According to the questionnaire, the industrial partners refined *Uncertainty Req. V2* using proposed *U-RUCM* templates. Subsequently, several workshops (A2/B2 in Figure 15) were held for each industrial case study with aims of presenting RUCM V1 to the industrial partners and collecting their feedbacks. Based on these workshops, the industrial partners refined *Uncertainty Req. V2* to *Uncertainty Req. V3* based on RUCM V1. Then, the researchers refined *U-RUCM V1* to *U-RUCM V2* (A3 in Figure 15) based on *Uncertainty Req.V3*. When *U-RUCM* is stable, i.e., *U-RUCM V2*, the *U-RUCM* editor was implemented. Consequently, we obtained *Uncertainty Req. V4* specified with the *U-RUCM* editor.

To compare *Uncertainty Req.V1* with *Uncertainty Req. V4* in terms of 20 use cases from GS and AW industrial case studies (Section 5.1), results showed that additional 512% for GS (306% for AW) of uncertainties were learned by applying the *U-RUCM* methodology. This implies that *U-RUCM* performed a significant improvement in dealing with uncertainties in requirements engineering, resulting in a more precise characterization of uncertainties.



- Note that *U-RUCM* was developed following *U-Model*, so *U-Model* is the final version of the development of *U-Model* (Figure 14) and *Uncertainty Req. V1* is the final version of uncertainty requirements specified by *RUCM* (Figure 14).

Figure 15. The process of the development and validation of U-RUCM

5.4 UncerTum (Paper C)

"Uncertainty-Wise Cyber-Physical System Test Modeling". M. Zhang, S. Ali, T. Yue, R. Norgren and O. Okariz. *Journal of Software & Systems Modeling (SOSYM)*. DOI: 10.1007/s10270-017-0609-6

An uncertainty modeling framework, named as *UncerTum*, was proposed in this paper, for supporting MBT of CPSs with explicitly represented uncertainties, e.g., the uncertain behavior of CPSs.

As shown in Figure 16, the process of developing *UncerTum* is iterative, which was intertwined with the incremental development of test ready models. The development of *UncerTum* and test ready models were mainly conducted by researchers (A1-A3 and B1-B3

in Figure 16), by taking inputs of industrial requirements and scenarios provided by industrial use case providers (FPX and ULMA). During the development phase of *UncerTum*, a modeling workshop (A4/B4/C1/D1 in Figure 16) was conducted to initially present *UncerTum* (*UncerTum V2*) and test ready models (*Test Ready Models V1*) specified with *UncerTum*, and collect feedbacks about *UncerTum* and test ready models from the use case providers and testbed providers. Based on the collected feedbacks, the researchers developed *UncerTum V3* and *Test Ready Models V2*. Moreover, two workshops for each case study (A5/B6/C3/D3/E2 in Figure 16) were held with aims of the validation of test ready models and discussion on implementations of test beds and test infrastructures. Thus, *Test Ready Models V2* was modified as *Test Ready Models V3.* At last, the final version of test ready models is *Test Ready Models V4* that was obtained by validating *Test Ready Models V3* using model execution (B7 in Figure 16).

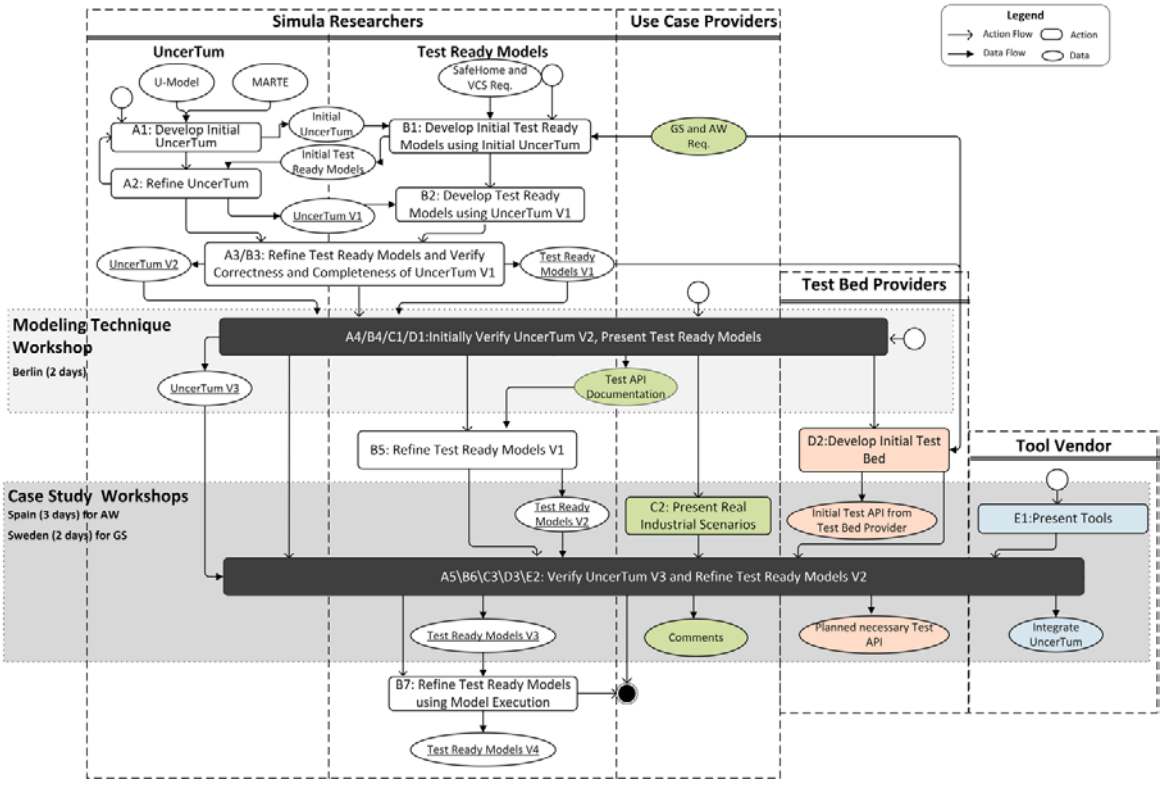


Figure 16. The process of the development of *UncerTum* and test ready models

Except for these two industrial case studies, *UncerTum* was also evaluated with one real-world case study of videoconference system (VCS) [72] developed by Cisco, Norway, and one open source CPS case study that is modified SafeHome system provided in [73].

The evaluation of *UncerTum* with these four case studies (i.e., AW, GS, VCS and SafeHome) was conducted from three main perspectives: (1) *Completeness* of profiles and model libraries: results showed that *UncerTum* is sufficiently complete to model all uncertainties identified in the four case studies; (2) *Effort* required to model uncertainty with *UncerTum*: On average, *UncerTum* required 18.5% more time to apply stereotypes from *UUP* and use datatypes from model libraries on test ready models; and (3) *Correctness* of developed test ready models by validation process: *UncerTum* identified seven types of problems in test ready models from two main categories (i.e., incorrect and incomplete).

5.5 UncerTolve (Paper D)

"Uncertainty-Wise Evolution of Test Ready Models". M. Zhang, S. Ali, T. Yue and R. Norgren. *Journal of Information and Software Technology (IST)*. DOI: 10.1016/j.infsof.2017.03.003

This paper proposed an approach (*UncerTolve*) to interactively evolve BM against real operational data, which is composed of three features: (1) validation of BMs based the real operational data via model execution for correcting syntactic problems, (2) derivation of objective uncertainty measurements in BMs based on the execution log, and (3) inference of state invariants and guards of transitions in BMs by a dynamic invariant detector. To evaluate *UncerTolve*, we applied *UncerTolve* on one industrial case study[1], GS, by the following steps: (1) we developed initial BMs for GS, denoted as *BM V1*; (2) we executed the initial BMs via model execution for validating BMs against real operational data of GS and updating objective measurements of uncertainty, which produced the evolved BMs, denoted as *BM V2*; (3) then we used Daikon to derive invariants based on the data that were used to further evolve BMs (denoted as BM V3) by integrating the derived invariants into the BMs; (4) since the integration may lead to new errors, we performed the model validation via model execution again. After these four steps, we obtained the final version of BMs, denoted as BM V3'.

[1] Since the operational data is not available for AW case study, we only applied GS to evaluate *UncerTolve*.

To assess the performance of *UncerTolve*, we collected the number of evolved model elements regarding *belief element*, *state*, and *transition*. By comparing *BM V1* with BM V3', we found that *UncerTolve* managed to evolve 51% of *belief elements*, 18% of *states* and 21% of *transitions*.

5.6 UncerTest (Paper E)

"Uncertainty-wise Test Case Generation and Minimization for Cyber-Physical Systems: A Multi-Objective Search-based Approach". M. Zhang, S. Ali and T. Yue. Journal paper that has been submitted to ACM Transactions Software Engineering and Methodology (TOSEM).

UncerTest is an approach for involving uncertainty aspect into MBT of CPSs. In *UncerTest*, two test case generation strategies (*ASiBP* and *ASlBP*) and four uncertainty-wise test case minimization problems (*Prob*1 -- *Prob4*) were developed. By combining [2] generation and minimization to get a test set, five combined strategies were identified in total: 1) *Str1*: *ASiBP*, 2) *Str2*: *ASlBP* and *Prob1*, 3) *Str3*: *ASlBP* and *Prob2*, 4) *Str4*: *ASlBP* and *Prob3*, 5) *Str5*: *ASlBP* and *Prob4*.

First, to evaluate the strategies (*Str2* -- *Str5*) that require test case minimization, we conducted an experiment by investigating uncertainty-wise test case minimization problems with eight multi-objective search algorithms (i.e., NSGA-II [74], NSGA-III [75], MOCell [76, 77], SPEA2 [78], CellDE [78], AbYSS [79], GDE3 [80] and SMPSO [81]) and random search algorithm using five use cases from two industrial case studies, i.e., four for AW and one for GS. Such experiment aims at answering two research questions (RQ1 and RQ2), and results are also reported as follows.

RQ1: *How does the selected multi-objective search algorithms compare to RS regarding solving uncertainty-wise minimization problems (Str2 -- Str5)?*

Results for RQ1: Results showed RS obtained the low confidence in order to become the best algorithm for *Str2* -- *Str5* among two case studies, which implies that problems

[2] The number of test cases generated by *ASiBP* is small in our case study, *ASiBP* does not need to combine with test case minimizations.

(*Prob1* -- *Prob4*) couldn't have been solved effectively with RS and thus provides the evidence of using complex multi-objective search algorithms.

RQ2*: Which algorithm is the best among selected ones to solve uncertainty-wise minimization problems (Str2 -- Str5) respectively?*

Results for RQ2: To obtain the result of RQ2, 36 pair-wise comparisons among selected algorithms (C_2^9) need to be made for each of five use cases for four uncertainty-wise minimization problems (*Str2* -- *Str5*), and the total is 720 (36×4×5). Results showed SPEA2 is the consistently best, or the second best (only in three instances). Thus, SPEA2 was recommended to solve uncertainty-wise minimization problems (*Str2* -- *Str5*) to find the most optimal test set.

The next evaluation was designed about assessing the performance of the test set obtained by uncertainty-wise strategies *(Str1* -- *Str5)* in term of the cost-effectiveness of observing uncertainties in CPSs with two industrial case studies (Section 5.1). For measuring *cost*, we defined one metric, *ET*, that is the time taken for executing the test set produced by one of the uncertainty-wise strategies *(Str1* -- *Str5)*. To measure *effectiveness*, two aspects were mainly considered in the evaluation: *known uncertainties observed* and *unknown uncertainties detected*.

For observing uncertainties in CPSs, the test set obtained by each uncertainty-wise strategy for each case study was required to be executed on the test infrastructures as shown in Figure 12 and Figure 13. Based on results, we were able to answer the following research question.

RQ3*: Which uncertainty-wise strategy (Str2 -- Str5) is effective to discover uncertainties in the real CPS?*

Results for RQ3: By analysing the results of the execution of test sets applied with the five strategies (*Str2* -- *Str5*) for the five use cases, *Str2* with *SPEA2* performed best, which observed on average 51% more uncertainties due to unknown indeterminacy sources as compared to the rest of test strategies for all the use cases. Moreover, it managed to discover 13 unknown uncertainties due to unknown indeterminacy sources across all the use cases.

6 Discussion

Uncertainty-wise Modeling. Currently our modeling methodologies are for specifying uncertainty at two phases: requirement engineering (use case models with *U-RUCM*) and test design (test ready models with *UncerTum*). From the uncertainty perspective, measurements, sources and characteristics of uncertainties are all allowed to be specified in models, which help to support uncertainty related reasoning (e.g., discovering unknown uncertainties and inferring measurements) and analyses (e.g., risk and reliability analyses).

In terms of use case modeling, our methodology (i.e., *U-RUCM*) is applicable for specifying uncertainty at the use case level (i.e., use case specification), scenario level (i.e., flow of events) and action level (i.e., sentence). In addition, we introduced the causality of uncertainty occurrences into the structure of use case specifications. That is, a use case level uncertainty can originate from one or more indeterminacy sources; a scenario level uncertainty can originate from an action level uncertainty; an action level uncertainty can originate from an indeterminacy source or another action level uncertainty. This can help to validate specified uncertainties and derive new uncertainty at the structure level.

In term of test modeling, with the defined profile (UUP) and model libraries, our methodology (i.e., *UncerTum*) can be used to specify uncertainty information (e.g., sources of uncertainty) in structure models (i.e., class diagram), and capture uncertain behaviors of system under test in behavior models (i.e., state machines). Such test ready models can be used to generate test cases and optimize test process with for example *UncerTest*, and can be evolved when taking uncertainty into account with for example *UncerTolve*. In the future, investigation should be conducted to integrate uncertainty modeling with other modeling notations such as activity diagrams and SysML.

Uncertainty-wise Modeling with U-Model. *U-RUCM* and *UncerTum* were proposed by establishing on *U-Model* for representing uncertainty and its characteristics in use case models and test ready models. Based on the evaluation of these two approaches with two industrial case studies, we observed that with our uncertainty modeling methodologies, all the identified uncertainties for the two case studies can be adequately captured. It gives us an indication that *U-Model* as an uncertainty conceptual model is sufficiently complete for classifying and characterizing uncertainty in the context of CPSs. Therefore, in the future,

we are confident that *U-Model* can be adopted or adapted for enabling uncertainty modeling at other phases of the software/system development lifecycle such as design, and it also can be used to support model-based uncertainty testing with other modeling languages such as SysML.

Uncertainty-wise Evolution. Currently *UncerTolve* managed to evolve model elements (e.g., states, transition, uncertainty) of behavior models (i.e., state machines) specified with *UncerTum*, but it is only applicable when real operational data is available. To ensure the quality of test ready models, it is required to develop an approach to evolve the models with uncertainty when lacking real operational data. One possible solution is to proactively evolve such models by directly executing the models on test infrastructures with seeded indeterminacy sources. To make the execution cost-effective, an execution strategy can be also evolved with the help of artificial intelligence techniques (e.g., genetic programming [82]) for achieving the high efficiency of detecting unknown behaviors (e.g., uncertainty).

Uncertainty-wise Testing with the UncerTum models. Our methodology supports to introduce uncertainty to test generation, test optimization and test execution. More specifically, for test case generation, we applied Uncertainty Theory to calculate measurements for each generated test cases. But Uncertainty Theory [31] is applicable when only subjective measurements are accessible. We will investigate more about measuring uncertainty and its derived test cases with different theories (e.g., Probability theory [28, 29], Dempster–Shafer theory [83]). For test optimization, we applied multi-objective search by reformulating test minimization problems as search problems. Each of the problems involves one uncertainty related objective for testing CPSs with different settings of uncertainties from thee perspectives of amount, coverage, measurement and space of uncertainty. In the future, the correlations between these uncertainty related objectives and unknown uncertainty detection will be investigated further. For test case execution, our methodology supports the generation of executable test cases with seeded indeterminacy sources, resulting in executing test cases with different environments and observing occurrences of uncertainties along with their indeterminacy sources. Doing so helps to observe previously unknown uncertainties during execution, study and examine relationships between uncertainties and their indeterminacy sources.

CPS Uncertainty-wise Testing. In this thesis, our empirical study was mainly conducted with two CPS domains (Automation, Healthcare) and thus additional case studies from more CPS domains are required to further generalize the evalutaion results.

Applying Uncertainty-wise approaches in industry. By benefiting from the U-Test project, all the proposed uncertainty-wise approaches were evaluated with the industrial case studies. The evaluation results are summarized in Section 5, which gives us an indication that the uncertainty-wise approaches are to certain extent applicable in industry for testing industrial CPSs with uncertainty. Encouraged by the positive feedback from the evaluation and industrial partners, our test modelling (*UncerTum*) and testing methodology (*UncerTest*) have been partially implemented in a commercial MBT tools, CertifyIT[3]. This provides a unique opportunity to attract more interests of industry in the uncertainty-wise modelling and testing and exploit their potential in real industrial contexts in the future.

[3] http://www.smartesting.com/en/certifyit/

7 Conclusion and Future Work

Uncertainty is inherent in Cyber-physical systems (CPSs) due to various reasons, e.g., CPSs often operate in unpredictable environments. Therefore, a systematic approach to handling uncertainty in CPSs is required, for ensuring that CPSs are capable of operating properly in the presence of uncertainties during their actual operations.

In the thesis, an uncertainty-wise CPSs testing was realized via the integration of the five proposed approaches: (1) *U-Model*, which is an unified and comprehensive uncertainty conceptual model; (2) *U-RUCM*, which facilitates a more precise characterization of uncertainty in requirement engineering; (3) *UncerTum*, which provides the modeling features to construct test ready models with uncertainty; (4) *UncerTolve*, which enables the evolution of the test ready models and subjective uncertainties; and (5) *UncerTest*, which offers a cost-effective manner to test CPSs under uncertainty by taking uncertainty into consideration in test generation, test optimization, and test case execution.

In total, by conducting the evaluation with at least one industrial case study for each of the five approaches, we found that (1) the modeling methodologies (i.e., *U-RUCM* and *UncerTum*) are sufficiently complete and comprehensive with the support of guidelines; (2) the testing methodology (i.e., *UncerTest*) implemented a cost-effective solution for observing CPSs under uncertainty; (3) all approaches together served a more comprehensive identification of uncertainty in CPSs; and (4) the overall uncertainty-wise CPSs testing is systematic by considering uncertainty in requirements elicitation, test models construction, test design, test optimization and test case execution.

In the future, some possible directions may be conducted: (1) evolving uncertainty in requirements specified with U-RUCM, by reasoning relationships (e.g., causality, dependency) and measurements among uncertainties in use case specifications, sentences and parts of sentences; (2) instead of evolving BMs against the past evidence (e.g., collected real operational data), developing the proactive evolution of BMs by directly executing models on the test infrastructure with seeded indeterminacy sources; (3) further studying correlations between the uncertainty-related objective (e.g., UM) and the identification of unknown uncertainties.

Reference

[1] D. B. Rawat, J. J. Rodrigues, and I. Stojmenovic, Cyber-physical systems: from theory to practice, CRC Press, 2015.

[2] S. Sunder, Foundations for Innovation in Cyber-Physical Systems, in: Proceedings of the NIST CPS Workshop, Chicago, IL, USA, 2012.

[3] E. Geisberger, and M. Broy, Living in a networked world: Integrated research agenda Cyber-Physical Systems (agendaCPS), Herbert Utz Verlag, 2015.

[4] M. Zhang, B. Selic, S. Ali, T. Yue, O. Okariz, and R. Norgren, 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, 2016.

[5] T. Yue, L. C. Briand, and Y. Labiche, aToucan: An Automated Framework to Derive UML Analysis Models from Use Case Models, ACM Transactions on Software Engineering and Methodology (TOSEM), vol. 24, no. 3 (2015) 13.

[6] T. Yue, L. C. Briand, and Y. Labiche, Facilitating the transition from use case models to analysis models: Approach and experiments, ACM Transactions on Software Engineering and Methodology (TOSEM), vol. 22, no. 1 (2013) 5.

[7] OMG, "Meta Object Facility (MOF) Core Specification (Version 2.4.2)," 2014, http://www.omg.org/spec/MOF/2.4.2.

[8] T. Yue, S. Ali, and M. Zhang, "Applying A Restricted Natural Language Based Test Case Generation Approach in An Industrial Context," *International Symposium on Software Testing and Analysis (ISSTA)*, 2015.

[9] C. Wang, F. Pastore, A. Goknil, L. Briand, and Z. Iqbal, Automatic generation of system test cases from use case specifications, in Proceedings of the 2015 International Symposium on Software Testing and Analysis, Baltimore, MD, USA, 2015, pp. 385-396.

[10] J. Wu, S. Ali, T. Yue, J. Tian, and C. Liu, Assessing the Quality of Industrial Avionics Software: An Extensive Empirical Evaluation, Empirical Software Engineering (2016).

[11] T. Yue, H. Zhang, S. Ali, and C. Liu, A Practical Use Case Modeling Approach to Specify Crosscutting Concerns: Industrial Applications, 2015.

[12] T. Yue, L. Briand, and Y. Labiche, A Use Case Modeling Approach to Facilitate the Transition Towards Analysis Models: Concepts and Empirical Evaluation, in: A. Schürr and B. Selic, eds. Model Driven Engineering Languages and Systems (MODELS 2009), 2009 2009.

[13] M. Shafique, and Y. Labiche, A systematic review of model based testing tool support, Carleton University, Canada, Tech. Rep. Technical Report SCE-10-04 (2010) 01-21.

[14] P. C. Jorgensen, The Craft of Model-based Testing, CRC Press, 2017.

[15] M. Utting, and B. Legeard, Practical model-based testing: a tools approach, Morgan Kaufmann, 2010.

[16] P. McMinn, Search-based software test data generation: A survey, Software Testing Verification and Reliability, vol. 14, no. 2 (2004) 105-156.

[17] M. Harman, P. McMinn, J. T. De Souza, and S. Yoo, Search based software engineering: Techniques, taxonomy, tutorial, *Empirical software engineering and verification*, pp. 1-59: Springer, 2012.

[18] M. Harman, S. A. Mansouri, and Y. Zhang, Search-based software engineering: Trends, techniques and applications, ACM Comput. Surv., vol. 45, no. 1 (2012) 1-61, 10.1145/2379776.2379787.

[19] P. McMinn, Search-Based Software Testing: Past, Present and Future, in: 2011 IEEE Fourth International Conference on Software Testing, Verification and Validation Workshops. pp. 153-163, 2011 21-25 March 2011.

[20] M. Harman, Y. Jia, and Y. Zhang, Achievements, Open Problems and Challenges for Search Based Software Testing, in: 2015 IEEE 8th International Conference on Software Testing, Verification and Validation (ICST). pp. 1-12, 2015 13-17 April 2015.

[21] W. Afzal, R. Torkar, and R. Feldt, A systematic review of search-based testing for non-functional system properties, Information and Software Technology, vol. 51, no. 6 (2009) 957-976, 2009/06/01/, https://doi.org/10.1016/j.infsof.2008.12.005.

[22] S. Ali, L. C. Briand, H. Hemmati, and R. K. Panesar-Walawege, A systematic review of the application and empirical investigation of search-based test case generation, IEEE Transactions on Software Engineering, vol. 36, no. 6 (2010) 742-762.

[23] M. Harman, S. A. Mansouri, and Y. Zhang, Search based software engineering: A comprehensive analysis and review of trends techniques and applications, Department of Computer Science, King's College London, Tech. Rep. TR-09-03 (2009).

[24] S. Yoo, and M. Harman, Regression testing minimization, selection and prioritization: a survey, Software Testing, Verification and Reliability, vol. 22, no. 2 (2012) 67-120.

[25] M. Khatibsyarbini, M. A. Isa, D. N. A. Jawawi, and R. Tumeng, Test case prioritization approaches in regression testing: A systematic literature review, Information and Software Technology (2017).

[26] K. Deb, and K. Deb, Multi-objective Optimization, *Search Methodologies: Introductory Tutorials in Optimization and Decision Support Techniques*, E. K. Burke and G. Kendall, eds., pp. 403-449, Boston, MA: Springer US, 2014.

[27] S. Yoo, and M. Harman, Pareto efficient multi-objective test case selection, in. pp. 140-150, 2007.

[28] P. C. Jorgensen, Software testing: a craftsman's approach, CRC press, 2016.

[29] Z. Xuemei, T. Xiaolin, and P. Hoang, Considering fault removal efficiency in software reliability assessment, IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans, vol. 33, no. 1 (2003) 114-120, 10.1109/TSMCA.2003.812597.

[30] B. Liu, Why is there a need for uncertainty theory, Journal of Uncertain Systems, vol. 6, no. 1 (2012) 3-10.

[31] B. Liu, Uncertainty theory, Springer, 2015.

[32] Y. Zhu, UNCERTAIN OPTIMAL CONTROL WITH APPLICATION TO A PORTFOLIO SELECTION MODEL, Cybernetics and Systems, vol. 41, no. 7 (2010) 535-547, 2010/09/24, 10.1080/01969722.2010.511552.

[33] L. Yang, K. Li, and Z. Gao, Train Timetable Problem on a Single-Line Railway With Fuzzy Passenger Demand, IEEE Transactions on Fuzzy Systems, vol. 17, no. 3 (2009) 617-629, 10.1109/TFUZZ.2008.924198.

[34] J. Peng, Risk metrics of loss function for uncertain system, Fuzzy Optimization and Decision Making, vol. 12, no. 1 (2013) 53-64, 2013//, 10.1007/s10700-012-9146-5.

[35] S. Han, Z. Peng, and S. Wang, The maximum flow problem of uncertain network, Information Sciences, vol. 265 (2014) 167-175, 5/1/, http://dx.doi.org/10.1016/j.ins.2013.11.029.

[36] W. Rudin, Real and complex analysis, Tata McGraw-Hill Education, 1987.

[37] M. Zhang, S. Ali, T. Yue, and R. Norgre, Uncertainty-wise evolution of test ready models, Information and Software Technology (2017), http://dx.doi.org/10.1016/j.infsof.2017.03.003.

[38] M. Zhang, S. Ali, T. Yue, R. Norgren, and O. Okariz, Uncertainty-Wise Cyber-Physical System test modeling, Software & Systems Modeling (2017), 2017/07/25, 10.1007/s10270-017-0609-6.

[39] OMG, Unified Modeling Language 2.5 (UML), June 2015, http://www.omg.org/spec/UML/.

[40] OMG, "Object Constraint Language (OCL)," 2014, http://www.omg.org/spec/OCL/.

[41] "U-Model," accessed; http://www.zen-tools.com/rucm/metamodels/U_Model/content/_Z4.v.f.wA.h.kE.eW31.c7B.e8.r.j_Q_root.html.

[42] accessed.

[43] "BeliefUCMeta," accessed; http://www.zen-tools.com/rucm/metamodels/belief_ucmeta/content/_.h.n.c.j.cJ.nFE.eW.a-.f8-.j.xNWI.w_root.html.

[44] "U-RUCM: Specifying Uncertainty in Use Case Models," accessed; http://zen-tools.com/rucm/U_RUCM.html.

[45] "UncerTum," accessed; https://bitbucket.org/ManZH/uncertum-v1.

[46] M. Zhang, S. Ali, T. Yue, and R. Norgre, An Integrated Modeling Framework to Facilitate Model-Based Testing of Cyber-Physical Systems under Uncertainty,

Technical report 2016-02, Simula Research Laboratory, 2016; https://www.simula.no/publications/integrated-modeling-framework-facilitate-model-based-testing-cyber-physical-systems.

[47] "IBM RSA Simulation Toolkit," accessed 2016; http://www-03.ibm.com/software/products/en/ratisoftarchsimutool.

[48] "Eclipse OCL," accessed 2016; http://www.eclipse.org/modeling/mdt/?project=ocl#ocl.

[49] "jMetal," accessed 2016; http://jmetal.sourceforge.net/.

[50] J. J. Durillo, and A. J. Nebro, jMetal: A Java framework for multi-objective optimization, Advances in Engineering Software, vol. 42, no. 10 (2011) 760-771.

[51] "UncerTest: an uncertainty-wise testing tool for test generation and optimization," accessed; https://bitbucket.org/ManZH/uncertest-v1.

[52] C. Tannert, H. D. Elvers, and B. Jandrig, The ethics of uncertainty, EMBO reports, vol. 8, no. 10 (2007) 892-896.

[53] M. H. Mishel, Uncertainty in illness, Image: The Journal of Nursing Scholarship, vol. 20, no. 4 (1988) 225-232.

[54] A. S. Babrow, C. R. Kasch, and L. A. Ford, The many meanings of uncertainty in illness: Toward a systematic accounting, Health communication, vol. 10, no. 1 (1998) 1-23.

[55] P. K. Han, W. M. Klein, and N. K. Arora, Varieties of Uncertainty in Health Care A Conceptual Taxonomy, Medical Decision Making, vol. 31, no. 6 (2011) 828-838.

[56] G. Bammer, and M. Smithson, Uncertainty and risk: multidisciplinary perspectives, Routledge, 2012.

[57] D. V. Lindley, Understanding uncertainty (revised edition), John Wiley & Sons, 2014.

[58] K. Potter, P. Rosen, and C. R. Johnson, From quantification to visualization: A taxonomy of uncertainty visualization approaches, *Uncertainty Quantification in Scientific Computing*, pp. 226-249: Springer, 2012.

[59] B. N. Taylor, Guidelines for Evaluating and Expressing the Uncertainty of NIST Measurement Results (rev, DIANE Publishing, 2009.

[60] S. Wasserkrug, A. Gal, and O. Etzion, A taxonomy and representation of sources of uncertainty in active systems, *Next Generation Information Technologies and Systems*, pp. 174-185: Springer, 2006.

[61] T. Yue, L. Briand, and Y. Labiche, aToucan: An Automated Framework to Derive UML Analysis Models from Use Case Models, ACM Transactions on Software Engineering and Methodology (2014).

[62] M. D. Ernst, J. Cockrell, W. G. Griswold, and D. Notkin, Dynamically discovering likely program invariants to support program evolution, IEEE Transactions on Software Engineering, vol. 27, no. 2 (2001) 99-123, 10.1109/32.908957.

[63] C. Csallner, N. Tillmann, and Y. Smaragdakis, DySy: dynamic symbolic execution for invariant inference, in Proceedings of the 30th international conference on Software engineering, Leipzig, Germany, 2008, pp. 281-290.

[64] I. Krka, Y. Brun, D. Popescu, J. Garcia, and N. Medvidovic, Using dynamic execution traces and program invariants to enhance behavioral model inference, in: 2010 ACM/IEEE 32nd International Conference on Software Engineering. pp. 179-182, 2010 2-8 May 2010.

[65] P. Ammann, and J. Offutt, Introduction to software testing, Cambridge University Press, 2016.

[66] "Future Position X," accessed 2017; http://www.fpx.se/.

[67] "Quuppa - Do more with Location," accessed 2017; http://quuppa.com/.

[68] "Nordic Med Test," accessed 2017; http://www.nordicmedtest.se/.

[69] "ULMA Handling System," accessed 2017; http://www.ulmahandling.com/en/.

[70] "IK4-IKERLAN," accessed 2017; http://www.ikerlan.es/eu/.

[71] T. Yue, L. C. Briand, and Y. Labiche, Facilitating the Transition From Use Case Models to Analysis Models: Approach and Experiments, ACM Transactions on Software Engineering and Methodology, vol. 22, no. 1 (2013).

[72] S. Ali, L. C. Briand, and H. Hemmati, Modeling robustness behavior using aspect-oriented modeling to support robustness testing of industrial systems, Software & Systems Modeling, vol. 11, no. 4 (2012) 633-670.

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

[74] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, A fast and elitist multiobjective genetic algorithm: NSGA-II, IEEE transactions on evolutionary computation, vol. 6, no. 2 (2002) 182-197.

[75] K. Deb, and H. Jain, An Evolutionary Many-Objective Optimization Algorithm Using Reference-Point-Based Nondominated Sorting Approach, Part I: Solving Problems With Box Constraints, IEEE Transactions on Evolutionary Computation, vol. 18, no. 4 (2014) 577-601, 10.1109/TEVC.2013.2281535.

[76] A. J. Nebro, J. J. Durillo, F. Luna, B. Dorronsoro, and E. Alba, Mocell: A cellular genetic algorithm for multiobjective optimization, International Journal of Intelligent Systems, vol. 24, no. 7 (2009) 726-746.

[77] A. J. Nebro, J. J. Durillo, F. Luna, B. Dorronsoro, and E. Alba, Design issues in a multiobjective cellular genetic algorithm, in: S. Obayashi, K. Deb, C. Poloni, T. Hiroyasu and T. Murata, eds. International Conference on Evolutionary Multi-Criterion Optimization. pp. 126-140, 2007 2007.

[78] E. Zitzler, M. Laumanns, and L. Thiele, SPEA2: Improving the strength Pareto evolutionary algorithm, in: Evolutionary Methods for Design, Optimization and Control with Applications to Industrial Problems (EUROGEN 2001), Athens. Greece, *International Center for Numerical Methods in Engineering*, 2001.

[79] A. J. Nebro, F. Luna, E. Alba, B. Dorronsoro, J. J. Durillo, and A. Beham, AbYSS: Adapting scatter search to multiobjective optimization, IEEE Transactions on Evolutionary Computation, vol. 12, no. 4 (2008) 439-457.

[80] S. Kukkonen, and J. Lampinen, GDE3: The third evolution step of generalized differential evolution, in. pp. 443-450, 2005.

[81] A. J. Nebro, J. J. Durillo, J. Garcia-Nieto, C. A. C. Coello, F. Luna, and E. Alba, SMPSO: A new pso-based metaheuristic for multi-objective optimization, in: 2009 IEEE Symposium on Computational Intelligence in Multi-Criteria Decision-Making (MCDM), Nashville, TN, USA. pp. 66-73, 2009 2009.

[82] J. R. Koza, "Genetic programming II: Automatic discovery of reusable subprograms," Cambridge, MA, USA, 1994.

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

Part II
Papers

Paper A

Understanding Uncertainty in Cyber-Physical Systems: A Conceptual Model

Man Zhang, Bran Selic, Shaukat Ali, Tao Yue, Oscar Okariz and Roland. Norgren

In: Proceedings of the 12th European Conference on Modelling Foundations and Applications (ECMFA 2016), pp. 247-264, 2016.

DOI: 10.1007/978-3-319-42061-5_16

Abstract

Uncertainty is intrinsic in most technical systems, including Cyber-Physical Systems (CPS). Therefore, handling uncertainty in a graceful manner during the real operation of CPS is critical. Since designing, developing, and testing modern and highly sophisticated CPS is an expanding field, a step towards dealing with uncertainty is to identify, define, and classify uncertainties at various levels of CPS. This will help develop a systematic and comprehensive understanding of uncertainty. To that end, we propose a conceptual model for uncertainty specifically designed for CPS. Since the study of uncertainty in CPS development and testing is still irrelatively unexplored, this conceptual model was derived in a large part by reviewing existing work on uncertainty in other fields, including philosophy, physics, statistics, and healthcare. The conceptual model is mapped to the three logical levels of CPS: Application, Infrastructure, and Integration. It is captured using UML class diagrams, including relevant OCL constraints. To validate the conceptual model, we identified, classified, and specified uncertainties in two distinct industrial case studies.

Keywords. Uncertainty; Cyber-Physical Systems; Conceptual Model.

1 Introduction

Cyber-Physical Systems (CPS) are present in a variety of safety/mission critical domains [1-3]. Given the pervasiveness of CPS and their criticality to the daily functioning of society, it is vital for such systems to operate in a reliable manner. However, since they generally function in an inherently complex and unpredictable physical environment, a major difficulty with these systems is that they must be designed and operated in the presence of uncertainty. By *uncertainty* we mean here the lack of certainty (i.e., knowledge) about the timing and nature of inputs, the state of a system, a future outcome, as well as other relevant factors.

As a first crucial step in such an investigation, we feel that it is necessary to understand the phenomenon of uncertainty and all its relevant manifestations. This means to systematically identify, classify and specify uncertainties that might arise at any of the three levels of CPS: *Application*, *Infrastructure*, and *Integration*. Based on studying and analyzing existing uncertainty models developed in other fields, including philosophy, physics,

statistics and healthcare [4-7], we have defined an uncertainty conceptual model for CPS (*U-Model*) with the following objectives: 1) provide a unified and comprehensive description of uncertainties to both researchers and practitioners, 2) classify uncertainties with the aim of identifying common representational patterns when modeling uncertain behaviors, 3) provide a reference model for systematically collecting uncertainty requirements, 4) serve as a methodological baseline for modeling uncertain behaviors in CPS, and, last but not least, 5) provide a basis for standardization of the conceptual model leading to its broader application in practice.

To verify the completeness and validity of the *U-Model*, we validated it using uncertainty requirements[4] collected from two industrial case studies from two different domains: 1) Automated Warehouses developed by ULMA Handling Systems (www.ulmahandling.com/en/), Spain, 2) GeoSports (fpx.se/geo-sports/) developed by Future Position X, Sweden. This empirical validation was systematically performed in several stages and, as a result, several revisions of the *U-Model* were obtained in addition to a refined set of uncertainty requirements. The version of the *U-Model* that emerged from this work is presented in this paper. Based on the results of this validation, we discovered 61.5% (averaged across the two case studies) additional uncertainties not identified in the initial specifications. The rest of this paper is organized as follows: Section 2 presents the background and a running example. Section 3 presents the *U-Model*. Section 4 presents evaluation and discussion. Section 5 discusses related work and we conclude the paper in Section 6.

2 Background and Running Example

A CPS is defined in [8] 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. A-1. As defined in [8], uncertainty can occur at the following three levels (Fig. A-1): 1) *Application level:* Due to events/data originating from the application of the CPS; 2) *Infrastructure level*: Due to

[4] Use cases containing scenarios having uncertainty.

interactions including events/data among physical units, networking infrastructure, and/or cloud infrastructure, 3) *Integration level:* Due to either interaction among uncertainties at the first two levels or due to interactions between application and infrastructure levels.

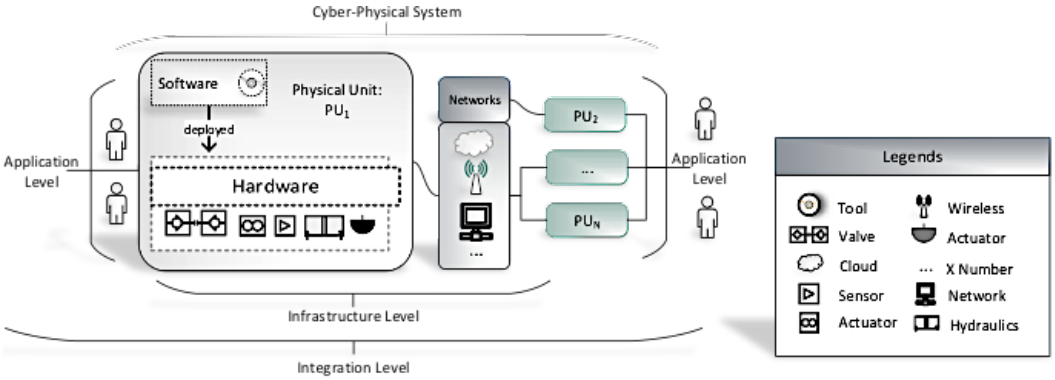


Fig. A-1. Conceptual model of a Cyber-Physical System [8]

Due to confidentiality constraints, the actual industrial CPS case studies that we used to evaluate the *U-Model* (Section 4) cannot be described in detail. Instead, we chose a Videoconferencing Systems (VCS) developed by Cisco, Norway, as an example to illustrate the conceptual model that has been used in our previous projects.

A typical VCS sends and receives audio/video streams to other VCS in a videoconference including dedicated hardware-based VCS, software-based VCS for PCs, and cloud-based VCS solutions (e.g., WebEx) as shown in Fig. A-2 (inspired from [9] and our existing collaboration with Cisco). To support videoconferences a complex infrastructure is provided by Cisco (Fig. A-2) comprising of a variety of hardware such as gateways (e.g., Expressway) and dedicated servers (e.g., Telepresence and unified Call Management servers). In Fig. A-2, we also show the various levels at which the uncertainties can occur in the context of our running example. For example, as shown in Fig. A-2, at Site 2, the interactions of *Application level* uncertainties in VCS 2 and uncertainties in the Telepresence Servers are shown as *Integration level* uncertainties.

To facilitate the understanding of concepts, a VCS represents aspects of the physical world in a somewhat simplified form. Among other functions, the VCS controls the movement of a set of cameras that are directly attached to it via wired/wireless media. This can also be performed via a cloud-based VCS application (i.e., WebEx) in addition to dedicated hardware-based solutions. In the course of a videoconference, a number of

different uncertainties exist due to the complex and heterogeneous collection of networks, cloud-based infrastructures, and VCSs.

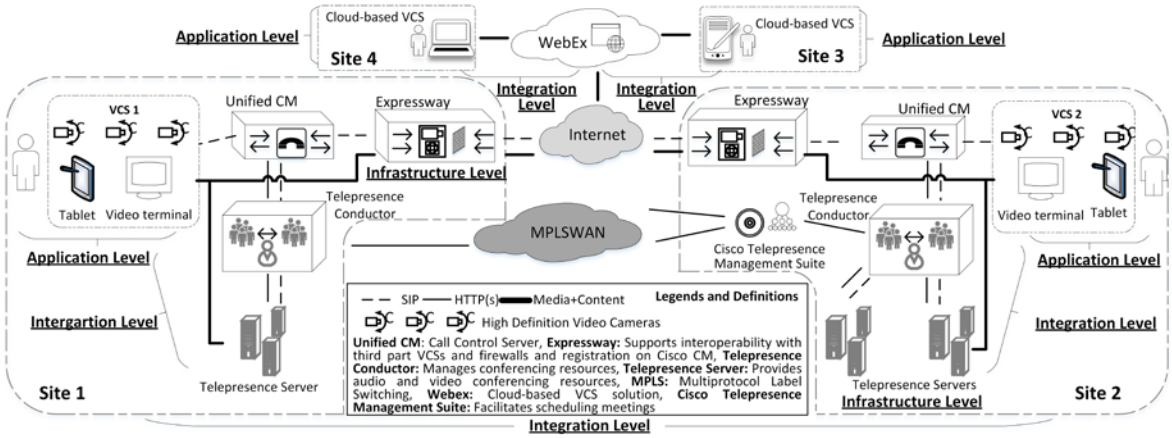


Fig. A-2. Running Example – Videoconferencing System (VCS)

3 Uncertainty Conceptual Model

The *U-Model* includes *Belief Model*, *Uncertainty Model* and *Measure Model*. Their key details are presented below, whereas more details are presented in [10].

3.1 Belief Model

The *U-Model* takes a subjective approach to representing *uncertainty*. This means that uncertainty is modeled as 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. Since it lacks perfect knowledge, a BeliefAgent possesses a set of subjective Beliefs about the subject. These may be *valid*, if the beliefs accurately represent facts, or *invalid*, if they do not[5]. A Belief is an abstract concept, but can be expressed in concrete form via one or more explicit BeliefStatements. Different BeliefAgents may hold different views about a given subject, which is why each BeliefStatement is associated with a particular BeliefAgent. Note that a BeliefAgent does not necessarily represent a human individual; it could constitute a

[5] Such a strictly binary categorization may not be always realistic, since *Belief*s could be characterized by degrees of validity. However, in this model, we choose to ignore such subtleties. Specifically, a BeliefStatement is deemed to be valid if it is a sufficient approximation of the truth for the purpose on hand.

community of individuals, some non-human organism, or even some technological system, such as a computer system[6].

These and other core concepts of the *U-Model* are represented as a class diagram in Fig. A-3, where *subjective* concepts are represented by the grey-filled boxes and *objective* concepts as the unfilled boxes in Fig. A-3. Subjective concepts are manifestations of the imperfect knowledge of a BeliefAgent. Conversely, objective concepts reflect objective reality and are, therefore, independent of BeliefAgents and their imperfections. One significant characteristic of the subjective concepts is that they can vary over time, as might occur, e.g., when more information becomes available[7].

Uncertainty (lack of confidence) represents *a state of affairs whereby a* BeliefAgent *does not have full confidence in a* Belief *that it holds*. This may be due to various factors: lack of information, inherent variability in the subject matter, ignorance, or even due to physical phenomena, e.g., the Heisenberg uncertainty principle. While Uncertainty is an abstract concept, it can be represented by a corresponding Measurement expressing in some concrete form the *subjective degree* of uncertainty held by the agent to a BeliefStatement. Since the latter is a subjective notion, a Measurement should not be confused with the degree of validity of a BeliefStatement. Instead, it indicates the level of confidence that the agent has in a statement[8].

Finally, note that this model is intentionally made very general, which allows it to be extended and customized for a variety of purposes, e.g., uncertainty model-based testing of CPS in the context of our project. Fig. A-3 does not show the complete model, e.g., to reduce visual clutter, some of the OCL constraints have been removed. The complete model is described in [10]. In the remainder of this section, we examine key concepts of the core model in more detail and illustrate some of them using the running VCS example (see Table A-1).

[6] In this case, the Beliefs would be reflected in the rules that are programmed into the system.
[7] However, more information does not necessarily imply a decrease in uncertainty.
[8] E.g, many people in the past were absolutely certain that the Earth was flat.

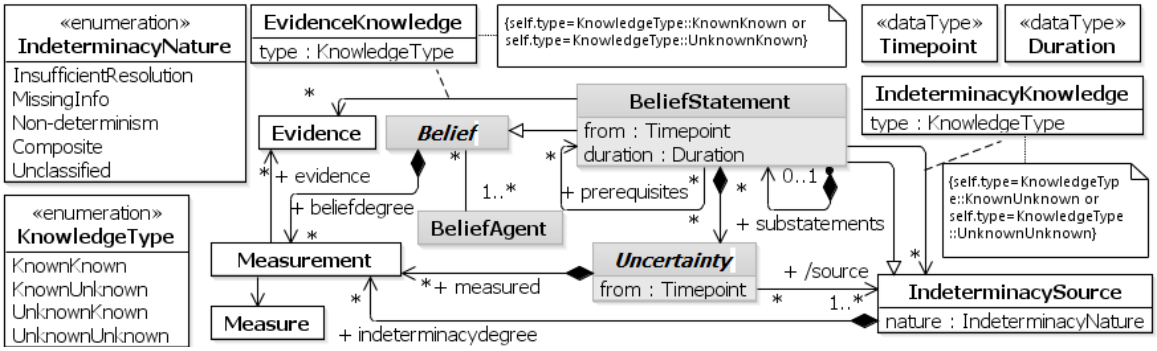


Fig. A-3. The Core Belief Model

3.1.1 Belief, BeliefAgent and BeliefStatement

A Belief is an *implicit* subjective explanation or description of some phenomena or notions [9] held by a BeliefAgent. This is an abstract concept whose only concrete manifestation is as a BeliefStatement. In our running example, a test engineer at Cisco may have his/her own Beliefs about how a VCS works. When coding test cases, he/she concretizes his/her Beliefs as executable test scripts that may or may not correspond to the actual implementation the VCS. A BeliefStatement in this context could be manifested as one executable test case file and in other contexts it may correspond to other artifacts, e.g., source code.

A BeliefAgent is a physical entity [10] owning one or more Beliefs about phenomena/notion. A BeliefAgent can take actions based on its Beliefs. In our example of CPS testing, BeliefAgents include: 1) *Application level*: software test engineers focusing on testing new versions of the VCS software, and 2) *Infrastructure level*: Network engineers focusing on testing a VCS under diverse network situations.

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. A BeliefStatement can be an aggregate of two or more component BeliefStatements, or it may require one or more prerequisite BeliefStatements.

[9] "Phenomena" here is intended to cover aspects of objective reality, whereas "notion" covers abstract concepts, such those encountered in mathematics or philosophy.

[10] We exclude here from this definition "virtual" BeliefAgents, such as those that might occur in virtual reality systems and computer games.

The concrete form of a BeliefStatement 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.

Due to the complex nature of objective reality and our human and technical limitations, 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 at a given point of time. 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 more interested in analyzing uncertainties in a BeliefStatement rather than studying its validity.

In our example, we define the following BeliefStatements: 1) *Application level*: The VCS will successfully connect to another VCS 70% of the time (see Table A-1); 2) *Infrastructure level*: The Expressway gateway is successful 99% of the time in connecting a Cisco VCS with a third party VCS (see Table A-1); and 3) *Integration level*: A VCS communicates with the Expressway gateway with a 90%-95% success rate.

Table A-1. Running Example – *Dial* of VCS

Package	**Concept**	**Explanation**
Belief Model	Level	*Application*
	BeliefAgent	Software testing engineers
	BeliefStatement	The VCS successfully dials to another VCS 70% of the time.
	Indeterminacy Source	Improper human behavior where he/she enters an incomplete name/number of VCS to dial `IndeterminacyNature:: Non-determinism`, and `IndeterminacyKnowledge.type= KnowledgeType::KnownUnknown`
	Evidence	Execution of 100 test cases on the VCS in the past week involving the dial command `EvidenceKnowledge.type =KnowledgeType::KnownKnown`
	Uncertainty	Uncertainty in whether the dial to another VCS will be successful or not. This concept may depend on (see self-association of Uncertainty in Fig. A-4) another uncertainty composed by another BeliefStatement specified by the network engineer, e.g. "The Expressway gateway is 99% of the time successful in connecting Cisco's VCS with third party VCS."
Uncertainty Model	Type	Occurrence
	Lifetime	Difference of time that the dial was initiated and response from the system was received
	Locality	Invocation of the dial API of VCS
	Pattern	Derived pattern from the collection of values of lifetime of the uncertainty
	Risk	Low or even can be ignored
	Measurement	70% of the time, derived from Evidence based on test execution history

Measure Model	Measure	Probability

3.1.2 Evidence, EvidenceKnowledge, IndeterminacySource and IndeterminacyKnowledge.

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. Evidence is inherently an objective phenomenon, representing *something that actually happened*. This means that we exclude here the possibility of counterfeit or invented evidence. Nevertheless, although Evidence represents objective reality, it needs not be conclusive in the sense that it removes all doubt (Uncertainty) about a BeliefStatement. In our example of an *Application level* BeliefStatement, i.e., "The VCS successfully dials to another VCS 70% of the time". The Evidence of the 70% of success rate of dial may be obtained from the execution of 100 test cases on the VCS in the past week (see Evidence Table A-1).

EvidenceKnowledge expresses an objective relationship between a BeliefStatement and relevant Evidence. It 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). This is formally expressed by the two constraints attached to EvidenceKnowledge (Fig. A-3). An example is provided in Table A-1.

Indeterminacy is a situation whereby the full knowledge necessary to determine the required factual state of some phenomena/notions is unavailable[11]. This is an abstract concept whose only concrete manifestation is in the form of an IndeterminacySource. As noted earlier, this may be due either to subjective reasons (e.g., agent ignorance) or to objective reasons (e.g., the Heisenberg uncertainty). It is also useful to explicitly identify factors that lead to Uncertainty referred to as IndeterminacySources. This 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.

[11] Care should be taken to distinguish between indeterminacy and non-determinism. The latter is only one possible source of indeterminacy.

One possible source of indeterminacy can be another BeliefStatement, which is why the latter is a specialization of IndeterminacySource (Fig. A-3). For example, for the following BeliefStatement: "The VCS successfully dials to another VCS 70% of the time", for which there might be several IndeterminacySources. A possibility is incorrect operator behavior, where an incomplete name of the target VCS specified (IndeterminacySource entry in Table A-1).

IndeterminacyNature represents the specific kind of indeterminacy and can be one of the following: 1) InsufficientResolution – The information available about the phenomenon in question is not sufficiently precise; 2) MissingInfo – The full set of information about the phenomenon in question is unavailable at the time when the statement is made; 3) Non-determinism – The phenomenon in question is either practically or inherently non-deterministic; 4) Composite – A combination of more than one kinds of indeterminacy; 5) Unclassified – Indeterminate indeterminacy.

IndeterminacyKnowledge expresses an objective relationship between an IndeterminacySource and the awareness that the BeliefAgent has of that source. 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 (represented as enumeration) has four values: 1) KnownKnown indicates that an associated BeliefAgent is consciously aware of some relevant aspect; 2) KnownUnknown (*Conscious Ignorance*) indicates that an associated BeliefAgent understands that it is ignorant of some aspect; 3) UnknownKnown (*Tacit Knowledge*) indicates that an associated BeliefAgent is not explicitly aware of some relevant aspect, but may be able to exploit in some way; 4) UnknownUnknown (*Meta Ignorance*) indicates that an associated BeliefAgent is unaware of some relevant aspect.

At a given point in time, a BeliefAgent always makes a statement based on a KnownKnown Evidence and a KnownUnknown IndeterminacySource. Splitting EvidenceKnowledge and IndeterminacyKnowledge provides the flexibility to enable transitions among different knowledge types (e.g., from UnknownKnown to

KnownKnown), based on the evolution of EvidenceKnowledge and IndeterminacyKnowledge related to the associated BeliefAgent. For the following BeliefStatement: “The VCS successfully dials to another VCS 70% of the time” and an IndeterminacySource is improper operator behavior, the KnowledgeType of IndeterminacyKnowledge is KnownUnknown.

3.1.3 Measurement and Measure.

Measurement when associated with a given IndeterminacySource represents the optional quantification (or qualification) that specifies the degree of indeterminacy of the IndeterminacySource. For example, in the case of a Non-determinism IndeterminacySource, its measurement could be expressed by a probability or a probability density function. For the example presented in Table A-1, ‘70%’ is the measurement of the IndeterminacySource improper operator behavior.

Measurement when associated with Uncertainty is a subjective concept representing the actual measured value of an uncertainty defined by a BeliefAgent. It may be possible to specify a Measurement that quantifies in some way (e.g., as a probability) the degree of the uncertainty that a BeliefAgent associates with a BeliefStatement. Measurement when associated with Belief represents sets of measured values of all the uncertainties contained by a BeliefStatement defined by a BeliefAgent. Several constraints on Measurement ensure that each Measurement owned by either Belief, Uncertainty or IndeterminacySource has a unique Measure. Currently, we modeled three different measures, i.e., Probability, Ambiguity and Vagueness that are discussed in the *Measure Model* (Section 3.3). In the future, we will provide UML model libraries for Measurement when implementing *U-Model* as a UML profile. Measure is an objective concept specifying method of measuring uncertainty. More details are presented in Section 3.3.

3.2 Uncertainty Model

This model (Fig. A-4) was inspired by concepts defined in the literature on uncertainty [11-15] and is an adjunct to the *Core Belief Model* (Section 3.1). 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. In the rest of the section, we discuss each subtype of Uncertainty and its associated concepts.

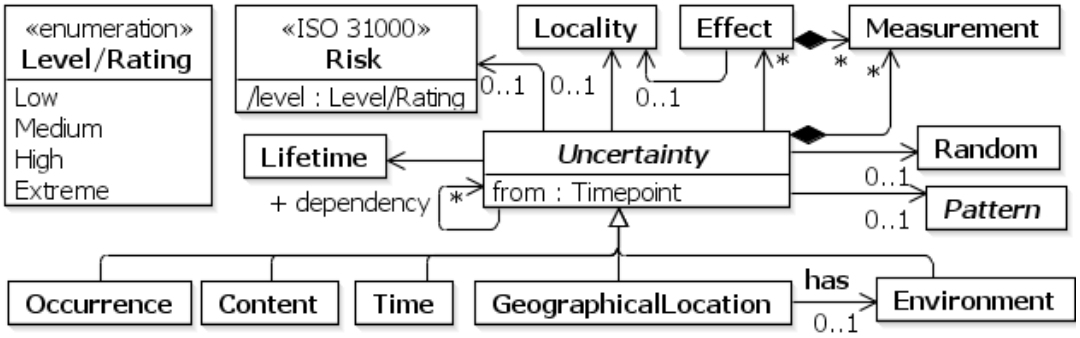


Fig. A-4. The Core Uncertainty Model

3.2.1 Uncertainty, Lifetime and Pattern.

Uncertainty represents a situation whereby a BeliefAgent lacks confidence in a BeliefStatement. Fig. A-4 shows a conceptual model for different types of Uncertainty inspired from the concepts reported in [12, 14, 15]. Uncertainty is specialized into the following types: 1) Content – represents a situation, whereby a BeliefAgent lacks confidence in content existing in a BeliefStatement; 2) Environment – represents a situation whereby a BeliefAgent lacks confidence in the surroundings of a physical system existing in a BeliefStatement; 3) GeographicalLocation –represents a situation whereby a BeliefAgent lacks confidence in geographical location existing in a BeliefStatement; 4) Occurrence – represents a situation whereby a BeliefAgent lacks confidence in the occurrence of events existing in a BeliefStatement; 5) Time –represents a situation whereby a BeliefAgent lacks confidence in time existing in a BeliefStatement. For example, for the BeliefStatement: "The VCS successfully calls another VCS 70% of the time", the Uncertainty is whether the dialing to another VCS will be successful or not and classified as Occurrence uncertainty. In case of the BeliefStatement: "The Expressway gateway is

successful 99% of the time in connecting a Cisco VCS with a third party VCS", the Uncertainty is in the connection of the gateway with the third party VCS, and type of uncertainty is again Occurrence (see type of Uncertainty in Table A-1).

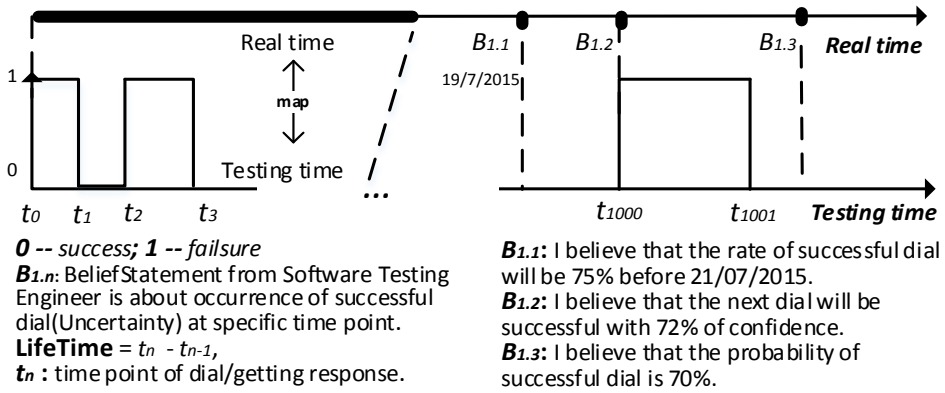


Fig. A-5. Example of Lifetime and Pattern of Uncertainty

Lifetime represents an interval of time, during which an Uncertainty exists. That is, an Uncertainty may appear temporarily and then disappear. On the other hand, an Uncertainty could be persistent, i.e., it remains until appropriate actions are taken to resolve it. An example of Lifetime is shown in Table A-1. We show two types of time in Fig. A-5: 1) *Real Time* showing the actual passing of the time, 2) *Testing Time*, i.e., a time point in real time, where a testing activity was performed, e.g., a call attempt to establish a videoconference (stimulus to the system under test) or a response from the system was received about success or failure of the call (test result). Time points t_n are shown on *Testing Time* in Fig. A-5. A BeliefStatement can be made at any point in the real time, for example, three versions of BeliefStatement B_1 ($B_{1.1}$, $B_{1.2}$, and $B_{1.3}$) can be made at different points of time as shown in Fig. A-5. Lifetime of Uncertainty (the occurrence of successful dial) in BeliefStatement B_1 should be $t_n - t_{n-1}$: difference of time that the dial was initiated and response from the system was received for $B_{1.3}$.

Fig. A-6 shows a conceptual model for the *occurrence Pattern of Uncertainty* inspired from concepts reported in [14, 16, 17]. Notice that in this section, patterns presented are by no means the representation of a complete set of patterns that may exist for an Uncertainty. Rather, we only present the most common patterns.

Periodic uncertainty occurs at regular intervals of time, whereas Persistent uncertainty is the one that lasts forever. The definition of "forever" varies; e.g, an uncertainty may exist

permanently until appropriate actions are taken. On the other hand, an uncertainty may not be resolvable and remains forever. Both Periodic and Persistent inherit from Systematic, which means that these types of patterns occur in some methodical manners, i.e., a pattern that can be described in a mathematical way.

An uncertainty with an Aperiodic pattern occurs at irregular intervals of time, which is further specialized into Sporadic and Transient. A Sporadic uncertainty occurs occasionally, whereas a Transient uncertainty occurs temporarily. Systematic and Aperiodic uncertainty patterns inherit from Temporal, which means that they both inherently have the notion of time. If an uncertainty occurs without a definite method, purpose or conscious decision, the type of the pattern it follows is referred to as Random. For example, when looking at Fig. A-5, a pattern of the Uncertainty (the occurrence of a successful call attempt) can be derived after collecting values of Lifetime of the Uncertainty (see Pattern in Table A-1).

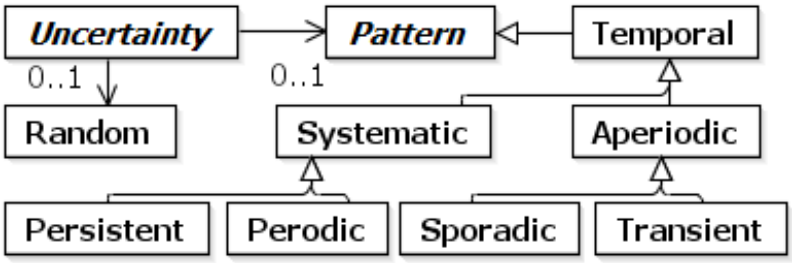


Fig. A-6. The Patterns of Uncertainty

3.2.2 Locality and Risk.

Locality (see Fig. A-4) is a particular place or a position where an Uncertainty occurs in a BeliefStatement. For example, for the BeliefStatement: “The VCS successfully dials to another VCS 70% of the time”, the Locality of the Uncertainty (whether the call attempt to another VCS will be successful or not) is in the invocation (position) of dial API of VCS (see Locality in Table A-1).

An uncertainty may have an associated Risk and high-risk uncertainties deserve special attention. As shown in Fig. A-4, an Uncertainty might or might not associated to Risk, whose level can be classified into four levels according to the ISO 31000 – Risk Management standard [18]. Level/Rating is derived from Measurement owned by Uncertainty (e.g., Probability of the Occurrence of an Uncertainty) and Measurement owned by Effect (e.g.,

high impact using the risk matrix in [19] or any other matrix). For example, for the BeliefStatement: “The VCS successfully calls another VCS 70% of the time”, the Risk associated with the Uncertainty in this BeliefStatement is low or the risk could be even ignored (see Risk in Table A-1).

3.3 Measure Model

Fig. A-7 shows the *Measure Model* of the *U-Model*, inspired from concepts reported in [12-14] and by no means complete. Depending on the type of Uncertainty, a variety of measures could be applied and new ones can also be proposed when needed. We aim to give a high-level introduction to commonly known measures.

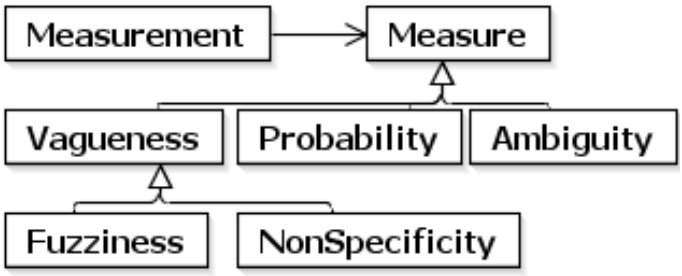


Fig. A-7. Measure Model

An uncertainty may be described ambiguously (Ambiguity). For example, in statement “The camera is down”, the ambiguity is in the measurement, i.e., the camera is either facing down or disconnected. Interested readers may consult [20] for various measures of Ambiguity. Another common way of measuring Uncertainty is in a vague manner (i.e., Vagueness), which can be further classified into Fuzziness and NonSpecificity. Regarding Fuzziness, an uncertainty may be measured using fuzzy methods. More details can be referred to the fuzzy logic literature such as [20]. 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 NonSpecificity methods. Finally, a common way of measuring uncertainty is via Probability. For example, for the BeliefStatement: “The VCS successfully calls another VCS 70% of the time”, the Uncertainty is measured by Probability (see Measure in Table A-1).

4 Evaluation

This section presents the results of the industrial case studies that we conducted to evaluate the *U-Model* and collect uncertainty requirements. First case study is about Automated Warehouse (AW) provided by ULMA Handling Systems and the second case study is about Geo Sports (GS) by Future Position X (further details in [10]).

4.1 Development and Validation of Uncertainty Requirements and U-Model

We collected uncertainty requirements from the two industrial case studies in the following ways. The uncertainty requirements were collected as part of an EU project on testing CPS under uncertainty (www.u-test.eu). An initial set of uncertainty requirements were collected by the industrial partners themselves and were later classified into the three CPS levels: *Application*, *Infrastructure*, and *Integration*. Later on, the researchers of Simula Research Laboratory conducted one workshop per partner to further refine the requirements. For AW, the onsite workshop took around three days, whereas in case of GS, a one-day onsite workshop was organized.

The validation procedure is summarized in Fig. A-8 and comprises two parallel validation processes. The first validation process is related to the validation of the *U-Model* and was mainly conducted by the researchers. The second validation process focuses on the validation of uncertainty requirements and was mainly performed by the industrial partners.

The validation was developed incrementally (Activities A1 and A2 in Fig. A-8), based on existing models in the literature and other related published works (see Section 2 for details). The Simula team validated the conceptual model using two types of examples shown as inputs to A2 in Fig. A-8: 1) Examples of uncertainties from domains other than CPS, and 2) A subset of VCS requirements. As a result an initial version of the *U-Model* was produced referred as *U-Model* V.1 in Fig. A-8.

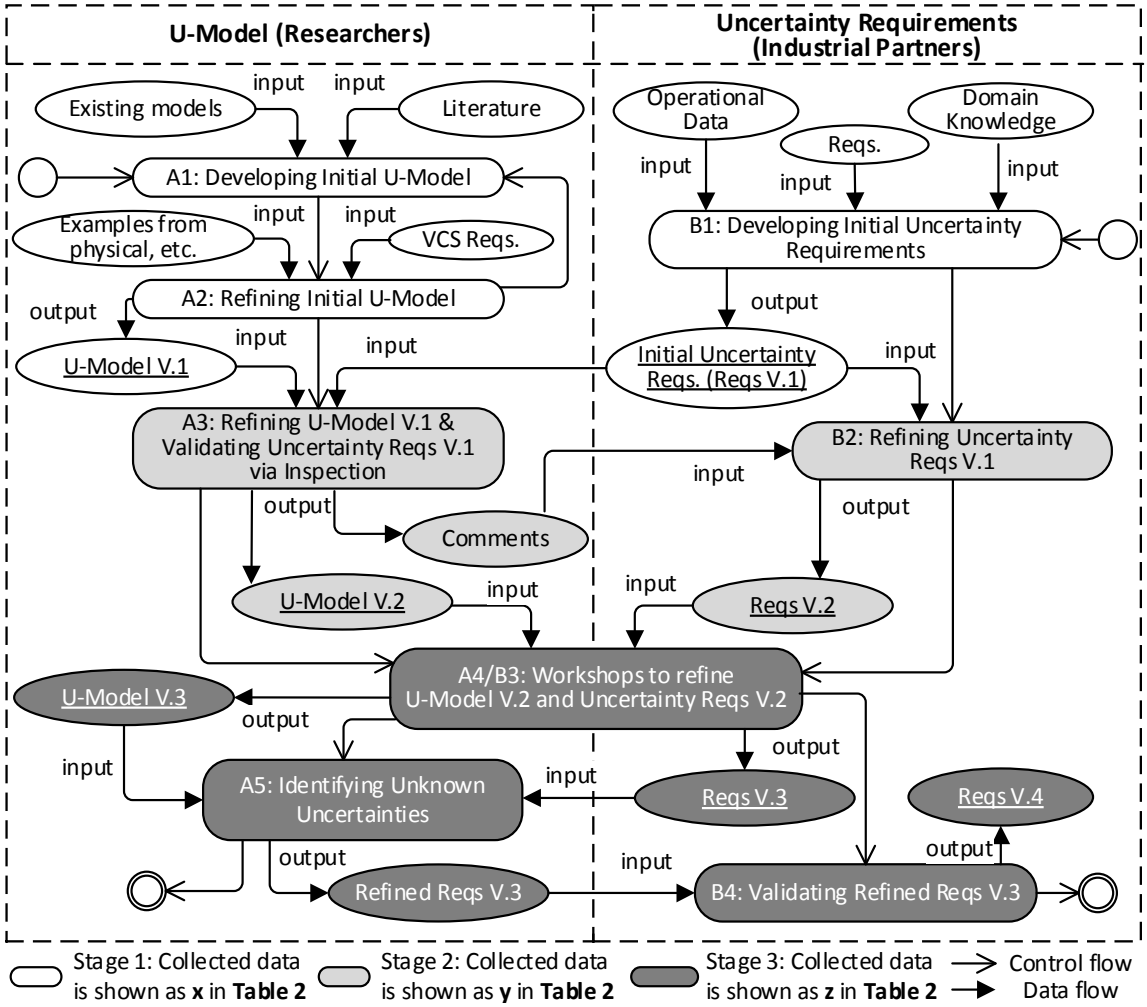


Fig. A-8. Development and Validation of Uncertainty Requirement and U-Model

In parallel, initial uncertainty requirements (Reqs V.1) were provided (Activity B1 in Fig. A-8) by the industrial partners based on their domain knowledge, existing requirements of their CPS, and some information from the real operation of the CPS. These initial uncertainty requirements were used as input for A3, focusing on further refining the *U-Model*. In addition, the researchers inspected the collected uncertainty requirements using a requirements inspection checklist provided in [21] and provided a set of comments for the industrial partners on how to improve their requirements. There were two key outputs of the A3 activity: *U-Model* V.2 and comments to refine the requirements. These comments were used by the industrial partners to produce a second version of requirements (Reqs V.2) in B2.

4.2 Evaluation Results

For each of the industrial case studies, we mapped the three versions of uncertainty requirements (Reqs V.1, Reqs V.2, and Reqs V.4) to the three versions of *U-Model* (V.1 to V.3). The number of the instances of the concepts are shown in columns x (for mapping Reqs V.1 to *U-Model* V.1), y (for mapping Reqs. V.2 to *U-Model* V.2), and z (for mapping Reqs V.4 to *U-Model* V.3) of Table A-2, respectively. Notice that Reqs V.3 was the result of the onsite workshops together with *U-Model* V.3 and thus these requirements are not mapped to the model since both the conceptual model and requirements were refined together. We analyzed in total 20 use cases for *AW* and 18 use cases for *GS*. Notice that, the number of use cases for each case study did not change during the requirements collection and the *U-Model* validation process. They were selected at the beginning of the process to capture and specify the key functionalities of the CPS.

Based on the final version of requirements, we can see from Table A-2 that most common types of identified uncertainties are Content uncertainties having 91 instances (the last column in Table A-2) and Occurrence uncertainties having 205 instances. On the other hand, a relatively lower number of Time uncertainties (50), Environment uncertainties (32), and GeographicalLocation uncertainties (31) were found in the case studies. Most of the time, uncertainties are due to InsufficientResolution (42 instances), MissingInfo (31 instances) or Non-determinism (89 instances). In terms of Measure, our analysis revealed that 76 of the uncertainties across the case studies may be measured with the Fuzziness measures, 119 with NonSpecificity, whereas 148 with Probability. Notice that in Table A-2, we do not show the concepts that have no instances identified from any of the case studies.

In Table A-2, the *R1 = y/x -1* column represents the increased percentage of mapping of concepts explicitly captured in Reqs V.2 as compared to Reqs V.1. The *R2* = z/y -1 column shows the increased percentage of mapping of concepts explicitly captured in Reqs V.4, i.e., including unknown uncertainties that weren't explicitly specified in Reqs V.2. As can be seen from Table A-2, in case of *AW* for *R1*, on average, we identified an additional 1.43 of uncertainties and in *R2* we identified an additional 0.51 of uncertainties. For *GS*, these percentages are 2.39 in *R1*, and 0.72 in *R2*, respectively. In total, in *R1* on average we identified additional 1.91 of uncertainties, whereas in R2 we identified on average 0.615 of unknown uncertainties.

In Table A-2, one can see that we didn't have exact data (e.g., probability) and risk information available at the moment. Such data will be collected using questionnaire-based surveys in the future to quantify the identified uncertainties. In addition, we didn't observe any pattern for the occurrences of the identified uncertainties. Moreover, the Belief part of the conceptual model (e.g., concepts Belief, BeliefAgent) was derived to understand Uncertainty and is not relevant for the validation.

Table A-2. Evaluation Results of Uncertainty Requirements and *U-Model*

Concept		**AW**					**GS**					**Freq**
		x	*y*	*z*	*R1**	*R2**	*x*	*y*	*z*	*R1*	*R2*	*Total*[+]
Uncertainty	*Content*	14	36	55	1.57	0.53	16	20	36	0.25	0.80	91
	Time	6	16	28	1.67	0.75	5	11	22	1.20	1.00	50
	Occurrence	27	81	126	2.00	0.56	6	50	79	7.33	0.58	205
	Environment	13	15	22	0.15	0.47	4	6	10	0.50	0.67	32
	Geographical Location	4	11	14	1.75	0.27	3	11	17	2.67	0.55	31
Sum for x, y, z / Average for R1, R2		**64**	**159**	**245**	**1.43**	**0.51**	**34**	**98**	**164**	**2.39**	**0.72**	**409**
Indeterminacy	*Insufficient Resolution*	7	18	24	1.57	0.33	11	14	18	0.27	0.29	42
	Non-determinism	7	45	52	5.43	0.16	11	20	37	0.82	0.85	89
	MissingInfo	2	19	24	8.50	0.26	0	5	7	N/A	0.40	31
Sum for x, y, z / Average for R1, R2		**16**	**82**	**100**	**2.67**	**0.43**	**22**	**39**	**62**	**0.55**	**0.57**	**162**
Measure	*Fuzziness*	6	22	51	2.67	1.32	6	15	25	1.50	0.67	76
	NonSpecificity	16	40	73	1.50	0.83	12	26	46	1.17	0.77	119
	Probability	18	56	98	2.11	0.75	4	37	50	8.25	0.35	148
Sum for x, y, z / Average for R1, R2		**40**	**118**	**222**	**2.09**	**0.96**	**22**	**78**	**121**	**3.64**	**0.60**	**343**

[*]R1 = y/x – 1 [*]R2 = z/y – 1 [+]Total = AW(z)+GS(z) Freq is Frequency

5 Related Work

Uncertainty is a term that has been used in various fields such as philosophy, physics, statistics and engineering to describe a state of having limited knowledge where it is impossible to exactly tell the existing state, a future outcome or more than one possible outcome [18]. Various uncertainty models have been proposed in the literature from different perspectives for various domains. For instance, from an ethics perspective, uncertainties are classified as objective uncertainty and subjective uncertainty, both of which are further classified into subcategories to support decision-making [4]. In healthcare, uncertainty has often been defined as "the inability to determine the meaning of illness-related events" [5]

and comprehensive domain-specific uncertainty models (e.g., [6]) have been proposed, as discussed in [7].

Uncertainty is receiving more and more attention in recent years in both system and software engineering, especially for CPS, which are required to be more and more context aware [22-24]. Moreover, CPS inherently involves tight interactions between various engineering disciplines, information technology, and computer science. This magnifies uncertainties. Therefore, adequate treatment of uncertainty becomes increasingly more relevant for any non-trivial CPS. However, to the best of our knowledge, there is no comprehensive uncertainty conceptual model existing in literature that focused specifically on CPS design or on system/software engineering in general. In the remainder of the section, we discuss how the concepts uncovered during the literature review align with our proposed conceptual model.

The *U-Model* concepts BeliefAgent, BeliefStatement, and Belief of the Belief model were adapted from [12]. The author of [12] postulates that uncertainty involves a statement whose truth is expected by a person, and therefore the truth might differ for different persons (defined as BeliefAgent in our model). However, as we discussed in Section 3.1, we assigned a broader meaning to BeliefAgent: which can be an individual, a community of individuals, or a technology. The *U-Model* concepts Environment and Locality were adapted from [12, 25-27], and we related them to the other *U-Model* concepts.

Our knowledge conceptual model aligns well with the model of knowledge reported in [28]. Here the authors looked at how to manage different types of known and unknown knowledge to distinguish what is known from what is not known. Knowledge is also classified from a different perspective: something that everyone knows, tacit knowledge, conscious ignorance and meta-ignorance. Their objective is to better understand ignorance. The author of [29] also studied unknowns and provided a taxonomy particularly focusing on ignorance (named as KnownUnknown and UnknownUnknown in our conceptual model). In our conceptual model, we further elaborate these concepts and captured them as KnowledgeType, which is associated to Evidence and IndeterminacySource via EvidenceKnowledge and IndeterminacyKnowledge.

We classified uncertainties into various types including Content, Time and Occurrence. In [12], a chapter was dedicated to the discussion of content uncertainty and its measurement.

The other two types of uncertainties were mentioned in [12, 14, 15], with examples but with no clear definitions provided. We adopted the measurements in our conceptual model. Different types of sources of uncertainty for various purposes have been identified in the literature. In [30], the authors captured sources of uncertainty by considering risk and reliability analyses, based on which they classified uncertainty. The authors of [15, 31] identified sources of uncertainty in active systems. In [23, 32], the authors described the sources of uncertainty in software engineering in general. We however proposed the *U-Model* concepts IndeterminacySource and IndeterminacyNature to capture sources of uncertainty.

Aleatory and *Epistemic* uncertainties are the two generic categories of uncertainties discussed in many works [30, 33]. According to the work reported in [30], Aleatory is due to the inherent randomness of phenomena, whereas the Epistemic uncertainty is mainly due to the lack of knowledge. These two types are also covered in the *U-Model*. For example, the Non-determinism (nature of indeterminacy in *U-Model*) represents the randomness as in Aleatory, and Epistemic is covered by MissingInfo — nature of indeterminacy.

In [34], the author noted that uncertainty can occur in a random or systematic manner. In the Pattern part of the *U-Model*, we further elaborated the "systematic" concept by introducing Pattern and its sub categories. In literature, uncertainty is often related to Risk. The acquisition project team of the US Air Force Electronic System Center (ESC) has proposed a risk matrix for evaluating risks [19]. They introduced the concepts of Risk, impact, likelihood of occurrence, and rate of Risk and also identified their relations. We reused these concepts and linked them with Uncertainty.

6 Conclusion

Cyber-Physical Systems (CPS) often consist of heterogeneous physical units (e.g., sensors, control modules) communicating via various networking equipment, interacting with applications and humans. Thus, uncertainty is inherent in CPS due to tight interactions between hardware, software and humans, and the need for them to be increasingly context aware. To understand uncertainty in the context of CPS, unified and comprehensive uncertainty conceptual model should be derived. The *U-Model* is such a conceptual model developed in an EU project, based on a thorough literature review of existing uncertainty

models from various domains (e.g., philosophy, healthcare), and refined and validated with two industrial CPS case studies of various domains. Based on the results of several stages validation, we obtained the current version of the conceptual model in addition to refined uncertainty requirements. On average, we managed to learn 61.5% of unknown uncertainties that weren't explicitly specified in the uncertainty requirements collected from the two case studies.

References

[1] M. Broy, Engineering Cyber-Physical Systems: Challenges and Foundations, in: Proceedings of the Third International Conference on Complex Systems Design & Management CSD&M 2012. pp. 1-13, 2013.

[2] H.-M. Huang, T. Tidwell, C. Gill, C. Lu, X. Gao, and S. Dyke, Cyber-Physical Systems for Real-Time Hybrid Structural Testing: A Case Study, in: Proceedings of the 1st ACM/IEEE International Conference on Cyber-Physical Systems. pp. 69-78, 2010.

[3] T. Tidwell, X. Gao, H.-M. Huang, C. Lu, S. Dyke, and C. Gil, Towards Configurable Real-Time Hybrid Structural Testing: A Cyber Physical Systems Approach, in: ISORC '09 Proceedings of the 2009 IEEE International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing. pp. 37-44, 2009.

[4] C. Tannert, H. D. Elvers, and B. Jandrig, The ethics of uncertainty, EMBO reports, vol. 8, no. 10 (2007) 892-896.

[5] M. H. Mishel, Uncertainty in illness, Image: The Journal of Nursing Scholarship, vol. 20, no. 4 (1988) 225-232.

[6] A. S. Babrow, C. R. Kasch, and L. A. Ford, The many meanings of uncertainty in illness: Toward a systematic accounting, Health communication, vol. 10, no. 1 (1998) 1-23.

[7] P. K. Han, W. M. Klein, and N. K. Arora, Varieties of Uncertainty in Health Care A Conceptual Taxonomy, Medical Decision Making, vol. 31, no. 6 (2011) 828-838.

[8] S. Ali, and T. Yue, 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, 2015.

[9] Cisco, Cisco Preferred Architecture for Video - Design Overview, 2015; http://www.cisco.com/.

[10] M. Zhang, B. Selic, S. Ali, T. Yue, O. Okariz, and R. Norgren, An Uncertainty Taxonomy to Support Model-Based Uncertainty Testing of Cyber-Physical Systems, Simula Laboratory Research, 2015; https://www.simula.no/publications/uncertainty-taxonomy-support-model-based-uncertainty-testing-cyber-physical-systems.

[11] G. Bammer, and M. Smithson, Uncertainty and risk: multidisciplinary perspectives, Routledge, 2012.

[12] D. V. Lindley, Understanding uncertainty (revised edition), John Wiley & Sons, 2014.

[13] K. Potter, P. Rosen, and C. R. Johnson, From quantification to visualization: A taxonomy of uncertainty visualization approaches, *Uncertainty Quantification in Scientific Computing*, pp. 226-249: Springer, 2012.

[14] B. N. Taylor, Guidelines for Evaluating and Expressing the Uncertainty of NIST Measurement Results (rev, DIANE Publishing, 2009.

[15] S. Wasserkrug, A. Gal, and O. Etzion, A taxonomy and representation of sources of uncertainty in active systems, *Next Generation Information Technologies and Systems*, pp. 174-185: Springer, 2006.

[16] A. Cimatti, A. Micheli, and M. Roveri, Timelines with Temporal Uncertainty, in: Aaai, 2013.

[17] B. Sprunt, L. Sha, and J. Lehoczky, Scheduling sporadic and aperiodic events in a hard real-time system, DTIC Document, 1989.

[18] ISO, "ISO 31000: Risk management," 2009, http://www.iso.org/iso/home/standards/iso31000.htm.

[19] P. R. Garvey, and Z. F. Lansdowne, Risk matrix: an approach for identifying, assessing, and ranking program risks, Air Force Journal of Logistics, vol. 22, no. 1 (1998) 18-21.

[20] G. Klir, Facets of systems science, Springer Science & Business Media, 2013.

[21] T. Yue, L. C. Briand, and Y. Labiche, Facilitating the transition from use case models to analysis models: Approach and experiments, ACM Transactions on Software Engineering and Methodology (TOSEM), vol. 22, no. 1 (2013) 5.

[22] R. R. Rajkumar, I. Lee, L. Sha, and J. Stankovic, Cyber-physical systems: the next computing revolution, in: Proceedings of the 47th Design Automation Conference. pp. 731-736, 2010.

[23] M. Conti, S. K. Das, C. Bisdikian, M. Kumar, L. M. Ni, A. Passarella, G. Roussos, G. Tröster, G. Tsudik, and F. Zambonelli, Looking ahead in pervasive computing: Challenges and opportunities in the era of cyber–physical convergence, Pervasive and Mobile Computing, vol. 8, no. 1 (2012) 2-21.

[24] D. Garlan, Software engineering in an uncertain world, in: Proceedings of the FSE/SDP workshop on Future of software engineering research. pp. 125-128, 2010.

[25] F. Hu, Cyber-Physical Systems: Integrated Computing and Engineering Design, CRC Press, 2013.

[26] B. H. C. Cheng, P. Sawyer, N. Bencomo, and J. Whittle, A goal-based modeling approach to develop requirements of an adaptive system with environmental uncertainty, *Model Driven Engineering Languages and Systems*, pp. 468-483 %@ 3642044247: Springer, 2009.

[27] K. Wan, K. L. Man, and D. Hughes, Specification, analyzing challenges and approaches for cyber-physical systems (CPS), Engineering Letters, vol. 18, no. 3 (2010) 308 %@ 1816-093X.

[28] A. Kerwin, None Too Solid Medical Ignorance, Science Communication, vol. 15, no. 2 (1993) 166-185.

[29] M. Smithson, Ignorance and uncertainty: Emerging paradigms, Springer-Verlag Publishing, 1989.

[30] A. Der Kiureghian, and O. Ditlevsen, Aleatory or epistemic? Does it matter?, Structural Safety, vol. 31, no. 2 (2009) 105-112 %@ 0167-4730.

[31] R. de Lemos, H. Giese, H. A. Müller, M. Shaw, J. Andersson, L. Baresi, and B. Becker, Software engineering for self-adaptive systems, in, 2009.

[32] H. Ziv, D. Richardson, and R. Klösch, The uncertainty principle in software engineering, in, 1997.

[33] H. G. Matthies, Quantifying uncertainty: modern computational representation of probability and applications, *Extreme man-made and natural hazards in dynamics of structures*, pp. 105-135 %@ 1402056540: Springer, 2007.

[34] S. Bell, A beginner's guide to uncertainty of measurement, National Physical Laboratory Teddington, Middlesex, 2001.

Paper B

Specifying Uncertainty in Use Case Models

Man Zhang, Tao Yue, Shaukat Ali, Bran Selic
Oscar Okariz, Roland Norgre, Karmele Intxausti

Journal paper that has been submitted to the Journal of Systems and Software (JSS), second revision

Abstract

Context: Latent uncertainty in the context of software-intensive systems (e.g., Cyber-Physical Systems (CPSs)) demands explicit attention right from the start of development. Use case modeling—a commonly used method for specifying requirements in practice, should also be extended for explicitly specifying uncertainty.

Objective: Since uncertainty is a common phenomenon in requirements engineering, it is best to address it explicitly by identifying, qualifying, and, where possible, quantifying uncertainty at the beginning stage. The ultimate aim, though not within the scope of this paper, was to use these use cases as the starting point to create test-ready models to support automated testing of CPSs under uncertainty.

Method: We extend the Restricted Use Case Modeling (RUCM) methodology and its supporting tool to specify uncertainty as part of system requirements. Such uncertainties include those caused by insufficient domain expertise of stakeholders, disagreements among them, and known uncertainties about assumptions about the environment of the system. The extended RUCM, called U-RUCM, inherits the features of RUCM, such as automated analyses and generation of models, to mention but a few. Consequently, U-RUCM provides all the key benefits offered by RUCM (i.e., reducing ambiguities in requirements), but also, it allows specification of uncertainties with the possibilities of reasoning and refining existing ones and even uncovering unknown ones.

Results: We evaluated U-RUCM with two industrial CPS case studies. After refining RUCM models (specifying initial requirements), by applying the U-RUCM methodology, we successfully identified and specified additional 306% and 512% (previously unknown) uncertainty requirements, as compared to the initial requirements specified in RUCM. This showed that, with U-RUCM, we were able to get a significantly better and more precise characterization of uncertainties in requirement engineering.

Conclusion: Evaluation results show that U-RUCM is an effective methodology (with tool support) for dealing with uncertainty in requirements engineering. We present our experience, lessons learned, and future challenges, based on the two industrial case studies.

Keywords. Use Case Modeling; Belief; Uncertainty.

1 Introduction

The problem of uncertainty in software-intensive systems such as Cyber-Physical Systems (CPSs)), is familiar to the requirements engineering community. However, it has not been adequately addressed and, therefore, it lacks both methodological and tool support in both the literature and in practice. In their well-known use case modeling book, Bittner and Spence [1] pointed out that it is essential to take the time to fill out missing areas and drill down into uncertainty. Uncertainty can be due to diverse causes, such as insufficient domain expertise or lack of information. Given the significant increases in the complexity of modern CPS and the diversity of the environments in which they are deployed, it is becoming critical to address uncertainty up front; that is, right from the start of development. This includes not only uncertainties about the requirements, but also uncertainties about its assumed operating environment.

Uncertainty in requirements has been studied in the context of dynamically adaptive systems in the presence of environmental uncertainty [2, 3]. Several goal-driven solutions [4, 5] have been proposed to handle uncertainty in similar contexts. Partial model-based solutions (e.g., [6, 7]) have been developed to support early requirements and architecture decision making. However, after conducting a literature review, we did not find any use case modeling methodology that explicitly handles uncertainty. Having such a methodology is important since use case modeling is a commonly used technique for specifying requirements in practice [6]. In our view, because uncertainty is a common phenomenon in requirements engineering, it is best to *address it explicitly* by identifying, qualifying, and, where possible, quantifying uncertainty.

The need for such a methodology arose in the context of an EU Horizon 2020 project, which focused on testing CPSs under uncertainty. The crucial first step in this project was to collect use cases with known uncertainties for two industrial CPSs and their environments. This was done with three industrial partners (Future Position X[12], ULMA[13] and Ikerlan[14], which are among the authors of this paper). The ultimate aim was to use these use cases as

[12] fpx.se/geo-sports/
[13] www.ulmahandling.com/en/
[14] www.ikerlan.es/

the starting point to create test-ready models to support automated testing of CPSs under uncertainty. To this end, we first introduced the RUCM methodology to our industrial partners and then extended it to enable specification of uncertainties. This led to the design of the U-RUCM methodology, which, to the best of our knowledge, is the first use case modeling methodology that explicitly addresses uncertainty.

As noted, U-RUCM is based on a practical use case modeling solution, called Restricted Use Case Modeling (RUCM) [8, 9]. RUCM was initially proposed by Yue et al. [9], for reducing inherent ambiguity in textual *Use Case Specifications* (UCSs) and to enable automated generation of UML models. Later on, RUCM was extended to address various industrial challenges, including requirements based testing [10] and use case based requirements inspection [11]. RUCM and its extensions have been used to address industrial challenges from various domains (e.g., telecommunication [10, 12, 13], automotive [14]).

To structure and specify uncertainties in use case models, two templates were proposed for specifying *Belief Use Case Specifications* (BUCS) and uncertainties. A BUCS annotates the UCS with uncertainty information, including the source of uncertainty, the degree (measurement) of uncertainty, the risk of uncertainty, etc., as perceived by stakeholders and based on available evidence. Such models can be automatically generated as instances of a formal U-RUCM metamodel.

Evaluation of U-RUCM based on the two industrial case studies revealed that, with U-RUCM, we were able to significantly improve on the characterization and understanding of uncertainties in the requirements (up to 306% and 512% for the two case studies) compared to base RUCM. In this paper, we summarize practical lessons learned in the course of this evaluation and also discuss future challenges.

The rest of the paper is organized as follows: Section 2 presents the background. The U-RUCM templates and keywords are explained in Section 3, followed by its formalization (Section 4). The tool support and recommended methodology are given in Section 5. Section 6 reports user experience, evaluation, lessons learned and future challenges. We discuss the related work in Section 7 and conclude the paper in Section 8.

2 Background and Running Example

In this section, we first briefly introduce the *U-Model* on which U-RUCM is based on, followed by the running example and a brief description of RUCM illustrated with the running example.

2.1 U-Model

To help us understand the nature of uncertainty in the general context of software engineering, in our previous work [15] we developed a conceptual model called *U-Model* to define uncertainty and its associated concepts. The *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 case studies from the two industrial partners of the EU Horizon 2020 project. To keep the paper self-contained, we have provided *U-Model* and definitions of its concepts in Appendix A and in the rest of the section, we briefly summarize the fundamental concepts and their relationships.

The *U-Model* takes a subjective approach to representing uncertainty. This means that uncertainty is modeled as a state (i.e., worldview) of some agents (called *BeliefAgents*), who, for whatever reason, do not have complete and fully accurate knowledge about some subjects of interest. In the *U-Model*, a *Belief* is an abstract concept, but it can be expressed in the concrete form via one or more explicit *BeliefStatement*s (a concrete and explicit specification of some *Belief* held by a *BeliefAgent* about possible phenomena or notions belonging to a given subject area). Uncertainty (i.e., lack of confidence) represents a state of affairs whereby a *BeliefAgent* does not have full confidence in a belief that it holds. This may be due to any number of factors: lack of information, inherent variability in the subject matter, ignorance, or even due physical phenomena such as the Heisenberg uncertainty principle. While *Uncertainty* itself is an abstract concept, it can be quantified by a corresponding *Measurement*, which expresses in some concrete form the subjective degree of uncertainty that the agent ascribes to a *BeliefStatement*. As the latter is a subjective notion, a *Measurement* should not be confused with the degree of validity of a *BeliefStatement*. Instead, it merely indicates the level of confidence that the agent has in a statement.

Some of the *U-Model* concepts were further extended for supporting Model-Based Testing (MBT) of CPSs under uncertainty. More specifically, we developed the Uncertainty Modeling Framework (*UncerTum*) [16, 17] for supporting MBT of CPSs, which contains a UML profile, called the UML Uncertainty Profile (UUP), for specifying and measuring uncertainties as part of UML models. UUP was derived based on the *U-Model*. *UncerTum* has been successfully used for discovering unknown uncertainties [16, 18] and generating test cases [19]. In addition, we recently developed the *UncerTolve* framework [18, 20] to evolve test ready models and uncertainty measurements in UML models specified with *UncerTum* with real operational data.

2.2 Running Example

We illustrate U-RUCM using a modified version of the SafeHome case study provided in [21]. The SafeHome system implements various security and safety features in smart homes, including intrusion detection, fire detection, and flooding. One of the key functionalities of the system is that a homeowner activates the monitoring function of the system, which continuously checks for intrusions until it is explicitly disabled. During monitoring, any occurrence of an intrusion should be detected, immediately followed by sending an intrusion notification to the homeowner and the activation of an alarm. The corresponding use case for this example, named *Monitor Windows and Doors*, is shown in . In Section 2.3, we illustrate the key elements of RUCM with the running example.

2.3 Restricted Use Case Modeling (RUCM)

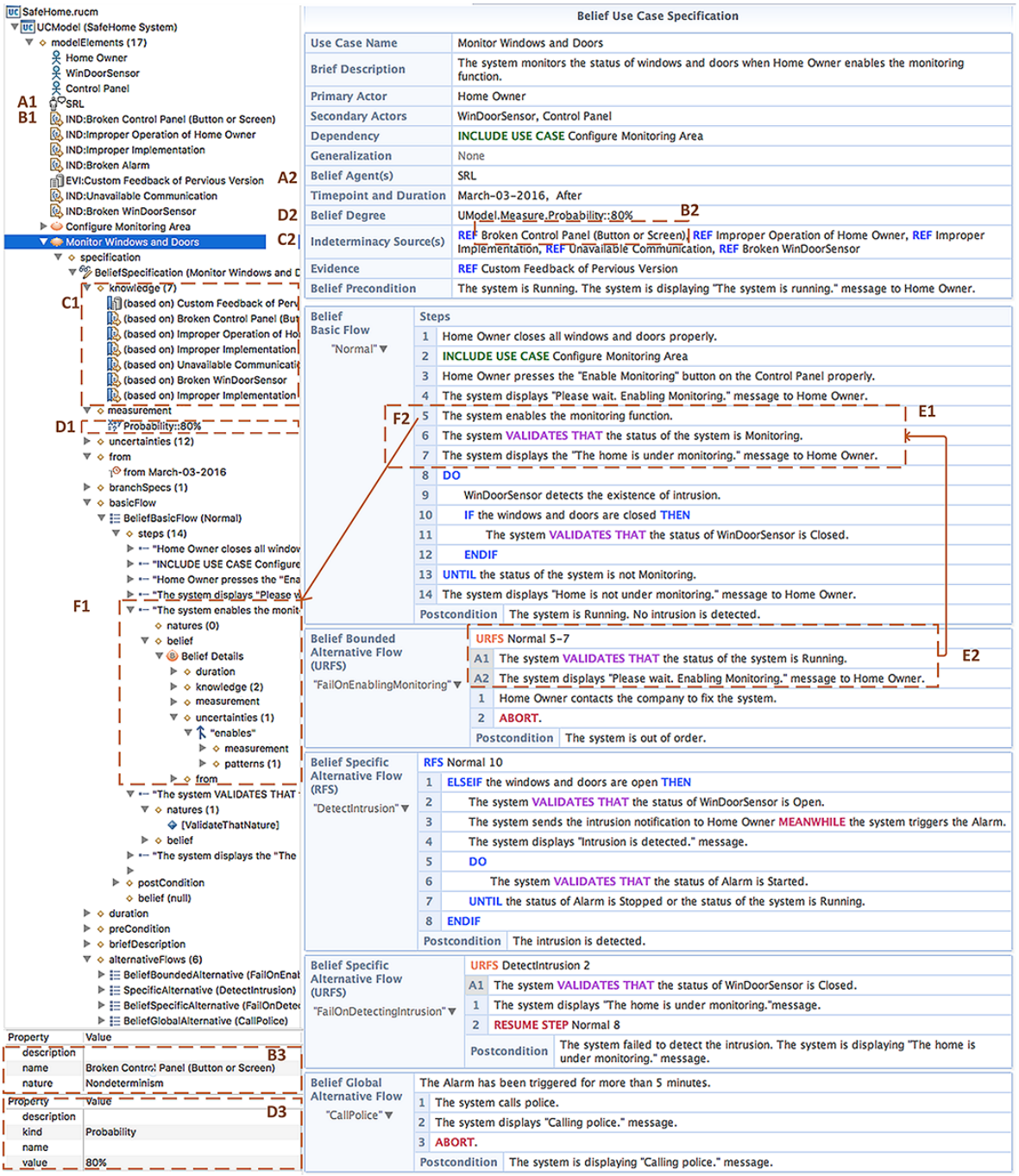
RUCM encompasses a use case template and 26 restriction rules for specifying textual UCSs [9]. RUCM aims to be easy to use, to reduce ambiguity and improve understanding, and to facilitate automated analysis. Results of two controlled experiments support these expectations [8, 9]. A RUCM UCS has one basic flow and, optionally, one or more alternative flows. An alternative flow always depends on a condition occurring in a specific step of another flow (referred to as the reference flow). We classify alternative flows into three types: A *specific* alternative flow refers to a specific step in the reference flow; a *bounded* alternative flow refers to more than one step (consecutive or not) in the reference flow; a *global* alternative refers to any step in the reference flow. For specific and bounded

alternative flows, a RFS (Reference Flow Step) section specifies one or more (reference flow) step numbers. For example, as shown in Fig. B-1, the use case has one basic flow, called *Normal*. The specific alternative flow of *DetectIntrusion* branches out from step 10 of the basic flow *Normal*, as indicated by keyword RFS and "Normal 10". The global alternative flow *CallPolice* is triggered whenever the branching condition "The Alarm has been triggered for more than 5 minutes". The bounded alternative flow of *FailOnEnablingMonitoring* refers to steps 5-7 of the basic flow *Normal*, as indicated by "URFS Normal 5-7". URFS is a new keyword introduced to U-RUCM and will be discussed in Section 3.

RUCM defines a set of keywords to specify sentences that involve conditional logic (IF-THEN-ELSE-ELSEIF-ENDIF), concurrency (MEANWHILE), condition checking (VALIDATES THAT), and iteration (DO-UNTIL). For example, as shown in Fig. B-1, step 6 of the basic flow (*Normal*) contains the keyword VALIDATES THAT, implying that the sentence of the step is a condition checking sentence.

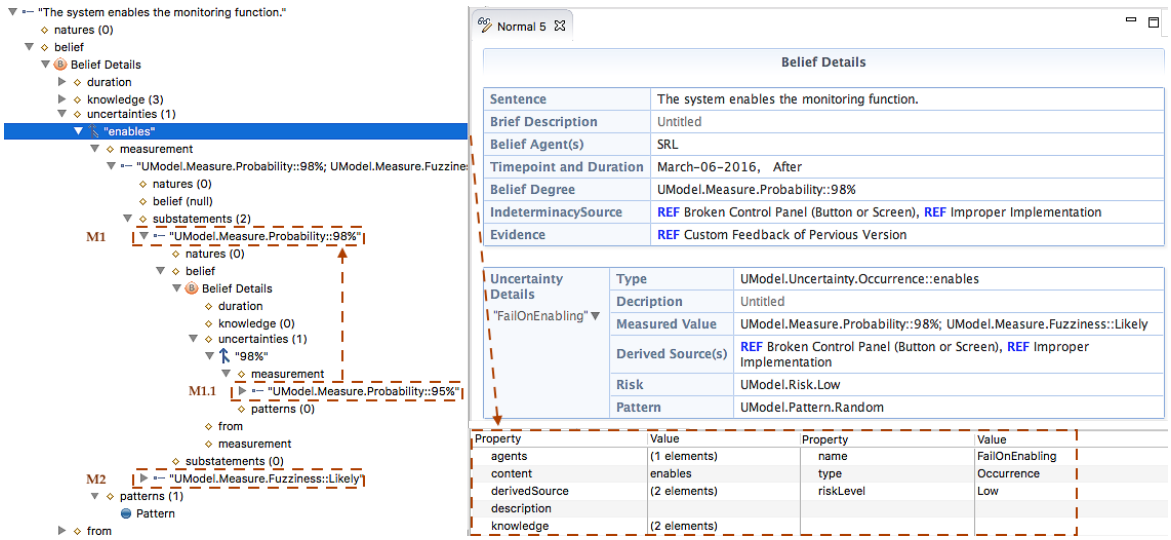
UCMeta is a metamodel that formalizes textual RUCM to facilitate automated analyses and generation of UML analysis models [8, 22]. UCMeta is specified using the OMG's standard Meta-Object Facility (MOF) [23], while the formalization of RUCM models to UCMeta instances is done automatically, as described in [8]. For example, as shown on the left panel of Fig. B-1, the basic flow *Normal* is formalized as an instance of metaclass *BasicFlow* (i.e., *basicFlow*).

Since RUCM was initially proposed by Yue et al. [24] in 2009, multiple extensions have been proposed. A restricted test case modeling methodology was presented in [10] to generate executable test cases automatically. In [14] , Wang et al. also presented another RUCM-based approach to generate test cases from use case models automatically. Both of these approaches have been evaluated using real industrial case studies. Wu et al. [25] extended RUCM for specifying safety requirements in the domain of safety-critical systems. The authors of [26] presented an approach for facilitating feature-oriented requirements validation in the context of automotive systems, where RUCM was used to specify system features.



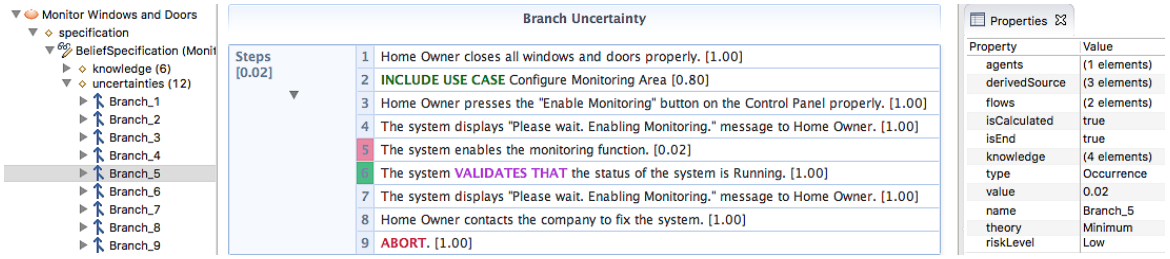
**A2 Belief Agent* is formalized into elements shown in *A1*; the properties of *B1(D1) IndeterminacySource(Measurement)* is shown in the property window *B3 (D3)*; *C2* is the set of knowledge that are formalized as elements shown in *C1*; *E2* refers to a set of sentences indicated by *E1*; the belief sentences indicated by *F2* are formalized into elements shown in *F1*.

Fig. B-1. Specifying Belief Use Case Specification of *Monitor Windows and Doors*



M1 and *M2* are the two formalized measurement statements. *M1.1* is a measurement statement defined for *M1*.

Fig. B-2. Specifying the Belief Sentence and an Associated Uncertainty (*NLUncertainty*) of *Normal* step *5* and Measurement Statement



*A step highlighted with Green (or Red) means that the condition is validated to be true (or the uncertainty does not occur).

Fig. B-3. An Example of Generated *BranchUncertainty* (across *Normal* and *FailOnEnablingMonitoring*)

3 U-RUCM Templates and Keywords

As noted in Section 2.3, the RUCM methodology has two key distinguishing features: specifying UCSs with the RUCM template, and applying the RUCM restrictions (including the keywords) to guide the way in which users use natural language to specify the control flows of UCSs. U-RUCM extends the RUCM template and proposes two U-RUCM templates and introduces two new keywords.

One of the U-RUCM templates is for specifying BUCSs as shown in Table B-1. The BUCS template inherits the key heading fields of the RUCM template, such as *Use Case Name* and *Brief Description*. In addition, U-RUCM introduces six new fields to indicate: 1)

who specifies the BUCS (*Belief Agent(s)*), 2) when it was specified and the length of time during which the belief agent(s) holds the belief (*Time Point and Duration*), 3) the degree to which the belief agent(s) believes the specification (*Belief Degree*), 4) a set of indeterminacy sources that resulted in the uncertainties of the BUCS (*Indeterminacy Source(s)*), 5) evidence used to support this BUCS and its belief and uncertainty elements (*Evidence*), and 6) the precondition of the UCS on which the belief and uncertainties are founded (*Belief Precondition*).

As discussed in Section 2, each RUCM UCS has one and only one basic flow and, optionally, three types of alternative flows. U-RUCM extends each type of event flows by 1) introducing a *belief degree*, which measures the degree to which the belief agent(s) believes a specific flow, 2) introducing a new keyword, *URFS* (Uncertain Reference Flow Step(s)), from which an alternative flow of events branches out, 3) providing the capability to annotate sentences in steps of flows and postconditions with belief and uncertainty information, and 4) introducing the new concept of *alternative steps* to enable the specification of uncertainties for alternative steps across flows of events. Note that for the case of a global alternative flow, a condition for branching from any step in the flow, should be specified via the *Belief Branching Condition* field. In Section 4, we formally define each of these fields and concepts.

We have developed an editor for U-RUCM (Section 5). The running example shown in Fig. B-1 is specified with the U-RUCM editor. The use case has one basic flow (called *Normal*) and several alternative flows. The complete specification is provided in [27] for reference.

The other U-RUCM template is used for specifying uncertainty in the belief sentence (BS) as shown in Table B-1. An example is shown in Fig. B-2. This template has fields for identifying indeterminacy sources and evidence, and for specifying uncertainty properties such as *Type*, *Pattern*, and *Measured Value*. More details are provided in Section 4.

In addition to the standard RUCM keywords (Section 2.3), REF is newly introduced for specifying associated indeterminacy sources and evidence so that they can be referenced in multiple places within a BUCS. For example, the field *Indeterminacy Source(s)* lists all the defined indeterminacy sources of the specification, each of which is referenced with REF (Fig. B-1, C1 and C2). Like the RUCM RFS keyword, the URFS keyword is used for

associating an alternative flow to the steps in its reference flow. However, RFS is used for branching out from a reference flow step under a clear condition. For example, in Fig. B-1 the alternative flow *DetectIntrusion* uses RFS to indicate that it branches from step 10 of the basic flow (*Normal*). In contrast, URFS is provided to associate uncertainties across flows of events. For example, in the running example (Fig. B-1, E1 and E2), the URFS keyword is applied to *FailOnDetectingIntrusion* to show that, in an uncertain unknown condition, it is possible that the alternative sentence *A1* can "replace" step 2 of the alternative flow *DetectIntrusion*.

Table B-1. The U-RUCM Template for Belief Use Case Specifications (BUCSs)

The template for specifying a BUCS

Use Case Name	The name of the use case. It usually starts with a verb.	
Brief Description	Summarizes the use case in a short paragraph.	
Primary Actor	The actor who initiates the use case.	
Secondary Actor(s)	Other actors the system relies on to accomplish the services of the use case.	
Dependency	Include and extend relationships to other use cases.	
Generalization	Generalization relationships to other use cases.	
Belief Agent(s)	One or more agents who hold belief about this BUCS.	
Time Point and Duration	The time point when the BUCS/BS is specified and the duration in which the belief agent(s)'s belief on the BUCS holds.	
Belief Degree	The degree to which the belief agent(s) believe the BUCS.	
Indeterminacy Source(s)	The set of indeterminacy sources related to the BUCS (REF is used).	
Evidence	Evidence to support the BUCS, and its contained belief and uncertainty elements (REF is used).	
Belief Precondition	Belief agent(s)' belief on the precondition of the BUCS, which describes what should be true before the use case is executed.	
Belief Basic Flow (*Belief degree*)	Specifies the main successful path, also called "happy path".	
	Steps (numbered)	A set of ordered **belief sentence**s.
	Belief Postcondition	Belief agent(s)' belief on what should be true after the basic flow executes.
Belief Specific Alternative Flow (*Belief degree*)	Applies to one specific step of the reference flow.	
	URFS	The reference flow step where the belief agent(s) believe there are uncertainties.
	Alternative Step	An alternative to the reference flow step.
	Steps (numbered)	A set of ordered belief sentences.
	Belief Postcondition	Belief agent(s)' belief on what should be true after the specific alternative flow executes.
Belief Bounded Alternative Flow (*Belief degree*)	Applies to more than one step of the reference flow, but not all of them.	
	URFS	A list of reference flow steps where the belief agent(s) believe there are uncertainties.

	Alternative Steps	A set of alternatives to the reference flow steps.
	Steps (numbered)	A set of ordered **belief sentences**.
	Belief Postcondition	Belief agent(s)' belief on what should be true after the bounded alternative flow executes.
Belief Global Alternative Flow (*Belief degree*)	Applies to all the steps of the reference flow.	
	Belief Branching Condition	Belief agent(s)' belief on the condition, which describes what should be true when branching from any of the steps of the reference flow.
	Steps (numbered)	The set of ordered **beliefs sentences.**
	Belief Postcondition	Belief agent(s)' belief on what should be true after the global flow executes.

The template of specifying an Uncertainty in a belief sentence

Uncertainty Details	Specifies the details of the Uncertainty in the BS.	
	Type	The type of this uncertainty (Occurrence/Content/Time/Environment/Geographical Location)
	Indeterminacy Source(s)	The set of indeterminacy sources related to this Uncertainty (REF is used).
	Measure Value	The measurement of this uncertainty.
	Risk	The possible risk led by this uncertainty.
	Pattern	The pattern of the occurrence of this uncertainty

4 U-RUCM Formalization

The formalization of U-RUCM is realized via integration of an extended version of UCMeta and *U-Model*. The full formalization is captured in a distinct metamodel, called *BeliefUCMeta*.

4.1 Relationships of BeliefUCMeta with UCMeta and U-Model

As shown in Fig. B-4, the *BeliefUCMeta* imports elements from *U-Model* and UCMeta, which formalizes RUCM. UCMeta is composed four packages: *UCTemplate* (corresponding to the RUCM template and formalizing template fields such as use case name, flows of events), *SentenceStructure* (formalizing sentence constructs such as *noun phrase*, *subject*), *SentenceSemantics* (formalizing different types of simple sentence patterns such as *subject-verb-direct object*) and *SentencePattern* (formalizing semantics functions of a use case model such as *Validation* for describing that the system validates a request and data).

Concepts of *U-Model* (Section 2.1) are divided into three groups and implemented as a metamodel, which is named as the *Technological U-Model* metamodel to distinguish itself from the *U-Model* conceptual model. *BeliefUCMeta* is composed of three packages, as shown in Fig. B-4: *BeliefUCM*, *BeliefTemplate* and *BeliefSentence*. Since *BeliefUCMeta* imports UCMeta, *BeliefUCMeta* can naturally benefit from the existing capability of UCMeta for automatically formalizing, by relying on natural language processing techniques, sentences into instances of metaclasses of packages *SentenceSemantics*, *SentencesStructure* and *SentencePattern* [8]. The complete lists of metaclasses of UCMeta, Technological *U-Model* and *BeliefUCMeta* are provided in [27] for reference.

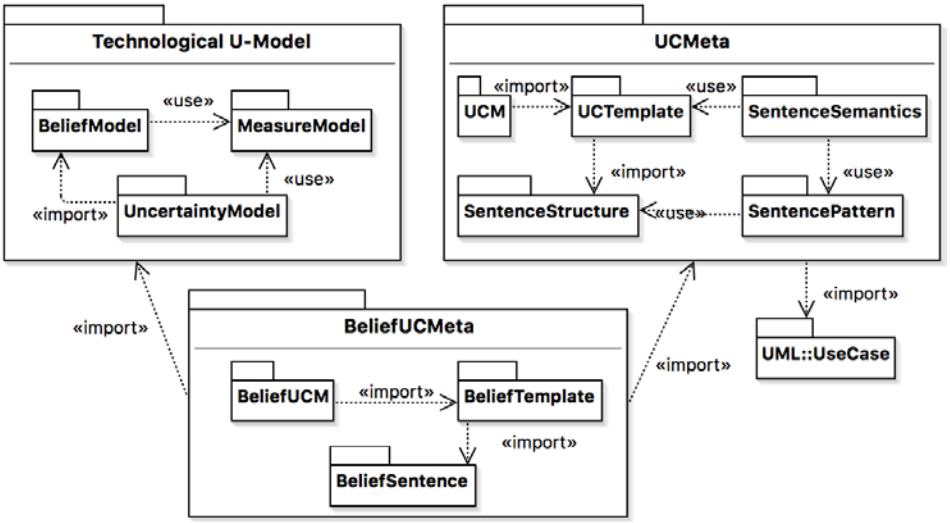


Fig. B-4. Relationships between BeliefUCMeta with UCMeta and Technological U-Model Metamodel

4.2 Belief Use Case Model, Element, and Classifier

BUElement, which specializes *UseCaseElement* of UCMeta, is the root of all the other elements of *BeliefUCMeta* (Fig. B-5). In other words, all the other metaclasses of *BeliefUCMeta* specialize *BUElement*.

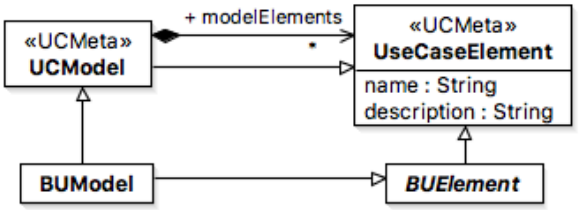


Fig. B-5. Root Elements of BeliefUCMeta

BeliefClassifier (Fig. B-6) is an abstract metaclass, introduced in U-RUCM to support the classification of a set of BUCS elements (e.g., *BeliefPrecondition*, *BeliefFlowOfEvents*), to which belief and uncertainty information can be attached. *BeliefClassifier* specializes *BeliefStatement* of *U-Model* (Appendix A.1), through which a belief classifier is associated with *Uncertainty*.

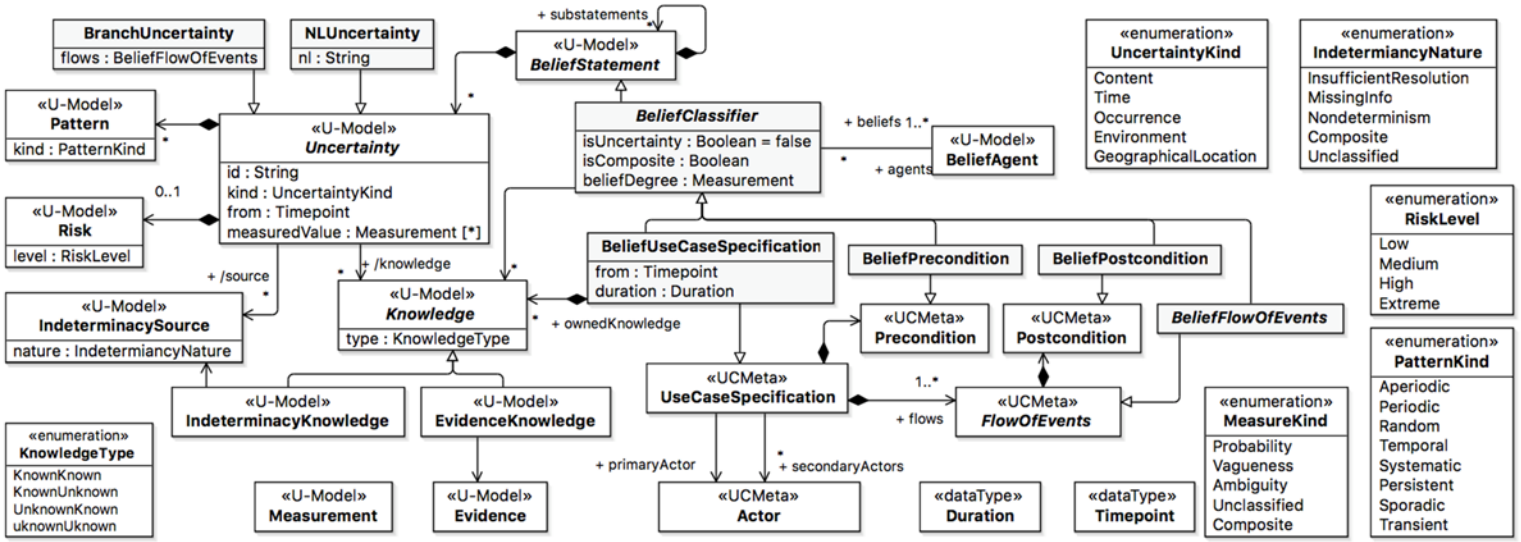
BeliefClassifier has three attributes: *isUncertainty : Boolean* (indicating if a belief classifier is associated to any known or unknown uncertainty), *isComposite : Boolean* (indicating if a belief classifier can be decomposed into finer belief elements), and *beliefDegree : Measurement* (formalizing the degree to which a belief agent believes in a belief classifier). More details on measurements are provided in Section 4.6. A belief classifier may be associated to one or more *BeliefAgent*s, which is defined, in *U-Model*, as an individual, a community of individuals sharing the same set of beliefs, or technology, such as a software system, with built-in beliefs (Appendix A.1). For example, as shown in Fig. B-1, the belief agent of the BUCS is *SRL*, the Simula team who developed the case study.

A belief classifier may be associated with *IndeterminacyKnowledge* and/or *EvidenceKnowledge*. *IndeterminacyKnowledge* is defined, in *U-Model*, to describe an objective relationship between an *IndeterminacySource* and the awareness that the *BeliefAgent* has of that source (Appendix A.1) and whereas *EvidenceKnowledge* in *U-Model* captures an objective relationship between an *IndeterminacySource* and *Evidence* (Appendix A.1). As the subtypes of metaclass *Knowledge*, *IndeterminacyKnowledge* and *EvidenceKnowledge* inherit the attribute of *Knowledge*, i.e., *type* typed with enumeration *KnowledgeType* with four literals: *KnownKnown*, *KnownUnknown*, *UnknownKnown*, and *UnknownUnknown*, definitions of which are provided in Appendix A.1 for reference. The two types of knowledge are respectively associated with *IndeterminacySource* and *Evidence*, which are discussed in the next sections with details.

4.3 Belief Use Case Specification

As shown in Fig. B-6, *BeliefUseCaseSpecification* specializes *BeliefClassifier* and *UseCaseSpecification* of UCMeta. Same as for RUCM specifications, a U-RUCM specification is composed of a set of flows of events, precondition, each flow of event is

composed of one postcondition, and each specification is associated to one primary actor and optionally associated to one or more secondary actors. In general, *BeliefUseCaseSpecification* is a concrete specification of a use case specified by a *BeliefAgent*. It includes information about the agent's confidence that the use case will execute as specified. For example, as shown in Fig. B-1, the *Monitor Windows and Doors* use case of the *SafeHome* case study is specified as a *BeliefUseCaseSpecification* (a subtype of *BeliefClassifier*), with its belief degree specified as 80%. A BUCS has two attributes: *from* and *duration*, which indicates when it was specified and the validity period of the belief. For example, for the running example, the use case was specified on March 3rd of 2016 and the belief is valid since after. Notice that these two attributes are inherited from *BeliefStatement* of *U-Model* (Appendix A.1).



UCMeta metaclasses are indicated with «UCMeta»; U-Model elements are highlighted with «U-Model»; the other metaclasses are newly proposed in U-RUCM. Constraints specified on this part of the metamodel are provided in Table B-2.

Fig. B-6. The core of BeliefUCMeta

The BUCS template of U-RUCM (Table B-1) conforms to *BeliefUCMeta*. An instance of *BeliefUseCaseSpecification* is created for each BUCS and serves as the container of other belief-related elements (e.g., *BeliefPrecondition*, *BeliefFlowOfEvents*, *BeliefSentence*). Different from a traditional UCS of RUCM, a BUCS includes specifications of *IndeterminacySource* and *Evidence* relevant to the use case. For example, as shown in Fig. B-1, the fields of *Indeterminacy Source (s)* and *Evidence* of the U-RUCM editor should be used for listing all indeterminacy sources and evidence relevant to the *Monitor Windows and Doors* use case. Notice that, the *REF* keyword should be used to refer to existing instances of *IndeterminacySource* and *Evidence* metaclasses of U-RUCM, definitions of which are from *U-Model* and provided in Appendix A for references. For example, all the indeterminacy sources defined for the belief use case model are displayed on the left side of Fig. B-1, starting with "INT". Furthermore, in the property window (e.g., the *B2* section of Fig. B-1) of an indeterminacy source, one can specify its name and nature (e.g., *Nondeterminism* of *IndeterminacyNature*). The attribute *nature* of *IndeterminacySource* is typed with *IndeterminacyNature* defined by five enumeration literals representing five possible indeterminacy sources. Notice that indeterminacy sources owned by BUCSs can be referenced by other belief elements such as *Uncertainty*. Such references are formalized as instances of *IndeterminacyKnowledge* and *EvidenceKnowledge* of *BeliefUCMeta* (Fig. B-6) such as C1 and C2 shown in Fig. B-1.

Table B-2. Constraints defined on the core part of BeliefUCMeta (Fig. B-6)

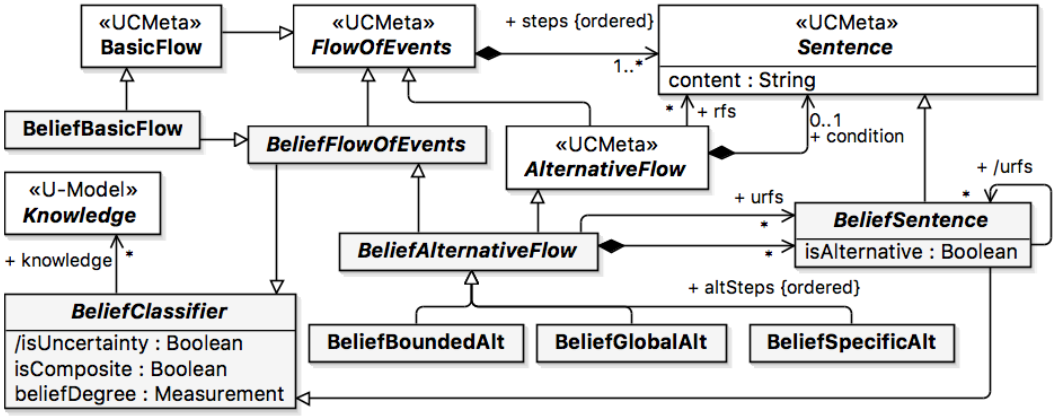
ID	Name	Constraints in Object Constraint Language (OCL)
Con1	The belief degree of a belief classifier is 100% if its *isUncertainty* attribute takes value of *False*.	**Context** BeliefClassifier **Inv**: (**not self**.isUncertainty) **implies** (**self**.beliefDegree = **null or** (**self**.beliefDegree <> **null and** **self**.beliefDegree.value = 1.0))
Con2	If a belief sentence has at least one uncertainty specified, then the belief degree of the belief sentence must not be null.	**Context** BeliefClassifier **Inv**: **self**.uncertainty->size()>0 **and self**.beliefDegree <> **null**
Con3	An uncertainty's kind should not be null.	**Context** Uncertainty Inv: self.kind <> null
Con4	An uncertainty's from and duration should not be null.	**Context** BeliefUseCaseSpecification Inv: self.from <> null and self.duration <> null
Con5	The precondition of a BUCS should be a *BeliefPrecondition*.	**Context** BeliefUseCaseSpecification **Inv**: **self**.precondition.**oclIsKindOf**(BeliefPrecondition)
Con6	All FlowOfEvents of a BUCS should be BeliefFlowOfEvents.	**Context** BeliefUseCaseSpecification **Inv**: **self**.flows->forAll(f:UCMeta::UCTemplate::FlowOfEvents| f.**oclIsKindOf**(BeliefFlowOfEvents))
Con7	All the uncertainties owned by a BUCS are *BranchUncertainty*.	**Context** BeliefUseCaseSpecification **Inv**: **self**.uncertainty->forAll(u:Uncertainty|u.**oclIsKindOf**(BranchUncertainty))
Con8	Knowledge associated to *BeliefSentence*, *BeliefFlowOfEvents* and Uncertainty must belong to the knowledge owned by *BeliefUseCaseSpecification*.	**Context** BeleifUseCaseSpecification **Inv**:**self**.flows- >forAll(f:UCMeta::UCTemplate::FlowOfEvents|**self**.ownedKnowledge-> includesAll(f.**oclAsType**(BeliefFlowOfEvents).knowledge) **and** f.steps->forAll (bs:UCMeta::UCTemplate::Sentence|**self**.ownedKnowledge- >includesAll(bs.**oclAsType**(BeliefSentence).knowledge) **and** (bs.**oclAsType**(BeliefSentence).uncertainty->size() > 0 **implies** bs.**oclAsType**(BeliefSentence).uncertainty- >forAll(u:Uncertainty| **self**.ownedKnowledge->includesAll(u.knowledge)))))

4.4 Belief Flow of Events

BeliefFlowOfEvents is a subtype of *BeliefClassifier* and also extends *FlowOfEvents* of UCMeta (Fig. B-7). Hence, it inherits *BasicFlow* and the three types of alternative flows: *Specific*, *Bounded* and *Global* of RUCM and therefore UCMeta, as shown in Fig. B-7. A belief flow of events is composed of a set of sentences, which are all belief sentences (Section 4.5), as enforced by the Con9 constraint (Table B-3). *BeliefFlowOfEvents*, as a specific type of *BeliefClassifier*, has a derived association to *Knowledge* (Fig. B-7), through

which a belief flow of events can be associated with indeterminacy sources and evidence if needed.

Specific and bounded alternative flows can be differentiated based on whether RFS or URFS is used to refer to one or more steps of a reference flow. RFS is used only when the branching condition is fully clear to the belief agent. In other words, the belief sentence corresponding to the branching condition has its belief degree specified at 100% and its *isUncertainty* attribute specified as *False* (the Con1 constraint, Table B-2). For example, the *DetectIntrusion* flow in Fig. B-1 branches out from step 10 of the basic flow under the condition that "the windows and doors are open". The belief degree to the sentence (step *1* of *DetectIntrusion*) corresponding to this condition is 100%.



Constraints specified on this part of the metamodel are provided in Table B-3.

Fig. B-7. BeliefFlowOfEvents of BeliefUCMeta

In contrast, URFS is used when the belief agent is not fully confident about a particular system behavior or condition (represented by one or more steps in a flow). In principle, an URFS alternative flow should always be linked to one or more indeterminacy sources. If such indeterminacy sources are known, U-RUCM provides a way to specify them (see Section 4.3). For example, the *FailOnEnablingMonitoring* flow is defined as the belief agent is not fully confident that "the system enables the monitoring function" (step 5 of the basic flow, Fig. B-1) will actually occur. URFS (instead of RFS) is used in this context because in which condition the event occurs and in which condition it does are not known at the time when the flow of events was specified.

Table B-3. Constraints defined on the BeliefFlowOfEvents part of BeliefUCMeta (Fig. B-7)

#	Name	Constraint
Con9	All sentences in belief flows of events are *BeliefSentences*.	**Context** BeliefFlowOfEvents **Inv**: **self**.steps->forAll(bs:UCMeta::UCTemplate::Sentence|bs.**oclIsKindOf**(BeliefSentence)) **and** (**self**.**oclIsKindOf**(AlternativeFlow) **implies** **self**.rfs->forAll(s:UCMeta::UCTemplate::Sentence|s.**oclIsKindOf**(BeliefSentence)))
Con10	All *altSteps* of a BeliefAlternativeFlow are alternative sentences (with *isAlternative* being *true*) and *urfs* of *Sentence* is a subset of *urfs* of the *BeliefAlternativeFlow*	**Context** BeliefAlternativeFlow **Inv**: **self**.altSteps->size()>0 **implies** **self**.altSteps->forAll(s:UCMeta::UCTemplate::Sentence|s.**oclAsType**(BeliefSentence).isAlternative **and** **self**.urfs->includesAll(s.**oclAsType**(BeliefSentence).urfs))
Con11	The condition sentence of a *BeliefGlobalAlt* must not be null and it must be a *BeliefSentence*.	**Context** BeliefGlobalAlt **Inv**: **self**.condition->size()>0 **and** **self**.condition.**oclIsKindOf**(BeliefSentence)
Con12	There must be no uncertainties specified in any RFS referred sentences.	**Context** BeliefAlternativeFlow **Inv**: **self**.rfs->size()>0 **and** **self**.urfs->size()=0 **and** **self**.rfs->forAll(s:UCMeta::UCTemplate::Sentence|s.**oclAsType**(BeliefSentence).uncertainty->size()=0)
Con13	At least one uncertainty must be specified in URFS referred sentences and the size of altSteps is more than 1.	**Context** BeliefAlternativeFlow **Inv**: **self**.rfs->size()=0 **and** **self**.urfs->size()>0 **and** **self**.urfs->forAll(s:UCMeta::UCTemplate::Sentence|s.**oclAsType**(BeliefSentence).uncertainty->size()>0) **and** **self**.altSteps->size()>0
Con14	At least one *AlternativeSentence* must be specified for a URFS alternative flow.	**Context** BeliefAlternativeFlow **Inv**: **self**.urfs->size()>0 **and** **self**.altSteps->size()>0 **and** **self**. altSteps->select(a:AlternativeSentence| a.urfs->size()>0)->size()>0 **and** **self**.altSteps->forAll(a:AlternativeSentence|self.urfs->includesAll(a.urfs))

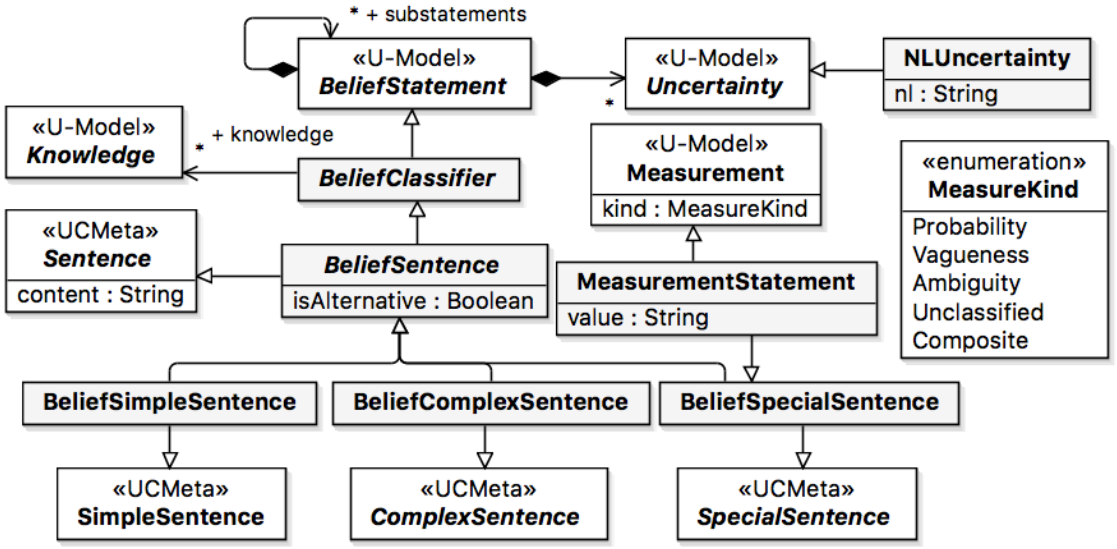
For global alternative flows, there is no need to use RFS and URFS. However, a global condition must be defined for a global alternative flow (Fig. B-7), which is a constraint defined by RUCM and also applies to U-RUCM. For example, as shown in Fig. B-1, *CallPolice* is a global alternative flow, which defines the global condition: "The Alarm has been triggered for more than 5 minutes." Since the condition is a belief sentence, one can associate uncertainties with it if needed.

In summary, U-RUCM provides four different ways of specifying alternative flows: *Bounded* with RFS, *Specific* with RFS, *Bounded* with URFS and *Specific* with URFS. Global alternative flows cannot be combined with RFS and therefore URFS, as we discussed above. For any URFS alternative flow, there should be at least one alternative sentence specified in the alternative flow, which "replaces" the referenced steps of the reference flows of events

defined in the URFS statement (the Con14 constraint, Table B-3). Any RFS alternative flow should not contain any alternative step (the Con12 constraint, Table B-3). Note that alternative sentences (which are ordered as sequential steps) are defined at the beginning of an URFS alternative flow, followed by a sequence of regular belief sentences.

4.5 Belief Sentence

All sentences in a BUCS are belief sentences, which are formalized as *BeliefSentence* elements (specializing the *Sentence* concept of UCMeta). Since *BeliefSentence* is a subtype of *BeliefClassifier*, belief sentences inherit all the attributes of *BeliefClassifier* (e.g., *beliefDegree*, *isUncertainty*, Fig. B-6), which distinguish themselves from regular RUCM sentences. In RUCM and UCMeta, sentences are classified into *simple*, *complex* and *special* sentences. Consequently, U-RUCM and *BeliefUCMeta* classify belief sentences into *BeliefSimpleSentence*, *BeliefComplexSentence* and *BeliefSpecialSentence* (Fig. B-8).



Constraints specified on this part of the metamodel are provided in Table B-4.

Fig. B-8. *BeliefSentence* of BeliefUCMeta

A belief sentence can be associated with *Uncertainty* via *BeliefClassifier* and *BeliefStatement* and with *IndeterminacySource* and *Evidence* via *Knowledge* and *BeliefClassifier* as shown in Fig. B-6. A belief sentence uncertainty must be a *NLUncertainty* (the Con15 constraint, Table B-4) and the *String* value of its *nl* attribute should be part of the content of the belief sentence (the Con16 constraint, Table B-4). More details about *NLUncertainty* are discussed in Section 4.6.

A belief simple sentence is an atomic belief statement, which, from the sentence structure perspective, is composed of only one subject and one predicate. Belief simple sentences can appear in action steps of flows, preconditions, postconditions, and other fields of a BUCS. Belief complex sentences are sentences with the following RUCM keywords applied: IF-THEN-ELSE-ELSEIF-THEN-ENDIF for conditions, DO-UNTIL for iterations, MEANWHILE for concurrency, and VALIDATE THAT for validation. A complex sentence can consist of one or more belief simple sentences. Belief special sentences are sentences involving keywords RESUME STEP, ABORT, RFS, EXTENDED BY, INCLUDE and URFS. Notice that special sentences involving keywords URFS are newly introduced in U-RUCM, and all the other special sentences are inherited from RUCM. It is important to point it out that U-RUCM benefits from the existing capability of UCMeta (which formalizes RUCM) for providing the formalization of the sentence structures, sentence patterns and sentence semantics (Section 4.1). U-RUCM can also benefit from the automated solution (i.e., aToucan [8]) of formalizing natural language sentences into instances of the metaclasses of these UCMeta packages. Doing so provides opportunities for automatically analyzing formalized belief use case models and transforming them into other artifacts when needed. For example, one possibility is to transform belief use case models into UML models specified in *UncerTum* [16] for facilitating MBT. However, providing such capability is beyond the scope of this paper.

In U-RUCM, we also introduce alternative sentences, which are formalized as the *isAlternative* attribute of *BeliefSentence* (Table B-4). From the natural language perspective, alternative sentences have no difference with other belief sentences. The only difference is that alternative sentences can only appear in URFS alternative flows as action steps (Section 4.4). In the current implementation of the U-RUCM editor, alternative sentences (e.g., *A1* and *A2* of *FailOnEnablingMonitoring*, Fig. B-1) are denoted with A1, A2, etc. An alternative sentence can be any type of belief sentences: *simple*, *complex* or *special*, as shown in Table B-4.

We also define, in *U-Model*, *MeasurementStatement*, a special type of *BeliefSpecialSentence* (Fig. B-8). For example, as shown in Fig. B-2, the *Measured Value* field of the uncertainty specifies two measurement statements, i.e., *UModel.Measure.Probability::2%* and *UModel.Measure.Fuzziness::Likely*. Since it is a

special type of belief special sentences, a measurement statement can be associated with all the belief sentence properties (including derived ones) such as *Uncertainty*. Please refer to Section 4.8 for detailed discussions of measurement statements, measurements and measures.

Table B-4. Constraints defined on the BeliefSentence part of BeliefUCMeta (Fig. B-8)

#	Name	Constraint
Con15	All the uncertainties associated to a belief sentence are of the *NLUncertainty* type.	`Context BeliefSentence` `Inv: self.uncertainty->size()>0 implies self.uncertainty->forAll(u:Uncertainty|u.oclIsKindOf(NLUncertainty))`
Con16	The attribute of *nl* of an *NLUncertainty* must not be null and it should be part of *content* of the belief sentence.	`Context BeliefSentence` `Inv: self.uncertainty->size()>0 implies self.uncertainty->forAll(u:Uncertainty| u.oclAsType(NLUncertainty)<> null and self.content.contains(u.oclAsType(NLUncertainty).nl))`
Con17	The attribute of *nl* of an *NLUncertainty* must not be null and it should be part of *value* of the measurement statement.	`Context MeasurementStatement` `Inv: self.uncertainty->size()>0 implies self.uncertainty->forAll(u:Uncertainty| u.oclAsType(NLUncertainty)<> null and self.value.contains(u.oclAsType(NLUncertainty).nl))`
Con18	The Measure of each Measurement owned by the same Uncertainty is different.	`Context Uncertainty` `Inv: self.measurement->size()>0 implies self.measurement->forAll(u1,u2:Measurement| u1.kind <> u2.kind)`

4.6 Uncertainty

Uncertainty is defined, in *U-Model*, for "representing a state of affairs whereby a belief agent does not have full confidence in a belief that it holds" (Appendix A.1). We adopt the definition of uncertainty from *U-Model* to U-RUCM and associate it with *BeliefClassifier*, as shown in Fig. B-6.

In *U-Model*, uncertainties are classified into five different types, which are inherited by U-RUCM and formalized as the five literals of enumeration *UncertaintyKind* in *BeliefUCMeta* (Fig. B-6): *Content*, *Time*, *Occurrence*, *Environment*, and *GeographicalLocation*. Definitions of these uncertainty types are provided in Appendix A.2 for reference. For example, the type of the uncertainty described in Fig. B-2 is *Occurrence* as indicated by the content in the field of *Type* of the editor.

Each uncertainty has a time point describing when the uncertainty is initialized. This attribute is formalized as an attribute of metaclass *Uncertainty* of *BeliefUCMeta* (Fig. B-6). For example, the uncertainty in Fig. B-2 was specified in *March-03-2016* and is active since then. An uncertainty can be optionally associated with *Risk*. Currently, in *U-Model* and therefore in U-RUCM, we define four risk levels: *Low*, *Medium*, *High* and *Extreme* (enumeration *RiskLevel*, Fig. B-6). For example, the uncertainty in Fig. B-2 is of low risk. To further characterize an uncertainty, it can be optionally (if information available) associated with one or more patterns (e.g., *Aperiodic*, *Sporadic*), which are formalized as enumeration *PatternKind* in *BeliefUCMeta* (Fig. B-6). For the uncertainty described in Fig. B-2, the belief agent is not aware of in which pattern the uncertainty appears, and therefore it is not specified. An uncertainty can be measured (i.e., *measuredValue: Measurement*, Fig. B-6) in different ways (i.e., *MeasureKind*, Fig. B-6). Details are described in Section 4.8.

In U-RUCM, we classify uncertainties into two basic types: *NLUncertainty* and *BranchUncertainty* (Fig. B-6). *NLUncertainty*(s) are defined at the level of belief sentences. Branch uncertainties can be derived automatically from flows of events and are owned by BUCSs (the Con7 of Table B-2). In addition to these two categories, in U-RUCM, uncertainties in flows of events can also be captured via URFS and alternative flows (Section 4.4).

4.6.1 Uncertainty in Belief Sentences (NLUncertainty)

An *NLUncertainty* refers to a Part of Speech (PoS) (e.g., noun, verb) of a belief sentence, about which a belief agent lacks confidence. This is enabled by the *nl* attribute of *NLUncertainty* (Fig. B-6). For example, one instance of *NLUncertainty* in step 5 of the basic flow (Fig. B-2) shows that the belief agent is 98% confident about the occurrence of the event. The uncertainty is associated with "*enables*", which is the predicator verb of the belief sentence. Notice that the *nl* attribute is typed with *String*. However, if automated solutions can always be proposed to parse a belief statement and automatically associate an *NLUncertainty* (through specified *nl* information) with constructs of a belief sentence (e.g., subjects, predicators, objects) if needed. As we discussed in Section 4.5, UCMeta and aToucan provide such a capability, from which U-RUCM can benefit.

An *NLUncertainty* in a belief sentence can be optionally associated with one or more indeterminacy sources owned by the BUCS, the container of all belief sentences and therefore uncertainties (Fig. B-6). Doing so helps to provide additional information describing situations whereby the information required to ascertain the validity of the belief sentence is indeterminate (Appendix A.1). For example, as shown in Fig. B-2, the uncertainty is associated with two indeterminacy sources, implying that the belief agent believes that there is a 98% probability (not 100%) that the system enables the monitoring function and therefore 2% probability that the system does not enable monitoring under unknown conditions due to the broken control panel or improper implementation of the monitoring functionality. Furthermore, the predefined indeterminacy source categories (e.g., *MissingInfo, Non-determinism*, defined in Appendix A.1), formalized as the literals of enumeration *IndeterminacyNature* (Fig. B-6) provides more information about indeterminacy sources and therefore uncertainties. For example, the indeterminacy source of *Broken Control Panel (Button or Screen)* is of a *Nondeterminism* indeterminacy nature (as shown in the left bottom corner of Fig. B-1), implying that the phenomenon of the control panel being broken is either practically or inherently non-deterministic.

It is also worth mentioning that *U-Model* allows one to specify uncertainties in measurement statements, which is enabled by the fact that a measurement statement is a belief sentence and therefore it can have uncertainties specified. We, however, in *U-Model*, enforce that when specifying such an uncertainty, its *nl* attribute should be part of the value or the whole of it (Con17, Table B-4). Notice that a measurement statement is composed of two parts: measure and value (Section 4.8). Such an uncertainty should be a *Content* type of uncertainty. For the example given in Fig. B-2, if an uncertainty is associated with the measurement statement (i.e., *UModel.Measure.Probability::98%*), *nl* of the uncertainty should be *98%*.

4.7 Branch Uncertainty

A set of branches can be derived from a BUCS, systematically by following different strategies. For example, in [10], we defined three test coverage strategies: all branch coverage strategy (covering all branches), all condition coverage strategy (covering all conditions of all branches at least once), and loop coverage strategy (ensuring that each loop

(DO-UNTIL) is exercised exactly one, none, and *x* number of times, where *x* can be specified beforehand). We adopt these three coverage strategies and use them to generate branches systematically from U-RUCM models. Each of these derived branches represents a straight path from the precondition of the specification all the way to a postcondition of a flow of events. One example of such a branch is provided in Fig. B-3, which is automatically generated from the use case specification provided in Fig. B-1.

The occurrence of a particular path might be uncertain, which therefore forms a branch uncertainty. Such an uncertainty is an instance of *BranchUncertainty* with the *Occurrence::UncertaintyKind* kind. Since such branches can be automatically generated, measurements of the branch uncertainties can be automatically calculated when needed, if and only if uncertainties of the belief sentences of the specifications are specified. For example, as shown in Fig. B-3, the branch takes the path of branching from step 5 and executing the *FailOnEnablingMonitoring* alternative flow and finishing at its last step. Since the chance of the system does not enable the monitoring is 1-0.98=0.02 as indicated at step 5 of the branch, the overall branch uncertainty can then be defined as 0.02 if we follow the simple strategy of taking the lowest value of the belief degrees of the belief sentences as the branch uncertainty measurement. We acknowledge that there are many alternatives regarding how to systematically obtain branch uncertainty measurements. Users can also manually define such measurements if they want. However, studying such strategies is out of the scope of this paper.

4.8 Measurement

As discussed in Section 4.5, in U-RUCM, we define *MeasurementStatement* as a special type of *BeliefSpecialSentence* (Fig. B-8). *MeasurementStatement* also inherits *U-Model*'s *Measurement*; therefore, each measurement statement should be associated with a specific type of *Measure*s (Fig. B-8). A U-RUCM measurement can take different kinds of measures such as *Probability*, *Vagueness,* and *Ambiguity*, which are formalized as enumeration *MeasureKind* of *BeliefUCMeta* (Fig. B-6). For example, as shown in Fig. B-2, two measurements are specified for the uncertainty in the field of *Measured Value* as *UModel.Measure.Probability::98%* and *UModel.Measure.Fuzziness::Likely* indicating that the probability the occurrence of enabling monitoring is 98% if measured with *Probability*

and it likely occurs if it is measured with *Fuzziness*. Note that *U-Model* defines a taxonomy of measures (Appendix A.3), which is integrated in U-RUCM as it is.

In U-RUCM, there are two situations where measurement statements should be specified: 1) for quantifying the belief degree of a belief classifier, and 2) for quantifying uncertainty. We would like to point it out that all belief degree and uncertainty measurements are subjective. This is because, at the requirements level, domain experts specify belief degrees and uncertainty measurements based on their experience, knowledge, and even preferences, as opposed to basing them on available hard data.

One special case is to define uncertainties in measurement statements as the way how uncertainties are specified for other types of belief sentences. This is enabled because measurement statements are defined as a special type of belief sentence, as we discussed earlier. In other words, the belief agent can attach an uncertainty (e.g., with the measurement statement specified as: *UModel.Measure.Probability::95%*, *M1.1* as shown in Fig. B-2) to the value of a measurement statement (e.g., *98%*) to indicate that she/he is not fully confident about the measurement statement (e.g., *UModel.Measure.Probability::98%*).

Since different belief agents might have different views on belief degree and uncertainty measurements, *U-Model* enables this by specifying a belief degree (or uncertainty measurement) as a specific type of *BeliefClassifier*, which is associated with one or more *BeliefAgent*s as shown in Fig. B-6. Moreover, *U-Model* also allows a belief agent to specify more than one measurements for an uncertainty; however, each of these measurements is enforced to take different kinds of measures (Con18, Table B-4).

U-RUCM (along with its editor) also provides the capability of specifying measurement statements as belief special sentences in the sense that a measurement statement is divided into two parts: the measure and the value with the format of *measure::value*. For example, as shown in Fig. B-2, one measurement statement corresponding to the uncertainty is specified as *UModel.Measure.Probability::98%*. Notice that the measure taxonomy of *U-Model* (Appendix A.3) has been embedded as part of the U-RUCM editor. Therefore, when typing "UModel.Measure.", all the measures of the taxonomy will automatically be listed in a drop list for selection. Based on this format, a measurement statement can be automatically parsed, and an instance of *Measurement* will be automatically created and a value will be assigned to the *value* attribute of the measurement statement.

5 Tool Support and Methodology

5.1 Tool Support

BUCSs are specified in the U-RUCM editor, which is implemented in a modeling framework, called the *Lightweight Modeling Framework* (LMF [28]). This framework implements functionality similar to those of the Eclipse Modeling Framework (EMF), but with a lightweight design with the aim of reducing tight coupling with Eclipse to facilitate easier porting to other platforms. LMF has two editors: a reflective model editor and a metamodel editor. The LMF Reflective Editor is a simple model editor implemented with the LMF metamodel reflection mechanism. The metamodel editor allows editing a LMF metamodel.

BUCSs specified with the editor can be automatically formalized into instances of *BeliefUCMeta* concepts. In the past, we have developed the transformation from RUCM to UCMeta, based on natural language processing techniques [8]. The transformation from U-RUCM to *BeliefUCMeta* is just an extension of the transformation from RUCM to UCMeta. The formalized specifications can be directly used for performing different kinds of analyses and generations of other artifacts when needed.

We have made a video to demonstrate the U-RUCM editor and the formalization from U-RUCM to *BeliefUCMeta*, along with the metamodel of *BeliefUCMeta*, UCMeta and *U-Model* in [15] for references. Note that Fig. B-1, Fig. B-2, Fig. B-3 and Fig. Appx-7 also demonstrate the user interfaces of the tool.

5.2 Methodology

Though U-RUCM can be used in many different ways, in this section, we recommend one methodology based on our own experience. It starts with the creation of a use case model, specifying the actors, use cases, and relationships among them (Fig. B-9). The belief agents in this case are the requirements engineers who capture the information, including indeterminacy sources, evidence, and uncertainty degrees from the various stakeholders. Of course, it is always possible to revisit the initial specifications subsequently should new evidence or indeterminacy sources be uncovered.

When the overall context of a belief use case model is established, one can start to develop a BUCS for each use case. The key steps of developing BUCSs are presented as a UML activity diagram in Fig. B-10. There is no particular order for specifying primary and secondary actors, belief agents. We recommend a sequence for guiding requirements engineers through the process that proceeds from simple tasks to more complicated ones. Specifying flows of events is the most challenging task, as it requires a lot of careful analysis, discussions, and design. The process is always iterative, although we do not show this in Fig. B-10 for reasons of simplicity.

When specifying belief alternative flows (Fig. B-10), belief global alternative flows are often used to specify exceptions and behaviors crosscutting all the steps of a reference flow. The key task here is to identify the proper branching condition. If one needs to refer to one or more (but not all) steps of a reference flow, belief specific or bounded alternative flows can be created. As discussed in Section 4.4, U-RUCM provides four different ways of specifying belief alternative flows and some constraints (e.g., alternative sentences only appear in URFS alternative flows) should be applied when using U-RUCM in this aspect. In our current implementation of the editor, we have enforced these constraints (all the constraints specified in Table B-2, Table B-3 and Table B-4) so that chances of violating them are eliminated. By definition, URFS and RFS are different and therefore should be applied in different situations, as discussed in Section 4.4. We highly recommend using URFS to identify uncertain alternative flows only after the entire structure of flows of events (using RFS) of a BUCS is defined.

Each flow of events consists of a set of steps, which are specified as belief sentences. For each belief sentence, one should refer to one or more relevant indeterminacy sources and evidence, based on which, one can define the belief degree and associated uncertainties. The essential activity of specifying a belief sentence is about specifying associated uncertainties if there are any. The key steps of specifying uncertainties of the *NLUncertainty* type for belief sentences are presented as a UML activity diagram in Fig. B-11.

As discussed in Section 4.6, uncertainties can be more precisely characterized by pattern and risk information and measured in different ways. In practice, it is not always possible to obtain and enter all of this information at once. So, a rule of thumb is to first identify as many uncertainties as possible and only then refine them with more detailed information. The

recommendation is also based on the fact that it is more important, at the requirements specification stage, to spend time (which is often limited) on identifying more uncertainties than elaborating on details of already identified uncertainties. If one wants to elaborate on the details of an already identified uncertainty, it is recommended to start from identifying associated indeterminacy sources. This is because identifying indeterminacy sources (e.g., *REF Broken Control Panel (Button or Screen)* in Fig. B-2) might lead to the discovery of previously-unknown uncertainties that might be caused by this indeterminacy (e.g., uncertainties due to a broken control panel).

We also recommend a methodology for specifying measurement statements. The key steps of the methodology are presented in Fig. B-12 as a UML activity diagram. If a measurement statement contains uncertainties, then a procedure similar to the one for specifying *NLUncertainty* for belief sentences can be followed.

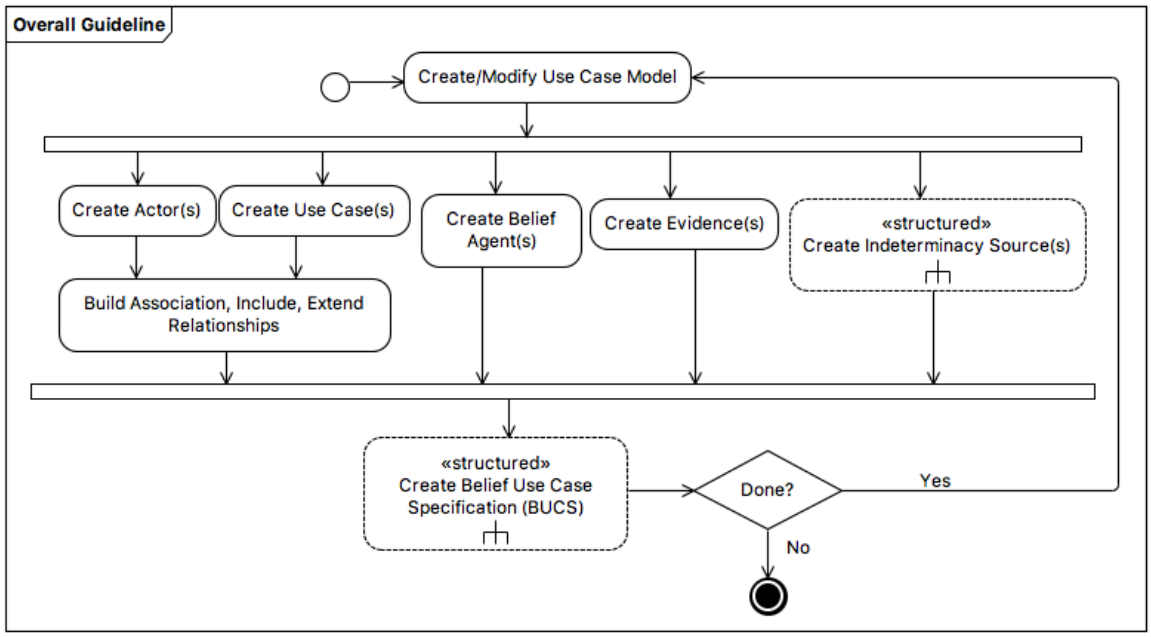


Fig. B-9. Methodology for creating a use case model (in UML Activity Diagram)

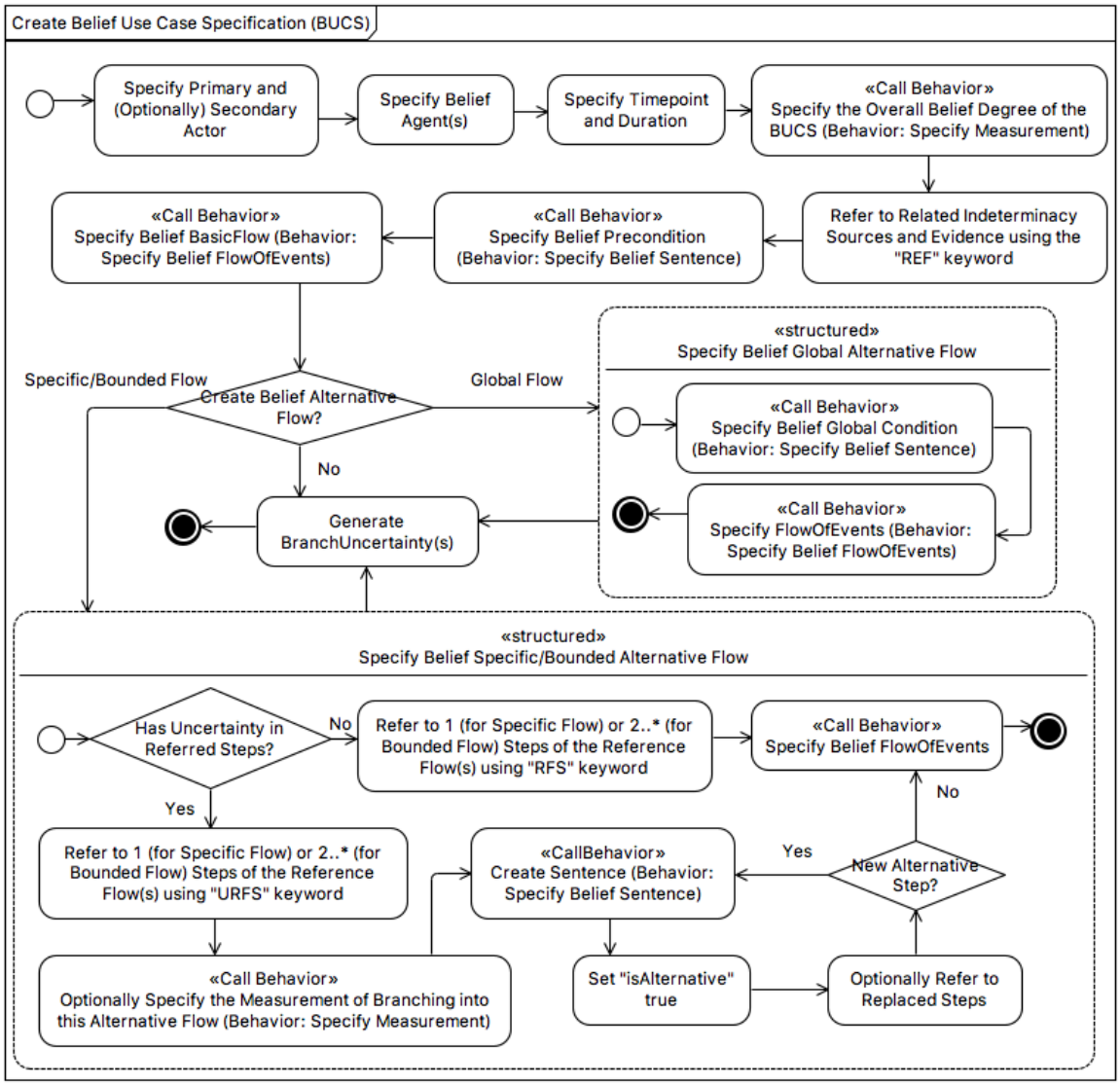


Fig. B-10. Methodology for specifying BUCSs (in UML Activity Diagram)

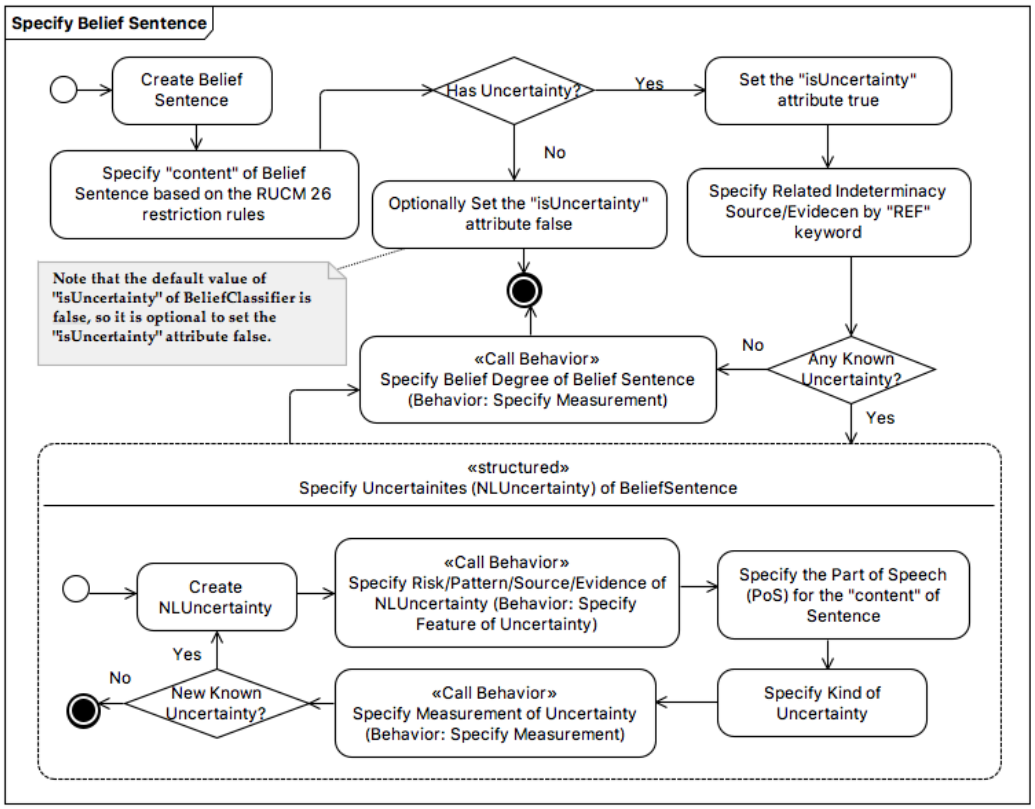


Fig. B-11. Methodology for specifying belief sentences (in UML Activity Diagram)

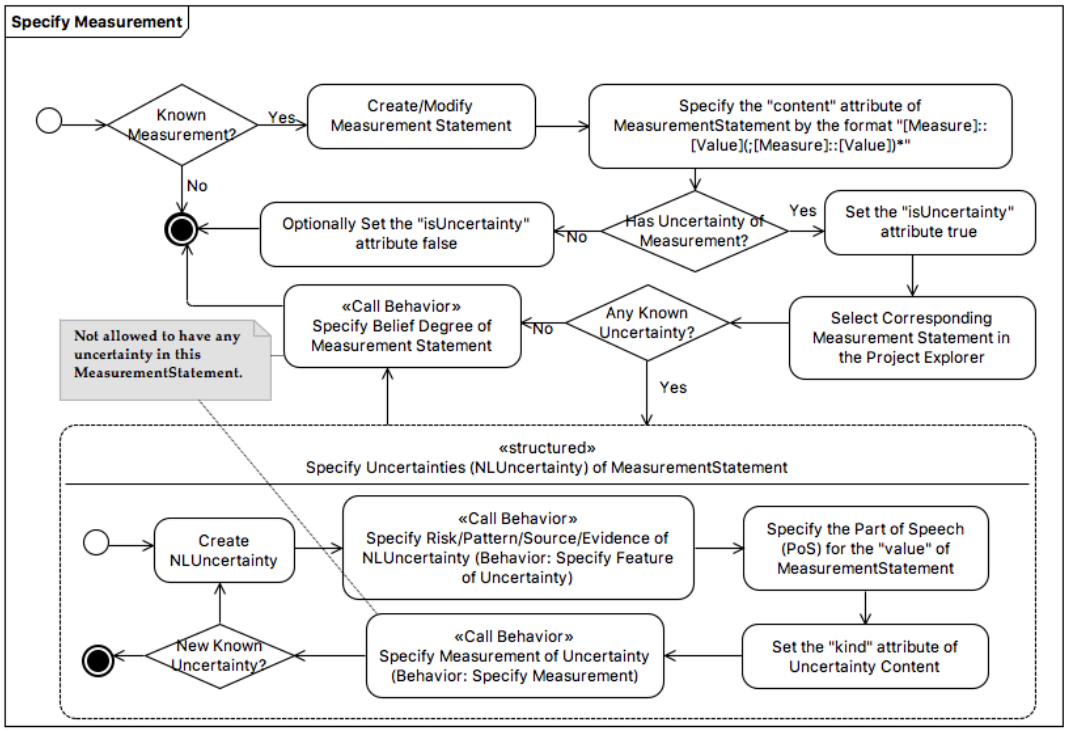


Fig. B-12. Methodology for specifying measurements (in UML Activity Diagram)

6 Evaluation

The overall objective of the evaluation was to assess, in an industrial setting, whether U-RUCM was effective regarding facilitating the development of use case models with the explicit focus on uncertainty. In *U-Model* Section 6.1, we briefly describe the two industrial case studies. In Section 6.2, we present the context, design, and execution of the evaluation. Results of the case studies are presented in Section 6.3. In Section 6.4, we share our experience, lessons learned and identified future challenges.

6.1 Case Studies

One of the two industrial case studies involved the Automated Warehouse (AW) from ULMA Handling Systems. These complex systems serve to monitor, control, and manage warehouses for goods of different types, such as food and beverages and textiles. Each handling facility (e.g., crane, conveyor) forms a physical unit, and together they are dedicated to one handling system application (e.g., storage).

The second industrial case study used the Geo Sports (GS) system from Future Position X [29], Sweden. This system measures the performance of an individual or a team as well as the conditions of athletes over a sustained period in actual game environments (e.g., a soccer field). The measurements (e.g., heartbeat, speed, location) are made continuously and in real time using geo-position sensors during both training and competition. These measurements are communicated during a game via a receiver station to the OpenField[15] system, where coaches can monitor them at runtime and take actions when needed. 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. Our case study involved Bandy, a form of ice hockey played predominantly in Northern Europe and Russia. To the best of our knowledge, this project was the first to monitor sports on ice using sensors. We consider GS as a human-in-the-loop CPS [30].

15 A cloud-based analytics platform for reporting and presenting data (see https://www.catapultsports.com/products/openfield for more information).

6.2 Context, Design, and Execution of Evaluation

The work was conducted in the context of the U-Test[16] project. The development of U-RUCM is an iterative process and interwoven with the activities of developing *U-Model*, eliciting, refining and validating uncertainty requirements of the two industrial case studies. Both researchers and industrial partners were involved in the process, during which intermediate versions of U-RUCM were developed. We consider that being transparent and therefore reporting the process in the paper are important since it provides an opportunity for readers to comprehend the rigorousness of the process and therefore gain confidence on the derived U-RUCM methodology. Moreover, interesting readers might consider following similar procedures to develop similar approaches in similar contexts.

The development and validation procedure of U-RUCM is shown in Fig. B-13 comprising of two parallel processes: 1) related to the development of U-RUCM mainly conducted by the researchers; 2) validation of uncertainty requirements mainly performed by the industrial partners. At the start of the project (before developing U-RUCM), RUCM was introduced to both industrial partners (i.e., ULMA and FPX) as a means for eliciting and specifying the initial versions of their uncertainty requirements as shown as step *B1* in Fig. B-13. In Fig. Appx-7 of Appendix B, we provide a sanitized example of UCSs capturing uncertainty requirements with an extended RUCM template. This example was documented by our industrial partners. Results of this activity are 20 use cases for each case study, 93 RUCM flows of events (52 for AW and 41 for GS), as shown in column *RUCM Model* of Table B-6. In total, the RUCM model for AW had 229 sentences, while the GS model had 256. About uncertainties specified in the RUCM models, 33 (for AW) and 26 (for GS) sets of steps of flows of events describing alternative scenarios were considered as involving uncertainties. It is important to point it out that at this stage, uncertainty requirements (i.e., *Uncertainty Req.V1*, Fig. B-13) were specified in a coarse-grained manner, which clearly justified the need of developing U-RUCM.

Second, we conducted a questionnaire-based survey to collect data to detail and quantify the identified uncertainties (step *A1*, Fig. B-13). During this non-trivial process, RUCM was deemed adequate to provide initial data, but it captured uncertainty requirements at a coarse-

[16] http://www.u-test.eu/

grained level. The output of this step is the initial version of *U-RUCM V1*, Fig. B-13. The questionnaire was derived from the RUCM models developed by the industrial partners (*Uncertainty Req.V1*, Fig. B-13) and it was designed for each use case specification. As summarized in Table B-5, the first two types of questions were meant for inspecting a UCS from the use case modeling perspective and they are generic; the third to sixth types of questions were proposed with the aim to elicit new uncertainty requirements; the last five types of questions were proposed with the aim to detail already specified uncertainty requirements. To get a concrete idea, we provide in Appendix B a list of questions derived for a particular use case (the original version of which is given inTable. Appx-6, a slightly improved version (refined by researchers with the RUCM editor) of which are provided in Fig. Appx-5 and Fig. Appx-6) as an example.

According to the questionnaire and comments provided by researchers (step *A1*), industrial partners developed the second version of the uncertainty requirements (*Uncertainty Req. V2*), which are specified using *U-RUCM V1*. Subsequently, two onsite workshops, i.e., one for each partner were conducted to further refine the collected requirements, i.e., *Uncertainty Req V2* (step *A2/B3*). The output of the step is *Uncertainty Req. V3*, which is input to the *A3* step for refining *U-RUCM V1* into *U-RUCM V2* and developing and formalizing *Uncertainty Req. V4* (step *A3*). *U-RUCM V2* is the final version presented in this paper, and the current U-RUCM editor was developed based on this version of the U-RUCM methodology, and *Uncertainty Req. V4* is the final version of uncertainty requirements, which is specified with *U-RUCM V2*. The collected results of the evaluation are based on the comparison of *Uncertainty Req. V1* and *Uncertainty Req. V4* as presented in Section 6.3. We also provide a sanitized example of the AW industrial case study in Appendix C for reference, which was a final use case specification specified with the latest version of the U-RUCM tool.

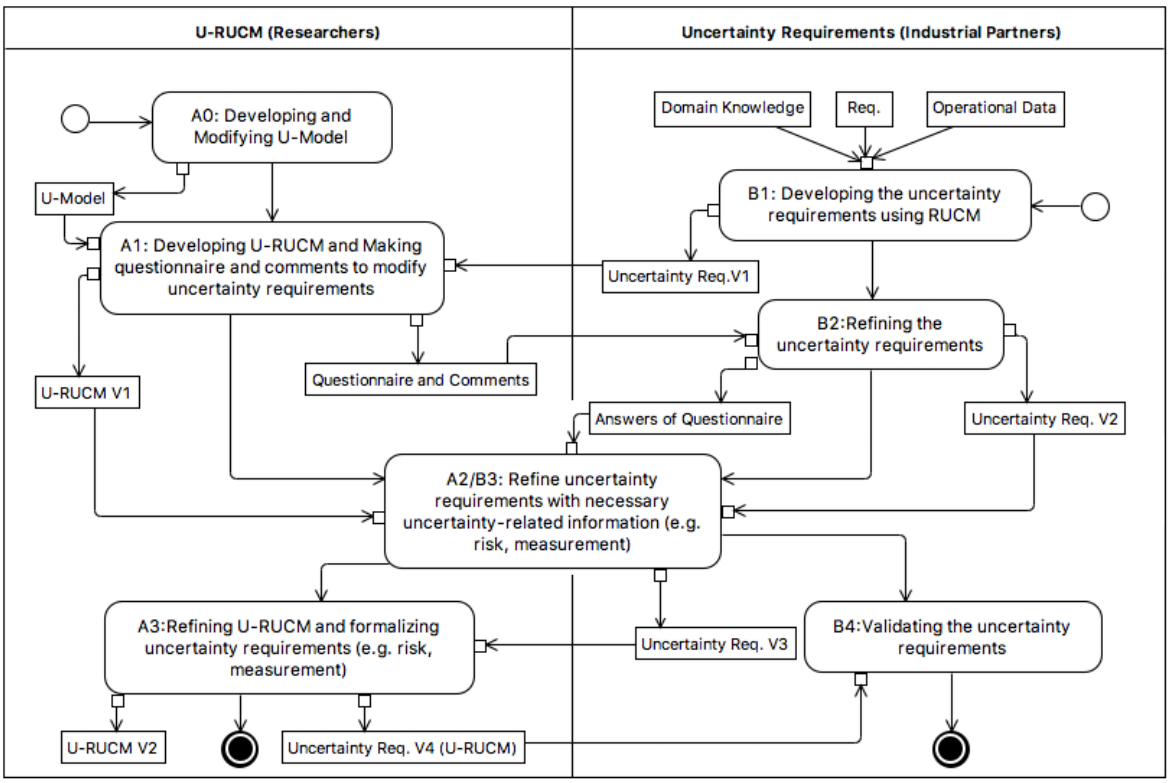


Fig. B-13. Development and Validation of U-RUCM

Table B-5. Design of the questionnaire (A1, Fig. B-13)

#	**Explanation**
1	Inquiry the boundary of the system to define actors in the use case model.
2	Check the completeness of the flow of events of each use case specification.
3	Inquiry the existence of sources related to an actor.
4	Inquiry the existence of potential uncertainties related to system properties or behaviors.
5	Inquiry existence of the potential uncertainties regarding time, nature and human being.
6	Inquiry if a potential uncertainty is valid by checking if it is derived based on system properties or behaviors.
7	Inquiry the completeness of the types of uncertainties defined in U-Model.
8	Inquiry the type of a specified uncertainty.
9	Inquiry the measure and measurement of a specified uncertainty.
10	Inquiry the risk of a specified uncertainty.
11	Inquiry the evidence to support the specified measurement and risk of a specified uncertainty.

6.3 Results

As previously discussed, all the RUCM models developed by the industrial partners were refined using U-RUCM to capture all the identified uncertainties. Descriptive statistics of

the resulted U-RUCM models, i.e., *Uncertainty Req. V1*, are reported in the *Key RUCM Element* columns of Table B-6. The table shows how many elements were added, modified, and removed during the process for the two case studies reported in the *Refinement* column. The elements existing in the final U-RUCM models, i.e., *Uncertainty Req. V4*, are reported in the U-*RUCM Model* column.

Recall that U-RUCM realizes the *Uncertainty* concept of *U-Model* by three concrete means: 1) *NLUncertainty* for belief sentences (Section 4.6.1), 2) *BranchUncertainty* for possible executions of BUCSs from the beginning to end (Section 4.7), 3) uncertainties in flows of events captured via URFS and alternative flows (Section 4.4). We applied these four U-RUCM mechanisms systematically by following the guidelines described in Section 5 and then carefully examined all the specified BUCSs to refine the U-RUCM models further.

As shown in Table B-6, we refactored the design of the RUCM use case model of AW by merging three use cases describing similar scenarios into one, which led to the deletion of 2 use cases (as shown in the table). We also added two use cases to the RUCM model of GS as the result of the refactoring, as these two use cases can be invoked (via the include relationships) by multiple use cases.

Table B-6 also indicates that three uncertainties in the AW RUCM model were removed and four from the GS model. This was because: 1) We optimized the design of the use case model by removing duplicated uncertainties, i.e., one from AW and two from GS. For example, uncertainties describing improper wearing of positioning devices is the same for both indoor and outdoor games; 2) We identified uncertainties from the RUCM models that are indeterminacy sources, which were two for AW and two for GS. For example, the long distance between a positioning device and the satellites is an indeterminacy source (previously identified as an uncertainty), which can lead to the failure of locating the satellites with insufficient resolution nature.

The uncertainty-specific concepts *Indeterminacy Source*, *Alternative Sentence*, and *BranchUncertainty* were only introduced in U-RUCM. Consequently, there were no corresponding elements in the RUCM models. After carefully going through details of the RUCM models using steps described previously, we derived a total of 23 indeterminacy sources for AW and 18 for GS, 45 alternative sentences for AW and 76 for GS. Furthermore,

we discovered 32 instances of *NLUncertainty* for AW and 48 for GS. These turned out to be cases of “unknown knowns” for our industrial partners, that is, tacit knowledge that was not explicit initially. This activity led to the addition of 18 belief flows of events for AW and 31 for GS, 72 new branch uncertainties for AW and 89 for GS, and 43 additional alternative flows with URFS applied for AW and 48 for GS.

In summary, the total numbers of the instances of metaclasses *NLUncertainty* and *BranchUncertainty* of *U-Model*, populated for each of the industrial case studies are 62+72=134 for AW and 70+89=159 for GS. When comparing this with their corresponding “rough” RUCM models, we conclude that, by using U-RUCM, we were able to significantly enhance the extent and precision of modeling uncertainties in requirements (i.e., (134-33)/33=306% for AW and (159-26)/26=512% for GS). This suggests that U-RUCM is an important improvement in dealing with uncertainty in requirements engineering. More specifically, to compare with RUCM, U-RUCM provides a way to elicit, specify and model 1) uncertainty alternative flows describing scenarios that the belief agent lacks confidence about which flow of events occurs given an indeterminacy source, 2) uncertainty alternative actions where the belief agent lacks confidence about which action occurs, given an indeterminacy source, and 3) measurements of uncertainty, such as probability, interval, which are useful when analyzing/reasoning uncertainty.

In our EU project, the U-RUCM models capturing uncertainty requirements were used as the input for developing the test ready models [16, 18] represented as UML class diagrams and state machines using *UncerTum* (see Section 2.1). The test ready models were used to generate test cases, which were then executed successfully in actual systems [19]. There are clear correspondences between the scenarios and uncertainties defined in the test ready models and the ones defined in the U-RUCM models. Note that uncertainty measurements, risk information, among others, were directly migrated from the U-RUCM models to the test ready models. Doing so significantly reduced the effort required to develop the test ready models. Also, throughout the project, the elicited and validated uncertainty requirements were used as the communication medium among the partners.

Table B-6. Descriptive Statistics of the RUCM Models, U-RUCM Models and Refinements

AW						Key U-RUCM Elements	GS					
Key RUCM Elements		**Refinement**			**U-RUCM Model**		**U-RUCM Model**	**Refinement**			**Key RUCM Elements**	
		#A	#M	#R				#A	#M	#R		
Use Case	20	0	20	2	**18**	Use Case	**23**	3	20	0	20	Use Case
FlowOfEvents	52	18	23	4	**66**	(Belief) Flow Of Events	**71**	31	21	1	41	FlowOfEvents
Sentence	229	63	56	16	**276**	(Belief) Sentence	**348**	97	46	5	256	Sentence
		45	0	0	**45**	Alternative Sentence	**76**	76	0	0		
Uncertainty	33	32	18	3(1,2)+	**62**	(NL)Uncertainty	**70**	48	15	4 (2,2)+	26	Uncertainty
		72	0	0	**72**	BranchUncertainty	**89**	89	0	0		
		43		0	**43**	URFS	**48**	48	0	0		
		23	0	0	**23**	Indeterminacy Source	**18**	18	0	0		

#A is the number of added elements, #M is the number of modified elements, and #R is the number of removed elements;
*(n,m)+ -- n is the number of uncertainties removed due to refactoring; m is the number of uncertainties that are changed to indeterminacy source.

6.4 Experience, Lessons Learned, and Future Challenges

This section presents our experience, lessons learned, and future challenges.

Identifying common uncertainties, measurements, and indeterminacy sources. From the GS case study, we noted that human behavior was the key indeterminacy source of uncertainties, due to incorrect interactions with the system. For the AW case study, on the other hand, uncertainties and indeterminacy sources centered mainly on the data communications between control units and their controlled devices. From these types of observations, we can conclude that it is possible in principle to identify common sources, types, and measurements of uncertainties that occur in a given domain or even across domains. This knowledge can be then used to define *reusable* uncertainty specifications and their corresponding behaviors.

Specializing U-RUCM. RUCM can be specialized for different purposes and domains. For example, in another research project, we developed a version of RUCM specifically for real-time systems [31]. In such cases, the standard RUCM template and keywords were extended to allow the specification of time constraints. These are also subject to uncertainty. Based on that, we anticipate that U-RUCM will also need to be extended to specific domains.

Learning about uncertainty by applying U-RUCM. In the past, we experienced that one can learn how to better design use case models by using RUCM. This is why RUCM is used as a teaching method for requirements engineering and software engineering courses both at the undergraduate and graduate levels[17]. Similarly, based on the results of this project, we surmise that it is possible to gain more precise and more direct understanding of both uncertainty and indeterminacy sources by using U-RUCM.

Automated, scalable, and systematic reasoning. For effective coping with uncertainty, automated/semi-automated reasoning about uncertainty and indeterminacy sources can indeed be helpful. This is because, for any non-trivial system, a use case model might be large and may contain a large number of potentially inter-related uncertainties. From our experience during the initial phases of our study when we were not using U-RUCM, we learned that unassisted human reasoning tends to be time-consuming and unsystematic. This is why we chose a more formal approach when developing U-RUCM – via the

[17] http://www.zen-tools.com/rucm/index.html

BeliefUCMeta metamodel – which provides a formal foundation for future, automated reasoning techniques.

Harvesting the benefits of natural language processing techniques. When deriving U-RUCM and performing the two industrial case studies, we noticed that there is an opportunity to further refine *NLUncertainty*, the core concept for representing uncertainties in belief sentences (see Section 4.6). The general idea here is to rely on natural language processing techniques to automatically identify grammatical structures (e.g., Subject), PoSs (e.g., Verb), sentence structures (e.g., Subject-Verb-Object), and/or sentence semantics (e.g., an actor) sends a request to the system in belief sentences. With this, heuristics can be defined to automatically identify potential uncertainties and/or verify already specified ones in belief sentences. For example, a verb of the predicator of a sentence might have an *Occurrence* uncertainty associated with it. A noun being the direct object of a simple sentence might be associated with a *Content* uncertainty.

Reckoning on branch uncertainties. It may be possible to automatically derive values of branch uncertainties (A branch uncertainty indicates a belief agent's confidence in the possibility that the execution of the use case takes this particular branch.) At the very least, branch uncertainties can help to 1) identify critical paths to reduce uncertainties or perform risk analyses (if the postcondition that a branch leads to is considered as the consequence of the branch), 2) verify the overall belief degree that a belief agent has in a belief specification, and 3) derive test cases targeting branches particularly with high uncertainty. This is a possible avenue of further research.

Systematically discovering unknown known indeterminacy sources and uncertainties and transforming them into known unknown uncertainties and known known indeterminacy sources. As the case study results showed, it is possible to systematically identify previously unknown known based on already-specified (known) uncertainties and indeterminacy sources. A systematic methodology (ideally with tool support) can be followed to identify more unknown knowns and currently known unknown uncertainties (e.g., by combining already identified uncertainties that are associated with the same part of system behavior).

Transforming U-RUCM models into other downstream artifacts. To maximize the benefit of U-RUCM models, one possibility is to transform them automatically or semi-

automatically into other artifacts that need to be developed during system development. For example, U-RUCM models can be transformed into UML state machines via the UUP profile (Section 2.1), for supporting MBT of CPSs under uncertainties. This is feasible as RUCM models can be transformed into UML models and test cases (Section 2.3).

7 Related Work

Runtime detection, monitoring, reasoning, and managing of requirements, generally referred to as being *requirements-aware*, is necessary for self-adaptive systems [32], due to inherent changes in operational environments and contextual uncertainties. For this purpose, RELAX [2, 3] – a representative requirements specification and reasoning solution – was proposed to support the development of requirements for dynamically adaptive systems with environmental uncertainty. RELAX consists of a set of keywords, which are classified into *modal*, *temporal*, *ordinal* and *uncertainty* operators. Uncertainty factors aim to indicate where a relaxation of requirements is warranted and, therefore, adaptive behavior is needed. Based on a structured natural language based notation, the authors also proposed a methodology for relaxing requirement statements with the RELAX keywords. Also, RELAX requirements can be formalized using fuzzy logic and reasoning can be performed, when needed.

RELAX has been also integrated with goal-modeling notations (i.e., KAOS [33]) to allow for fuzzy goals [2]. Along the same line, Luciano et al. [4] proposed FLAGS for enabling the specification of adaptive goals, which are of two types: *crisp* goals (specified via linear temporal logic) and *fuzzy* goals (specified using a fuzzy temporal logic). Chen et al. [34] proposed a goal-driven self-optimization framework to handle three different types of uncertainty in goal models: *contribution*, *preference*, and *effect* uncertainty. ReAssuRE was proposed by Welsh et al. [5] to attach *claims* to softgoal contribution links of goal models, with the aim to record the rationale for selecting a goal realization strategy when the optimum choice is uncertain. Later on, Ramirez et al. [35] integrated ReAssuRE with RELAX to assess the validity of claims at runtime, for dealing with environmental uncertainty in dynamic adaptive systems.

Compared to these goal-based approaches, U-RUCM is more generic, as it is not targeting dynamic adaptive systems in particular. Second, U-RUCM is built on a use case modeling

methodology, such that it can naturally facilitate the specification of uncertain alternative scenarios. Furthermore, U-RUCM also enables the specification of various types of uncertainties (e.g., *Time*, *Occurrence*), more precise characterization of uncertainties with information such as *Pattern*, and the ability to quantify uncertainties in different ways (e.g., *Probability*, *Fuzziness*). Currently, U-RUCM has a dedicated template for specifying uncertainty. In the future, it would be useful to investigate using keywords (similar to RELAX) to reduce the effort in specifying uncertainties.

Uncertainty is also considered as an important factor that complicates early requirements definition and decision making. Salay et al. [6] proposed the *MAVO* annotations for modeling uncertainty in requirements engineering models, based on the concept of *partial models* (with their properties checked as *True*, *False* or *Maybe*) [36]. The *MAVO partiality* annotations consist of: *May partiality* (indicating that an element should exist in the model), *Abs partiality* (indicating that an element is a collection of elements), and *Var partiality* (indicating that it is unclear if an element should be merged with others). Famelis and Santosa [7] proposed to use colored Entity-Relation models for explicitly capturing the MAVO partiality, as well as *Points of Uncertainty*, a concept representing a specific decision about which there is uncertainty. Compared to these partial-model solutions, U-RUCM is systematically derived from *U-Model*, and it is integrated with RUCM, which enables the specification of uncertain alternative scenarios.

Uncertainty can hinder organizations in making strategic decisions due to, for example, uncertain stakeholders' goals and priorities. In this context, uncertainty is defined as the lack of knowledge of the consequences of decision alternatives. Letier et al. [37] proposed ways for reasoning about uncertainties to support early requirements and architecture decision analysis. Uncertainties are represented as probability distributions, while Monte-Carlo simulations are used for simulating the impact of alternative decisions. That paper, however, does not provide a solution for uncertainty specification and elicitation. Similarly, Esfahani et al. [38] proposed GuideArch, a framework for the quantitative exploration of the architectural solution space under uncertainty, which is based on fuzzy mathematical methods for reasoning about uncertainty. Although these works support means for reasoning, simulation, and exploration in the presence of uncertainty, they lack the capability specifying and modeling of uncertainty in the context of requirements engineering.

8 Conclusion and Future Work

The impact of uncertainty, which is increasingly being recognized as an inherent and crucial property of non-trivial software-intensive systems (e.g., CPSs), needs to be better understood and addressed explicitly in all phases of system development. In particular, it has to be explored and characterized as much as possible during requirements engineering (e.g., elicitation, specification, and verification). Use case modeling is a well-known and commonly applied requirements specification/modeling method in practice. Specifying uncertainty as part of use case models is therefore particularly useful. In this paper, we described a methodology and a corresponding tool (U-RUCM) for helping practitioners to specify uncertainties in requirements as part of use case models.

U-RUCM originated in the context of the EU project [39], which involved a consortium of nine partners. The initial version of the uncertainty requirements was developed by our industrial partners using the basic RUCM methodology, on which U-RUCM was founded. After refining the RUCM models, by applying the U-RUCM methodology, we successfully identified and specified more than 300% and 500% (previously unknown) uncertainty requirements for the two case studies. The resulting U-RUCM models were used as a reference to develop test ready models for generating executable test cases to test the two industrial applications. As users of U-RUCM ourselves during the evaluation (i.e., case studies), we gained invaluable experience about its use and future potential.

In the future, we plan to enrich the capabilities of U-RUCM from the following aspects: 1) identifying and specifying common uncertainties within and across domains, 2) specializing U-RUCM for the real-time domain, 3) reasoning uncertainties at various levels such as at the sentence level by relying on NL techniques, the structure level of use case specifications by, e.g., analyzing the cause-effect of the sequential order of steps of flows of events, and 4) automated transformation of U-RUCM models into other artifacts such as test cases. We also plan to conduct controlled experiments to evaluate the applicability of U-RUCM and conduct more industrial case studies to understand its potential and limitations better.

Acknowledgment

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, RCN funded Certus SFI and the EU COST action MPM4CPS. Man Zhang is funded by the EU Horizon 2020 funded project (Testing Cyber-Physical Systems under Uncertainty, Project Number: 645463).

References

[1] K. Bittner, and I. Spence, Use Case Modeling, Addison-Wesley, 2003.

[2] B. H. C. Cheng, P. Sawyer, N. Bencomo, and J. Whittle, A Goal-Based Modeling Approach to Develop Requirements of an Adaptive System with Environmental Uncertainty, *Model Driven Engineering Languages and Systems: 12th International Conference, MODELS 2009, Denver, CO, USA, October 4-9, 2009. Proceedings*, A. Schürr and B. Selic, eds., pp. 468-483, Berlin, Heidelberg: Springer Berlin Heidelberg, 2009.

[3] J. Whittle, P. Sawyer, N. Bencomo, B. H. C. Cheng, and J.-M. Bruel, RELAX: a language to address uncertainty in self-adaptive systems requirement, Requirements Engineering, vol. 15, no. 2 (2010) 177-196.

[4] L. Baresi, L. Pasquale, and P. Spoletini, Fuzzy goals for requirements-driven adaptation, in: 2010 18th IEEE International Requirements Engineering Conference. pp. 125-134, 2010.

[5] K. Welsh, P. Sawyer, and N. Bencomo, Towards requirements aware systems: Run-time resolution of design-time assumptions, in: Automated Software Engineering (ASE), 2011 26th IEEE/ACM International Conference on. pp. 560-563, 2011.

[6] R. Salay, M. Chechik, J. Horkoff, and A. D. Sandro, Managing requirements uncertainty with partial models, Requirements Engineering, vol. 18, no. 2 (2013) 107-128, 10.1007/s00766-013-0170-y.

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

[8] T. Yue, L. C. Briand, and Y. Labiche, aToucan: An Automated Framework to Derive UML Analysis Models from Use Case Models, ACM Transactions on Software Engineering and Methodology (TOSEM), vol. 24, no. 3 (2015) 13.

[9] T. Yue, L. C. Briand, and Y. Labiche, Facilitating the transition from use case models to analysis models: Approach and experiments, ACM Transactions on Software Engineering and Methodology (TOSEM), vol. 22, no. 1 (2013) 5.

[10] T. Yue, S. Ali, and M. Zhang, "Applying A Restricted Natural Language Based Test Case Generation Approach in An Industrial Context," *International Symposium on Software Testing and Analysis (ISSTA)*, 2015.

[11] H. Zhang, T. Yue, S. Ali, and C. Liu, Facilitating requirements inspection with search-based selection of diverse use case scenarios, in: proceedings of the 9th EAI International Conference on Bio-inspired Information and Communications Technologies (formerly BIONETICS) on 9th EAI International Conference on Bio-inspired Information and Communications Technologies (formerly BIONETICS). pp. 229-236, 2016.

[12] M. Zhang, T. Yue, S. Ali, H. Zhang, and J. Wu, A Systematic Approach to Automatically Derive Test Cases From Use Cases Specified in Restricted Natural Languages, in: D. Amyot, P. F. i. Casas and G. Mussbacher, eds. 8th System Analysis and Modelling Conference (SAM 2014), Switzerland, 2014.

[13] T. Yue, and S. Ali, Bridging the gap between requirements and aspect state machines to support non-functional testing: industrial case studies, in: European Conference on Modelling Foundations and Applications. pp. 133-145, 2012.

[14] C. Wang, F. Pastore, A. Goknil, L. Briand, and Z. Iqbal, Automatic generation of system test cases from use case specifications, in Proceedings of the 2015 International Symposium on Software Testing and Analysis, Baltimore, MD, USA, 2015, pp. 385-396.

[15] M. Zhang, B. Selic, S. Ali, T. Yue, O. Okariz, and R. Norgren, 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, 2016.

[16] M. Zhang, S. Ali, T. Yue, and R. Norgren, An Integrated Modeling Framework to Facilitate Model-Based Testing of Cyber-Physical Systems under Uncertainty, 2016; https://www.simula.no/publications/integrated-modeling-framework-facilitate-model-based-testing-cyber-physical-systems.

[17] M. Zhang, S. Ali, T. Yue, R. Norgren, and O. Okariz, Uncertainty-Wise Cyber-Physical System test modeling, Software & Systems Modeling (2017), 2017/07/25, 10.1007/s10270-017-0609-6.

[18] M. Zhang, S. Ali, T. Yue, and R. Norgren, Interactively Evolving Test Ready Models with Uncertainty Developed for Testing Cyber-Physical Systems, Technical Report 2016-12, Simula Research Laboratory, 2016; https://www.simula.no/publications/interactively-evolving-test-ready-models-uncertainty-developed-testing-cyber-physical.

[19] M. Zhang, S. Ali, and T. Yue, Uncertainty-wise Test Case Generation and Minimization for Cyber-Physical Systems: A Multi-Objective Search-based Approach, Technical report 2016-13, Simula Research Laboratory, 2016; https://www.simula.no/publications/uncertainty-based-test-case-generation-and-minimization-cyber-physical-systems-multi.

[20] M. Zhang, S. Ali, T. Yue, and R. Norgre, Uncertainty-wise evolution of test ready models, Information and Software Technology (2017), http://dx.doi.org/10.1016/j.infsof.2017.03.003.

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

[22] J. Guo, J. White, G. Wang, J. Li, and Y. Wang, A genetic algorithm for optimized feature selection with resource constraints in software product lines, Journal of Systems and Software, vol. 84, no. 12 (2011) 2208-2221.

[23] OMG, "Meta Object Facility (MOF) Core Specification (Version 2.4.2)," 2014, http://www.omg.org/spec/MOF/2.4.2.

[24] T. Yue, L. Briand, and Y. Labiche, A Use Case Modeling Approach to Facilitate the Transition Towards Analysis Models: Concepts and Empirical Evaluation, in: A. Schürr and B. Selic, eds. Model Driven Engineering Languages and Systems (MODELS 2009), 2009.

[25] J. Wu, S. Ali, T. Yue, J. Tian, and C. Liu, Assessing the Quality of Industrial Avionics Software: An Extensive Empirical Evaluation, Empirical Software Engineering (2016).

[26] T. Yue, H. Zhang, S. Ali, and C. Liu, A Practical Use Case Modeling Approach to Specify Crosscutting Concerns: Industrial Applications, 2015.

[27] "U-RUCM: Specifying Uncertainty in Use Case Models," accessed; http://zen-tools.com/rucm/U_RUCM.html.

[28] G. Zhang, T. Yue, J. Wu, and S. Ali, Zen-RUCM: A Tool for Supporting a Comprehensive and Extensible Use Case Modeling Framework, in: Demos/Posters/StudentResearch@ MoDELS. pp. 41-45, 2013.

[29] "Future Position X," accessed 2017; http://www.fpx.se/.

[30] D. S. Nunes, P. Zhang, and J. S. Silva, A survey on human-in-the-loop applications towards an internet of all, IEEE Communications Surveys & Tutorials, vol. 17, no. 2 (2015) 944-965.

[31] H. Zhang, T. Yue, S. Ali, J. Wu, and C. Liu, A Restricted Natural Language Based Use Case Modeling Methodology for Real-Time Systems, in: 9th Workshop on Modelling in Software Engineering (MiSE'2017), 2017.

[32] P. Sawyer, N. Bencomo, J. Whittle, E. Letier, and A. Finkelstein, Requirements-Aware Systems: A Research Agenda for RE for Self-adaptive Systems, in: 2010 18th IEEE International Requirements Engineering Conference. pp. 95-103, 2010.

[33] A. Van Lamsweerde, Requirements engineering: from system goals to UML models to software specifications, Wiley Publishing, 2009.

[34] B. Chen, X. Peng, Y. Yu, and W. Zhao, Uncertainty handling in goal-driven self-optimization–limiting the negative effect on adaptation, Journal of Systems and Software, vol. 90 (2014) 114-127.

[35] A. J. Ramirez, B. H. C. Cheng, N. Bencomo, and P. Sawyer, Relaxing Claims: Coping with Uncertainty While Evaluating Assumptions at Run Time, *Model Driven Engineering Languages and Systems: 15th International Conference, MODELS 2012, Innsbruck, Austria, September 30–October 5, 2012. Proceedings*, R. B. France, J. Kazmeier, R. Breu and C. Atkinson, eds., pp. 53-69, Berlin, Heidelberg: Springer Berlin Heidelberg, 2012.

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

[37] E. Letier, D. Stefan, and E. T. Barr, Uncertainty, risk, and information value in software requirements and architecture, in: Proceedings of the 36th International Conference on Software Engineering. pp. 883-894, 2014.

[38] N. Esfahani, S. Malek, and K. Razavi, GuideArch: guiding the exploration of architectural solution space under uncertainty, in: 2013 35th International Conference on Software Engineering (ICSE). pp. 43-52, 2013.

[39] S. Ali, and T. Yue, 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, 2015.

[40] P. R. Garvey, and Z. F. Lansdowne, Risk matrix: an approach for identifying, assessing, and ranking program risks, Air Force Journal of Logistics, vol. 22, no. 1 (1998) 18-21.

[41] G. Klir, Facets of systems science, Springer Science & Business Media, 2013.

Paper C

Uncertainty-Wise Cyber-Physical System Test Modeling

Man Zhang, Shaukat Ali, Tao Yue, Roland Norgren and Oscar Okariz

Journal of Software & Systems Modeling (SOSYM).
DOI: 10.1007/s10270-017-0609-6.

Abstract

It is important that a Cyber-Physical System (CPS) deals 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 (UTP) 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* requires18.5% more time to apply stereotypes from UUP on test ready models.

Keywords. Uncertainty; Cyber-Physical System; UML; Model-based Testing

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. C-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. C-1. *UncerTum*, developed by researchers, supports modeling test ready models with known uncertainty based on uncertainty test requirements provided by use case providers (Fig. C-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. C-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. C-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. C-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. C-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. C-1). Note that the focus of this paper is only on *UncerTum*, which is indicated by a dashed line box of Fig. C-1 (i.e., “Scope of the paper”) and the rest is ongoing.

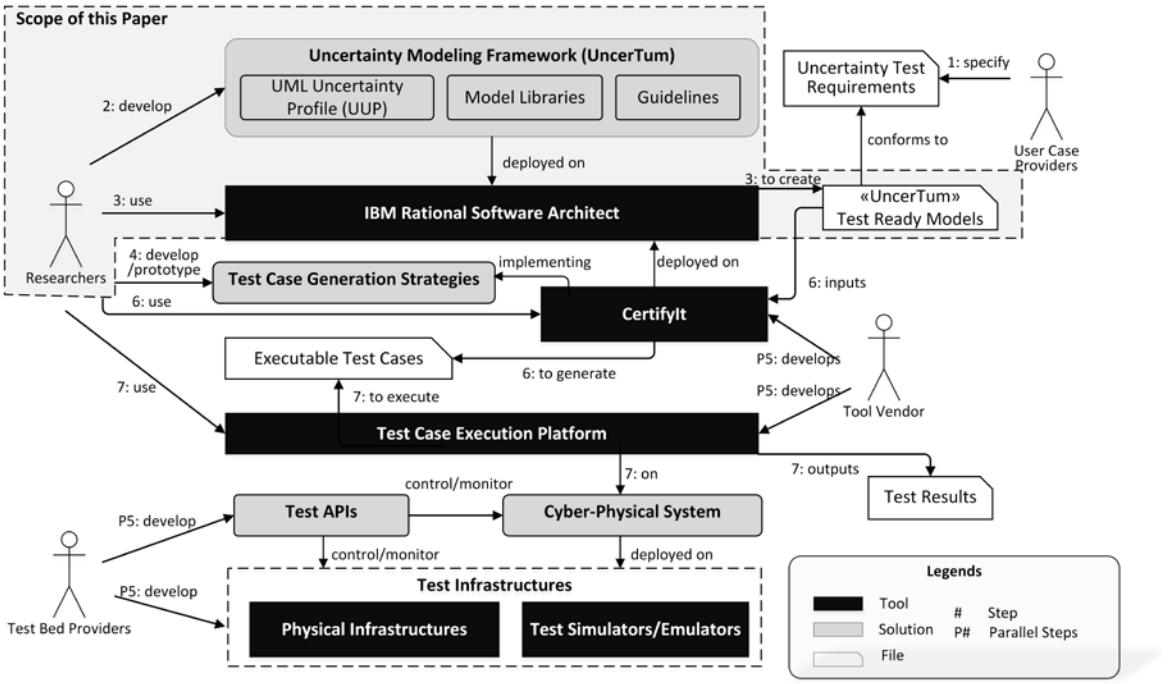


Fig. C-1. Overall Workflow of the U-Test EU Project

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. 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 below. Section 2 presents the background, followed by a running example (Section 3). Section 4 presents the overview of *UncerTum*. Section 5 discusses details of the *UUP* profile and Section 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 Section 10 presents the related work. We conclude the paper in Section 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. C-2. Uncertainty can occur at the following three logical levels [7] (Fig. C-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].

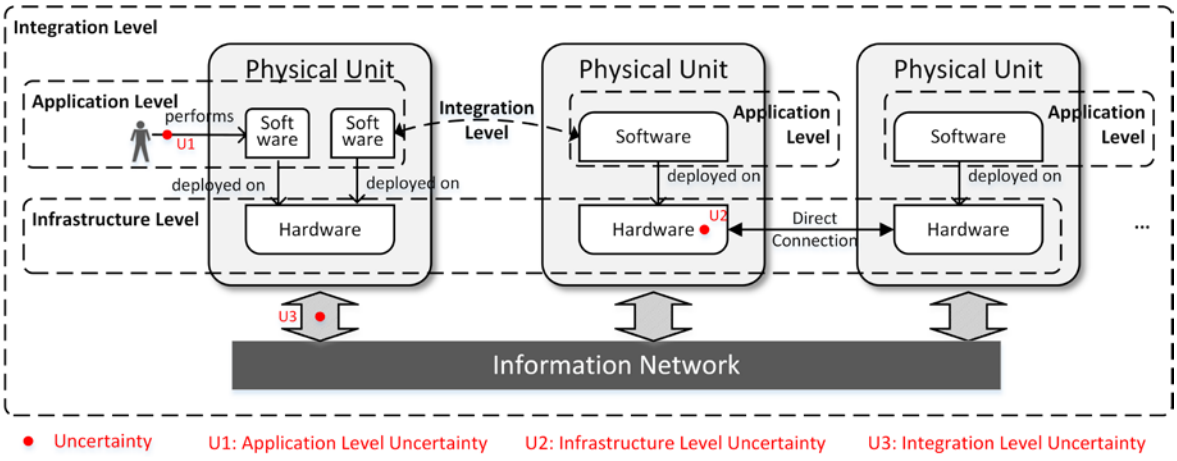


Fig. C-2. Conceptual model of a Cyber-Physical System and the Three Levels

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 (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-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 is 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].

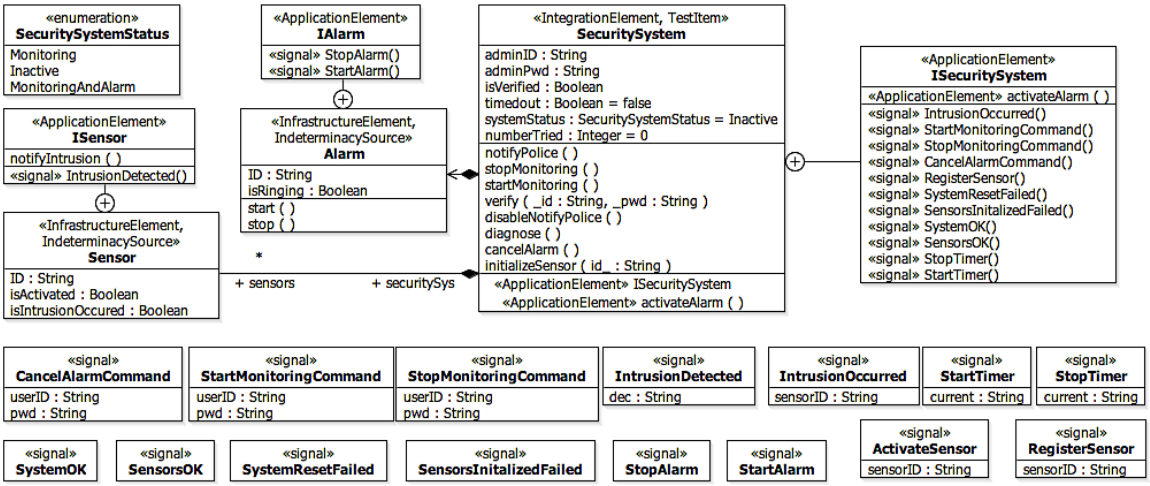
To enable MBT of CPSs under uncertainty, we rely on UTP V.2 to model the testing aspect of test ready models. In our context, only a subset of UTP V.2 was used.

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. C-3), a composite structure diagram (Fig. C-4), and a set of state machines (Fig. C-5, Fig. C-6, and Fig. C-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.

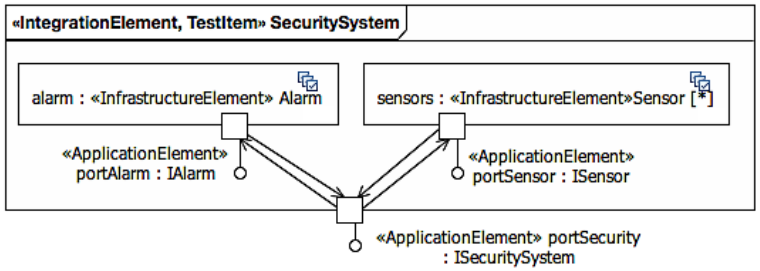
In general, the security system controls and configures *Alarm* and related *Sensor*s through their corresponding interfaces (class diagram in Fig. C-3, detailed explanation in Table C-1). In Fig. C-4, we show a composite structure of the security system. Notice that the alarm

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. C-6).



-- «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. C-3. Class diagram of the Simplified *Security System*



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

Fig. C-4. Composite Structure diagram of the *Security System*

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. C-10) shown in Fig. C-5, Fig. C-6, and Fig. C-7, respectively. The alarm state machine has two states: *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 (Fig. C-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 (Fig. C-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 minutes before notifying the police (*TimerStarted*), and the police is notified (*PoliceNotified*).

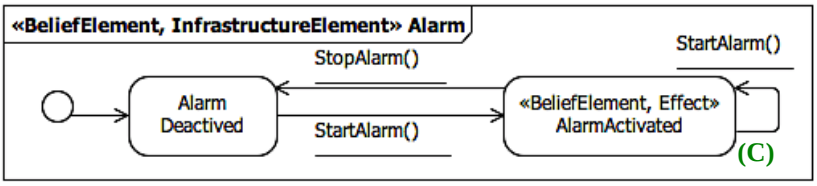


Fig. C-5. State Machine of *Alarm*

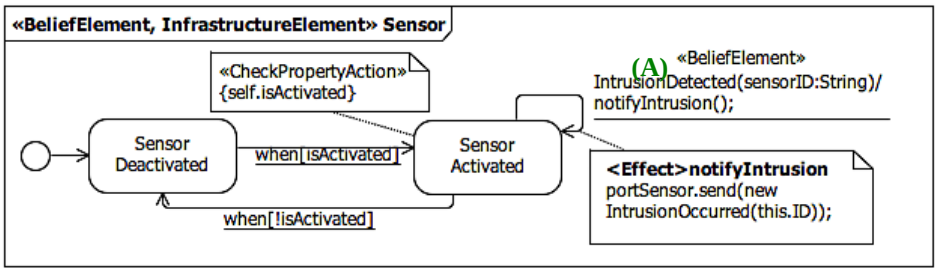


Fig. C-6. State Machine of *Sensor*

These three state machines communicate via signals using the ports defined in the composite structure (Fig. C-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. C-6) once the intrusion is detected via the effect *notifyIntrusion* defined in the self-transition (Fig. C-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. C-7, B.1). In parallel, as one can see from the bottom region (*Timer Control*) of the *MonitoringAndAlarm*

composite state of the system (Fig. C-7), the system sends the *StartAlarm* signal to the *Alarm* state machine via *activateAlarm* (Fig. C-7 and effect* in the Table C-1) and triggers *StartTimer()* when entering the *IntrusionDetected* state (Fig. C-7, B.2), which leads to the transition from *TimerStopped* to *TimerStarted* (Fig. C-7). 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*.

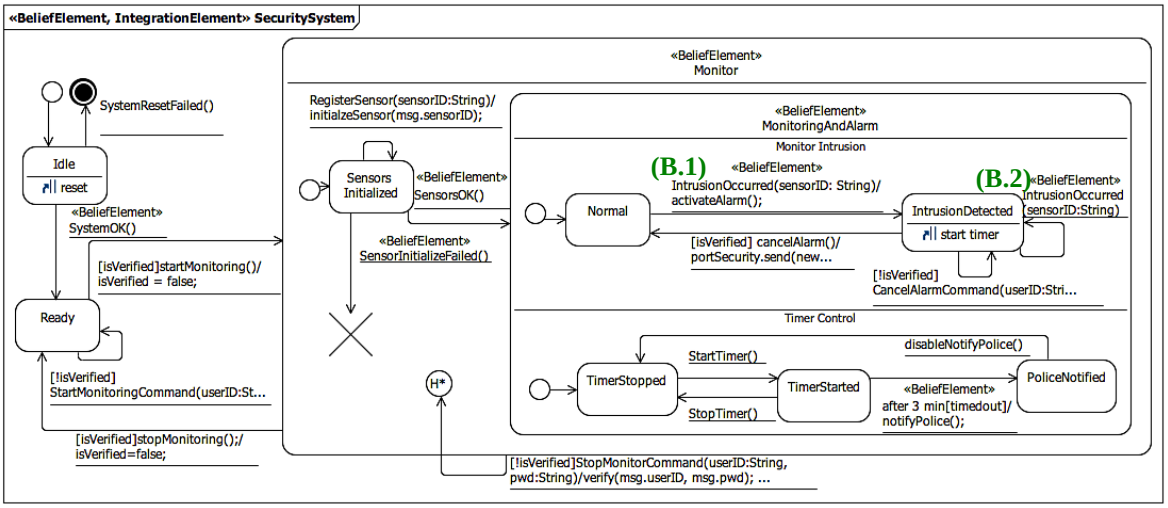


Fig. C-7. State Machine of the Security System

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)

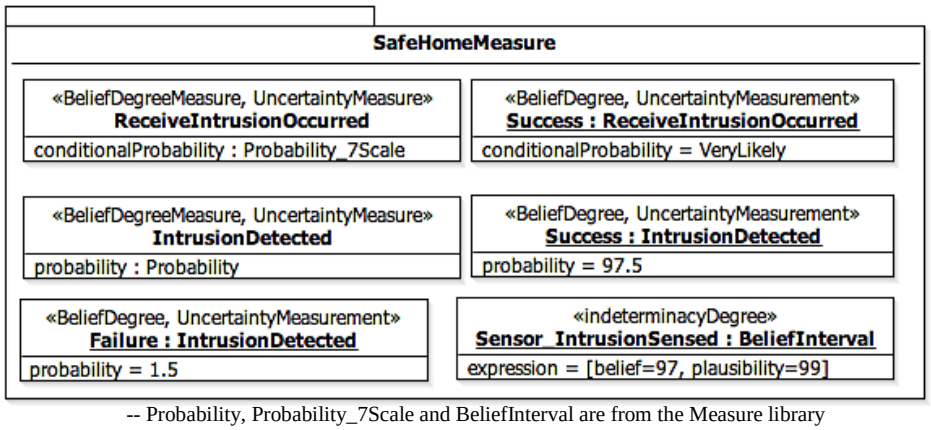
Fig. C-8. StateInvariant (in OCL) of *IntrusionDetected* (B.2)

Fig. C-9 and Fig. C-10 illustrates how we model uncertainty using *UUP/Model Libraries*, whereas Fig. C-3 and Fig. C-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 C-1. An example of using the model libraries is shown in Table C-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 C-1 and *Success:ReceiveIntrsionOccurred* in Fig. C-9). More details are presented in the following sections.

Table C-1. An Example of Classifier *SecuritySystem* using *UncerTum*

Name	SecuritySytem (Fig. C-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. C-4) communicates with *portSensor* of *Sensor*:«ApplicationElement» ISensor communicates with *portAlarm* of *Alarm*: «ApplicationElement» IAlarm	
State Machine	«IntegrationElement, BeliefElement» SecuritySytem (Fig. C-7) **Stereotypes** **agent:** «BeliefAgent» *Man_Simula* (Fig. C-10)	
	Transition	«BeliefElement» *Normal*: State→*IntrusionDetected*: State (Fig. C-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. C-10) **measurement:** -measureInDTViaClass: «BeliefMeasure» *ReceiveIntrusionOccurred* (Fig. C-9) -measurementInVS:<InstanceValue>*VeryLikely* **uncertainty:** -kind: *UncertaintyKind::Occurrence* -referredIndeterminacySourceInClassifier: «IndeterminacySource, InfrastructureElement»*Sensor* -referredCause: «BeliefElement, Cause» *notifyIntrusion* (see Fig. C-6, **A**) -referredEffect: «BeliefElement, Effect» *AlarmActivated* (**C**)

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



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

Fig. C-9. The Example of Modeling Measurement/Measure

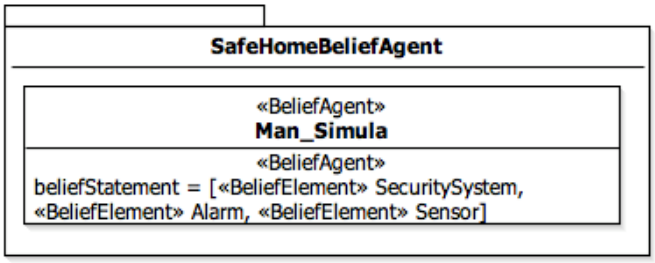


Fig. C-10. The Example of Modeling BeliefAgent

4 Overview of UncerTum

Fig. C-11 shows the overall flow of using *UncerTum* and an overview of *UncerTum* is presented in Fig. C-12. Step 1 in Fig. C-11 is about creating test ready models using the UML profiles (e.g., UUP), model libraries, and guidelines defined in *UncerTum*. Section 5 presents the profiles in detail, Section 6 presents the model libraries, whereas Section 7 presents the guidelines. Step 2 in Fig. C-11 involves developing executable test ready models to support validation of these model based on the guidelines defined in Section 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 Section 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.

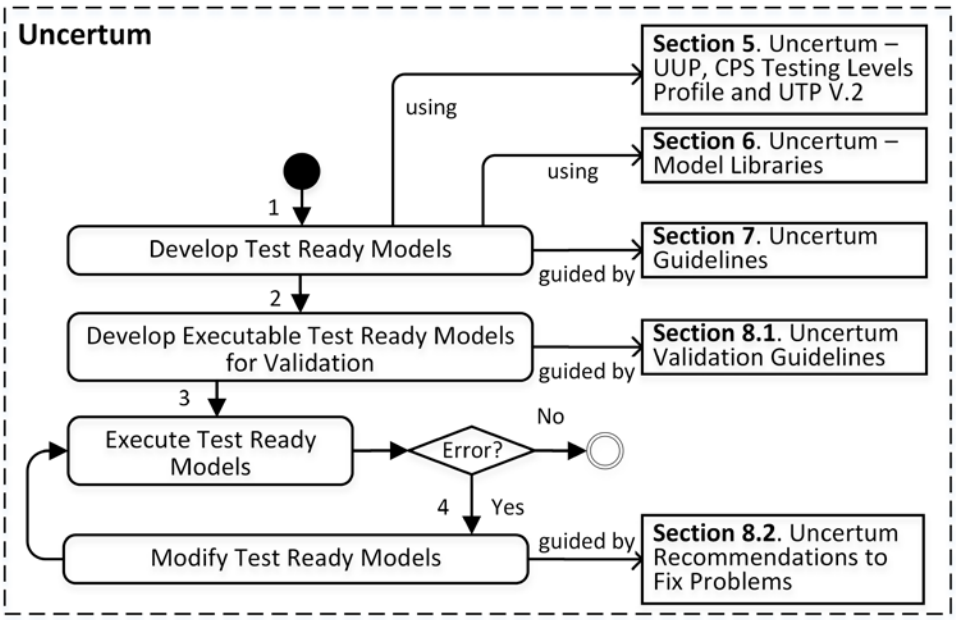


Fig. C-11. Overall Flow of Using *UncerTum*

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. C-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: *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.

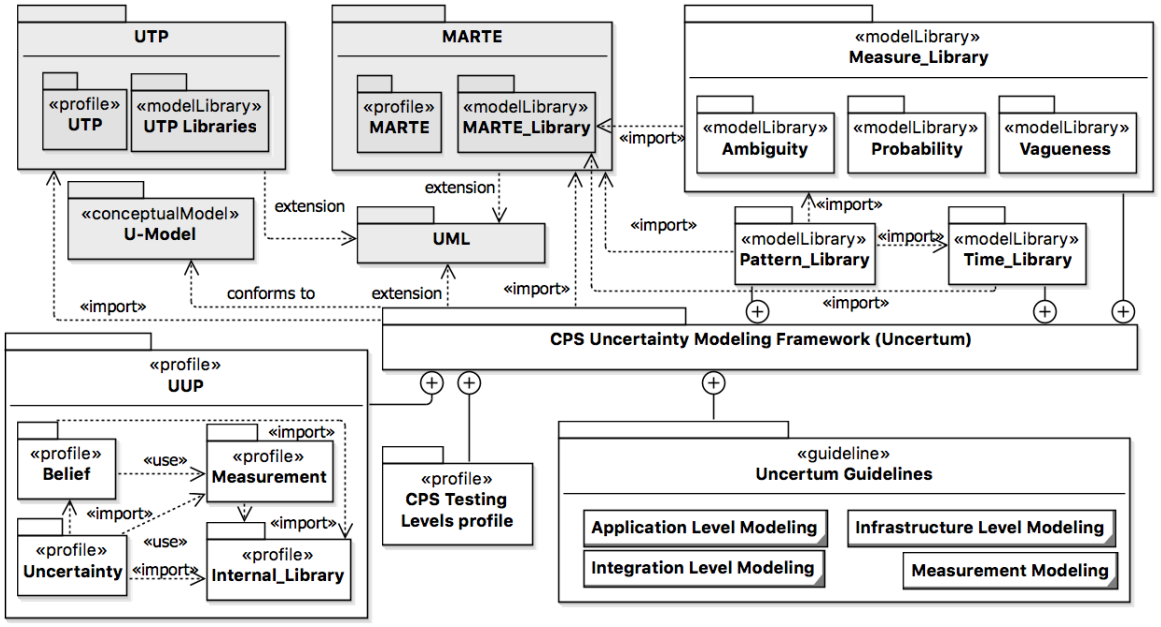


Fig. C-12. Overview of *UncerTum*

5 UUP and CPS Testing Levels Profile

This section presents *UUP*, whose modeling notations are composed of stereotypes and classes for *Belief* (Section 5.1), *Uncertainty*, and *Measurement* (Section 5.2), as shown in Fig. C-13, Fig. C-14 and Fig. C-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 C-2. We also present the *CPS Testing Levels* profile in Section 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].

Table C-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
	«Indeterminacy Source»	BeliefModel::IndeterminacySource

Profile	Stereotype/Class in UUP	Concept in U-Model (Appendix A)
	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

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. C-13, it implements the high-level concepts defined in U-Model:BeliefModel provided in Appendix A.1 (the mapping is provided in Table C-2 and further discussion is provided in Section 9.2.1). As shown in Fig. C-13 and Table C-2, five stereotypes are defined, among which «BeliefElement» is the key stereotype associated to 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.

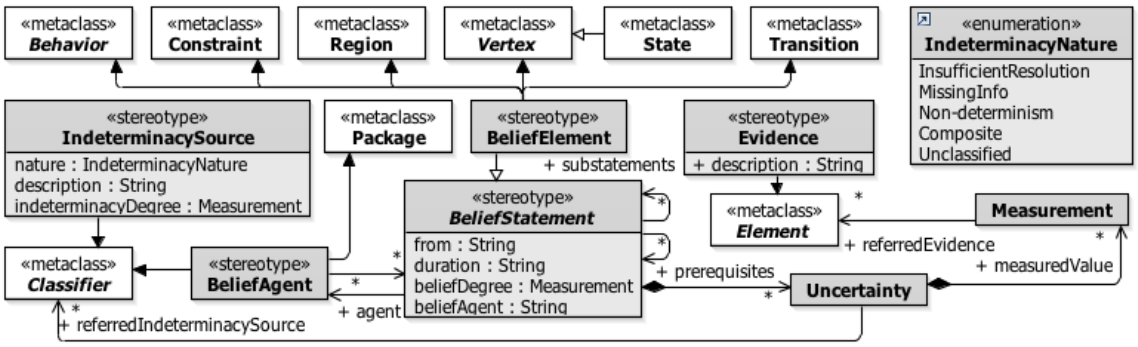


Fig. C-13. The *Belief* Profile

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. C-13) as defined in Appendix A.

The profile also implements OCL constraints defined in U-Model. For example, as shown in Fig. C-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 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 example, the belief agent (Fig. C-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 C-1, «BeliefElement» is applied on the *IntrusionOccurred* transition from *Normal* to *IntrusionDetected* (Fig. C-7, B.1). The belief agent of this belief element is specified as class *Man_Simula* (stereotyped with «BeliefAgent» shown in Fig. C-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 C-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. C-14) and is presented in Fig. C-14. In addition, each *Uncertainty* may have associated measurements that are captured in the *Measurement* part as shown in Fig. C-15. In Fig. C-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 Section 7.

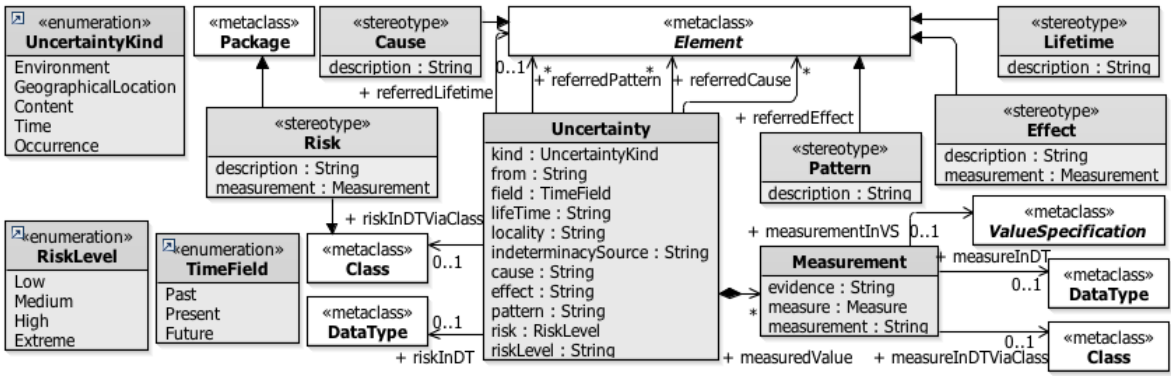


Fig. C-14. The *Uncertainty* Profile

«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.

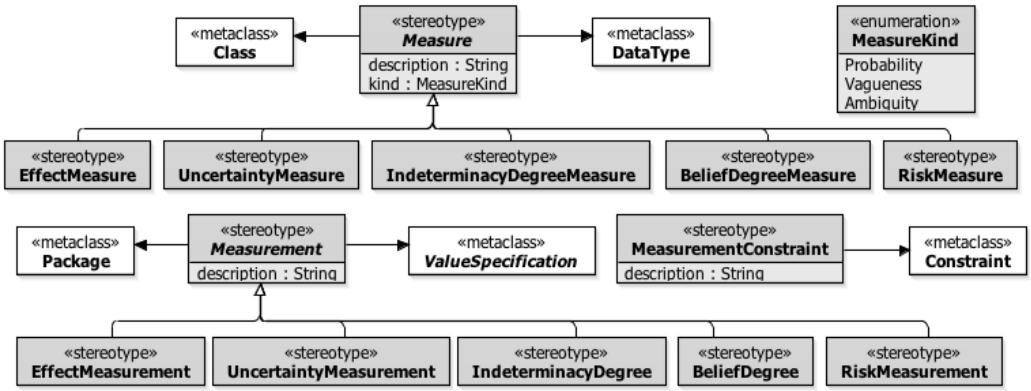


Fig. C-15. The *Measurement* Profile

«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 C-1).

5.3 CPS Testing Levels Profile

We define a small *CPS Testing Levels* profile with the three stereotypes (Fig. C-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. C-3 is stereotyped with «IndeterminacySource» and «InfrastructureElement», meaning that sensors are infrastructure elements. As shown in Fig. C-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. C-3. This is the reason that the composite structure is stereotyped as «IntegrationElement».

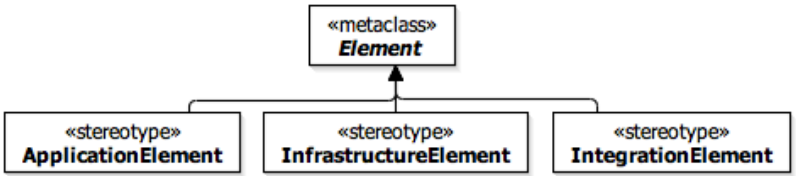


Fig. C-16. The CPS Testing Level Profile

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. C-20),

uncertainty *Measurement*s (corresponding to *Probability*, *Vagueness*, and *Ambiguity* inFig. C-17, Fig. C-18 and Fig. C-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*, thus do 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 (Fig. C-17, Fig. C-18, Fig. C-19, and Table C-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 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. C-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 C-3 for more details and the technical report corresponding to this paper [25]. The concepts of the fuzzy set theory [26] are defined in the *Vagueness* library (Fig. C-18) and Table C-3 lists the relevent references.

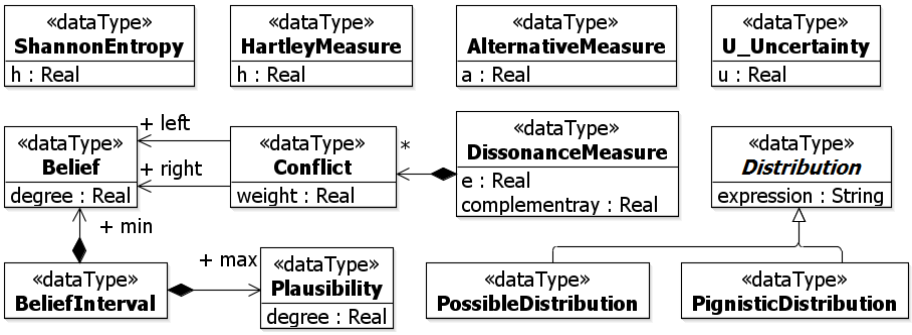


Fig. C-17. The Ambiguity Model Library

Various data types related to the probability are defined in the *Probability* library (Fig. C-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.

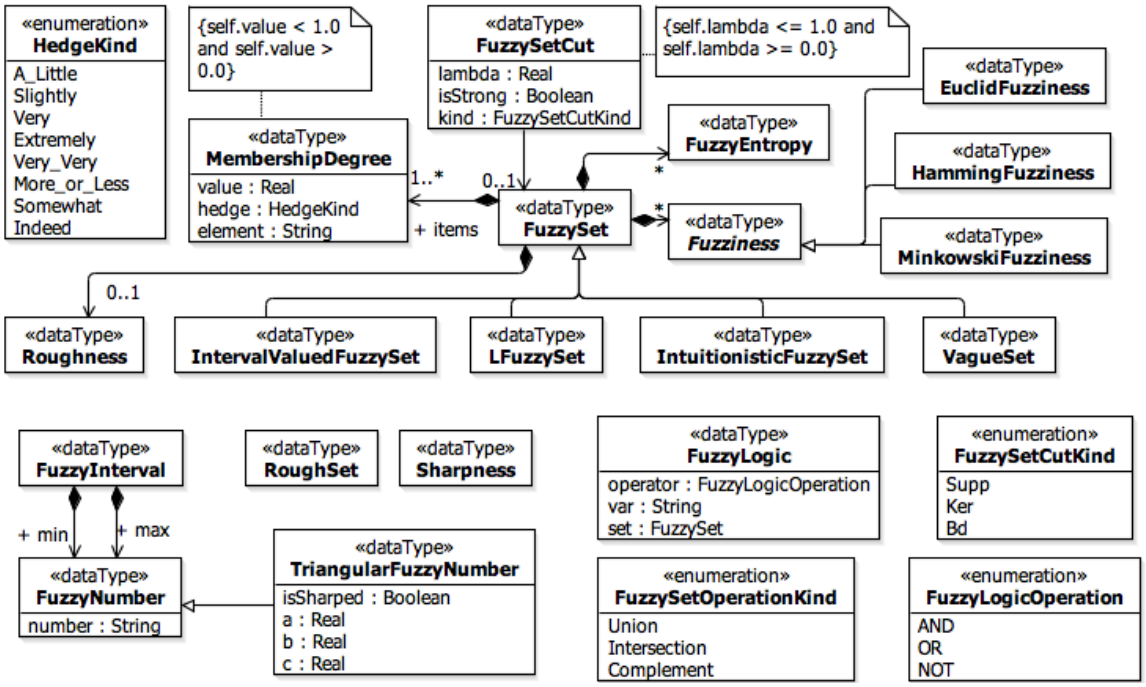


Fig. C-18. The Vagueness Model Library

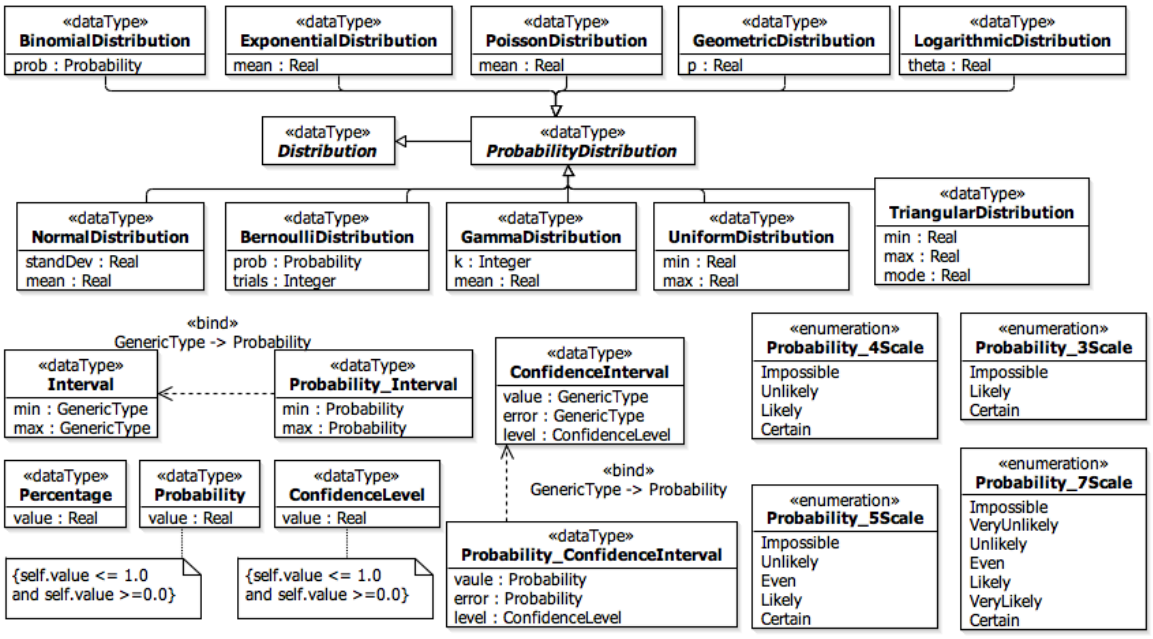


Fig. C-19. The Probability Model Library

For example, as shown in Fig. C-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. C-6), is expressed by *BeliefInterval* [27], which is composed of belief (97%) and plausibility (99%), which are pre-defined in the *Ambiguity* library (Fig. C-17). Further details are provided in the technical report corresponding to this paper [25] for references.

Table C-3. The Main Concepts in Measure Libraries

Measure Library	Concept	Reference
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]

6.2 Pattern Library

This section presents *Pattern Library* shown in Fig. C-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 *maxDuration* 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.

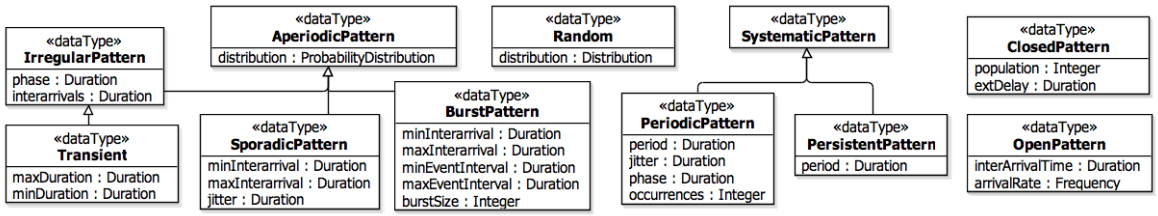


Fig. C-20. Pattern Library

7 UncerTum Modeling Methodology

In this section, we present our modeling methodology for *UncerTum*. 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 the above three level. 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 C-4. We refer to them in the text whenever those are discussed.

Table C-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.

#	Scope	Decision	Justification
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 (Section 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.
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.

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. C-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. C-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 C-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 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.

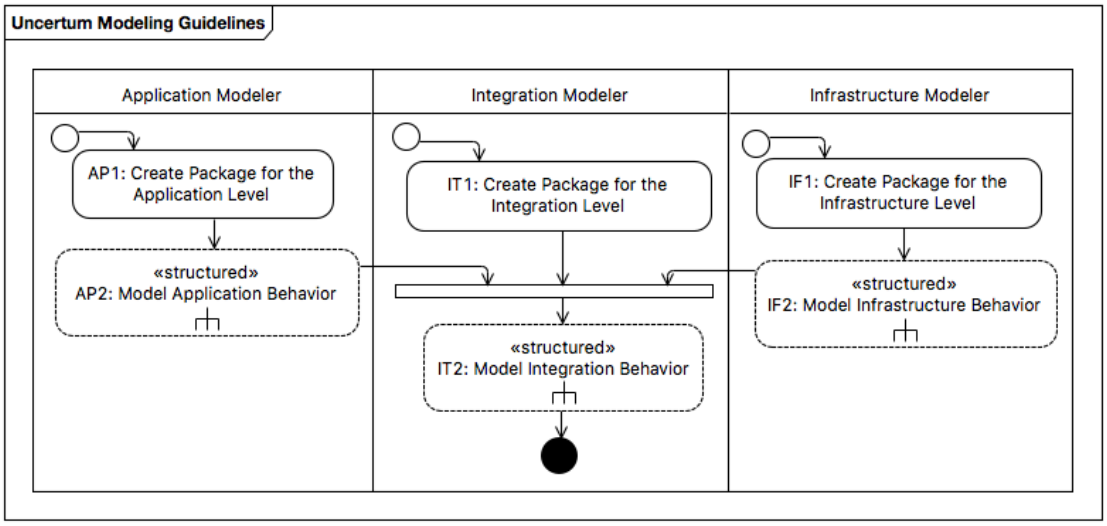


Fig. C-21. The Top Level Guideline

7.2 Application Level Modeling

The application level modeling activities include four sequential steps: creating application level class diagrams (AP2.1, DD2 in Table C-4), creating application level state machines (AP2.2, DD5 in Table C-4), apply 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 C-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 (Section [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 C-4), each state is precisely defined with an OCL constraint specifying its state invariants (DD6 in Table C-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 C-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 (Section 7.4.).

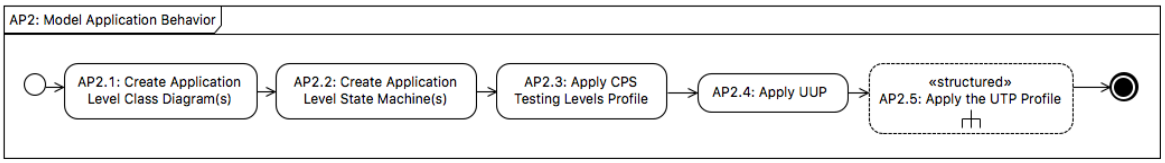


Fig. C-22. Application Level Guidelines

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 UTP (further details in Section 7.4), as shown in Fig. C-23. One difference is that attributes of infrastructure level class diagrams should capture observable infrastructure attributes. For example, an attribute (*isIntrusionOccurred* in Fig. C-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 C-4).

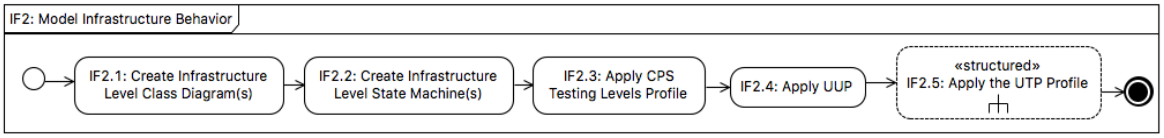


Fig. C-23. Infrastructure Level Guideline

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 Fig. C-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.

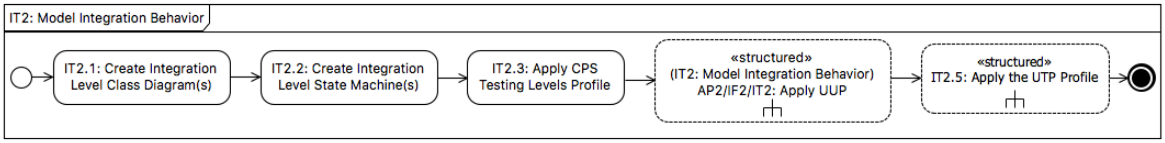


Fig. C-24. Integration Level Guidelines

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 C-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. C-10 (e.g., a phd 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 C-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 Fig. C-25 to Fig. C-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. C-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. C-25).

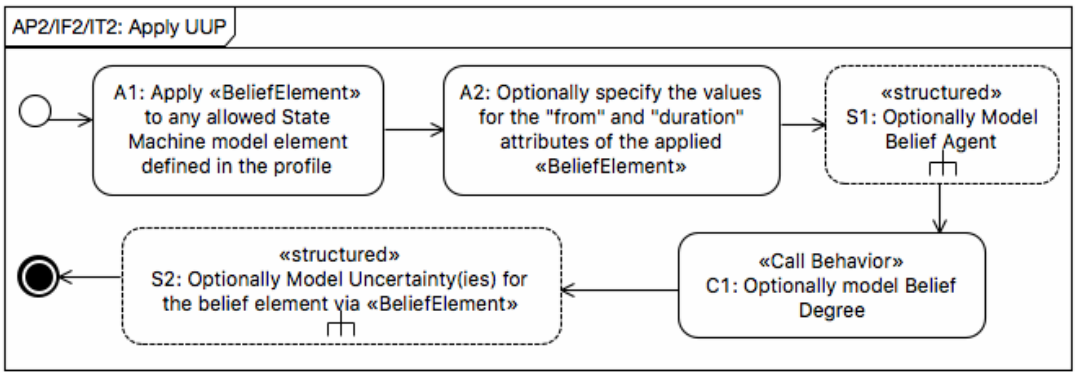


Fig. C-25. Applying UUP

As shown in Fig. C-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 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. C-10.

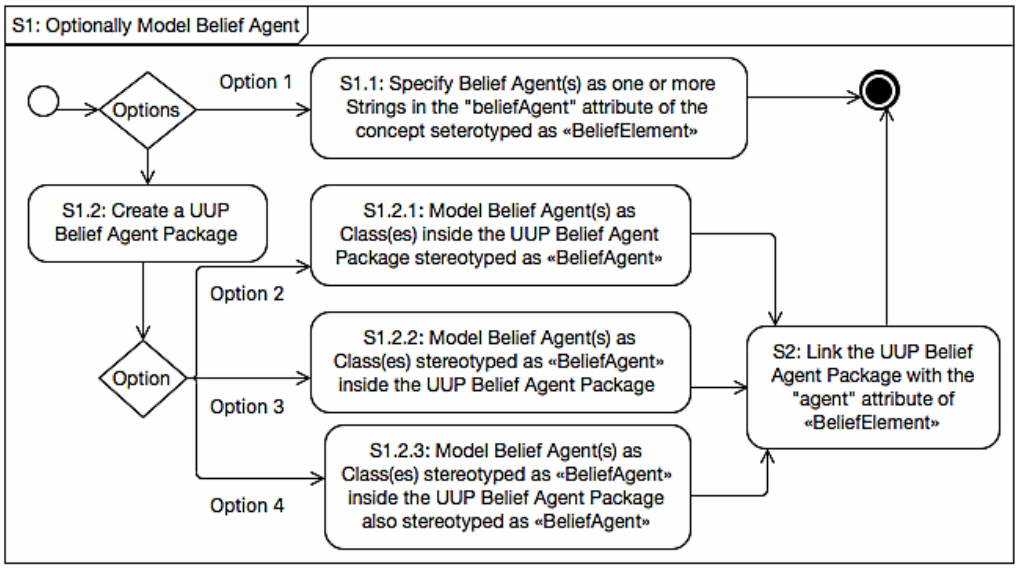


Fig. C-26. Model «BeliefAgent»

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

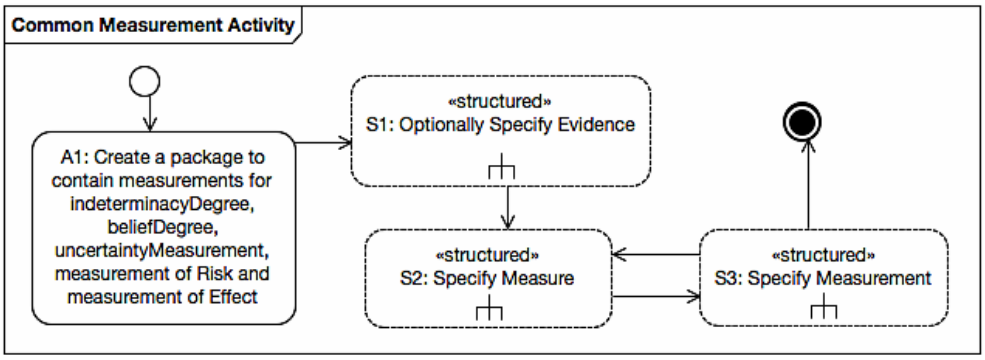


Fig. C-27. Common Measurement Modeling Activity

A. Specify Evidence

As shown in Fig. C-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 representing evidence to the "referredEvidence" attribute of *Measurement* (S1.2.3).

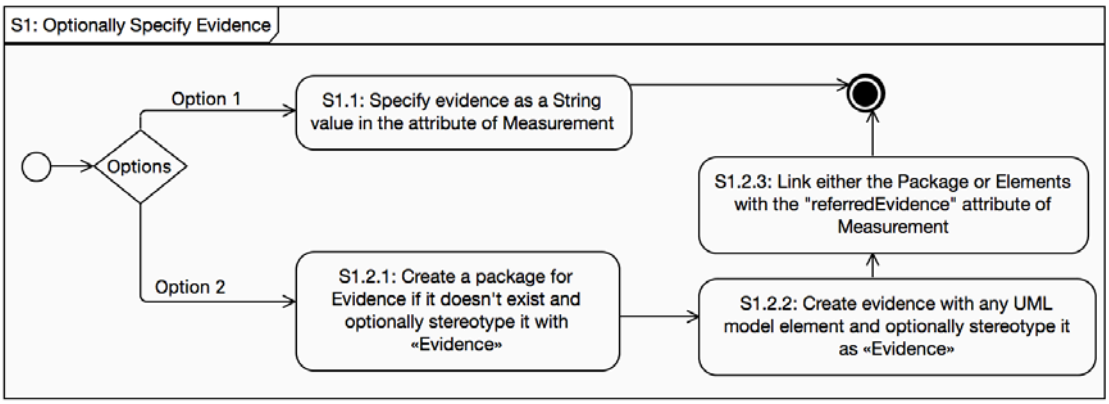


Fig. C-28. Specify Evidence

B. Specify Measure

As shown in Fig. C-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* (Section 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).

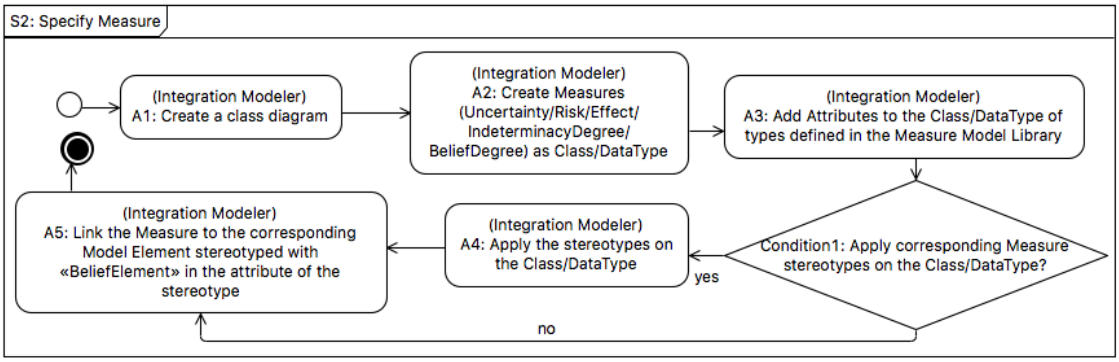


Fig. C-29. Specify Measure

C. Specify Measurement

There are three ways (D12, D13) to specify measurements (in Fig. C-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).

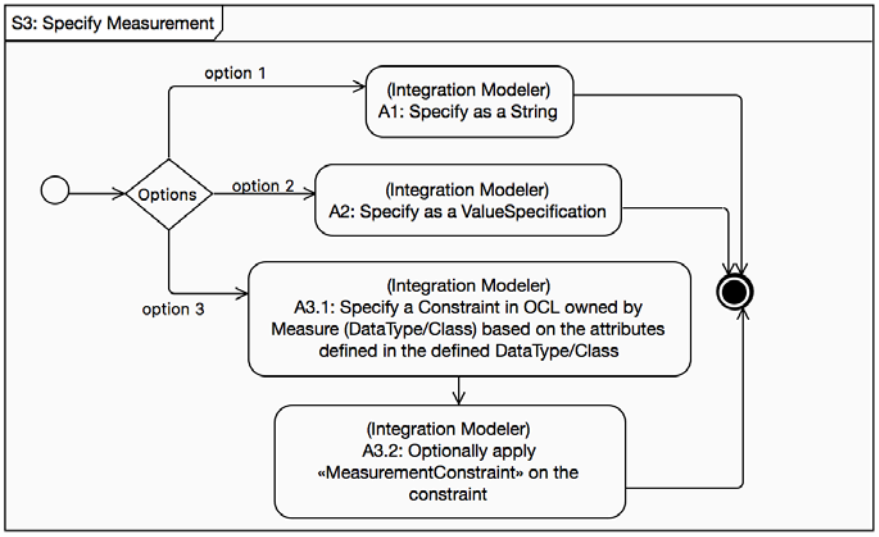


Fig. C-30. Specify Measurement

7.5.2 Uncertainty Modeling

As shown in Fig. C-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*.

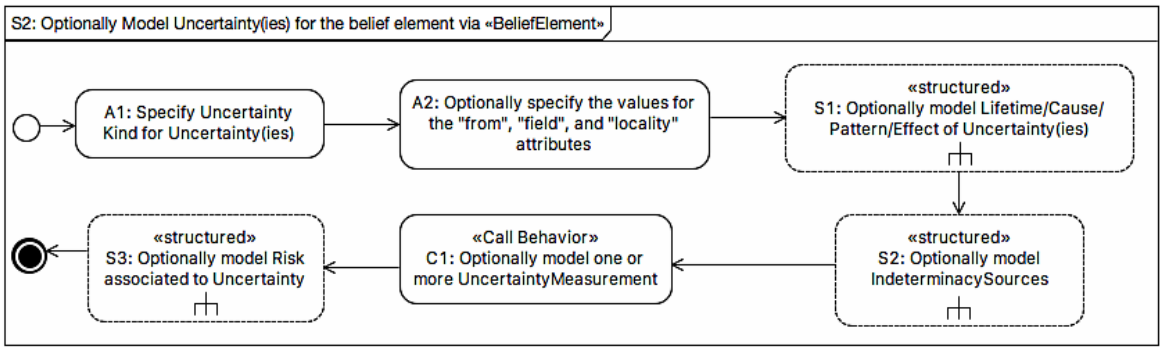


Fig. C-31. Model Uncertainty

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. C-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 (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".

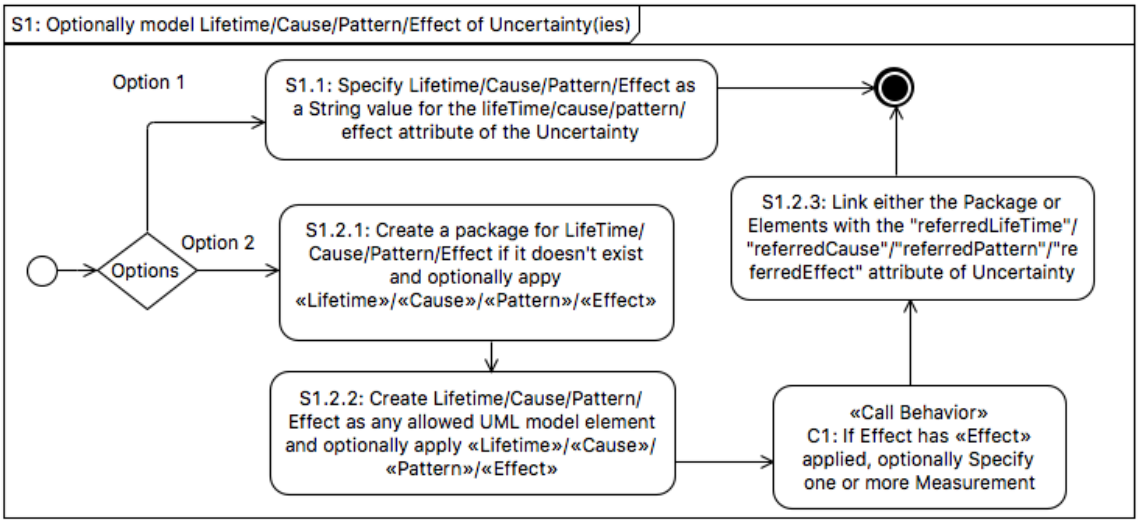


Fig. C-32. Model Lifetime/Cause/Patten/Effect of Uncertainty

B. Model IndeterminacySource

As shown in Fig. C-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*.

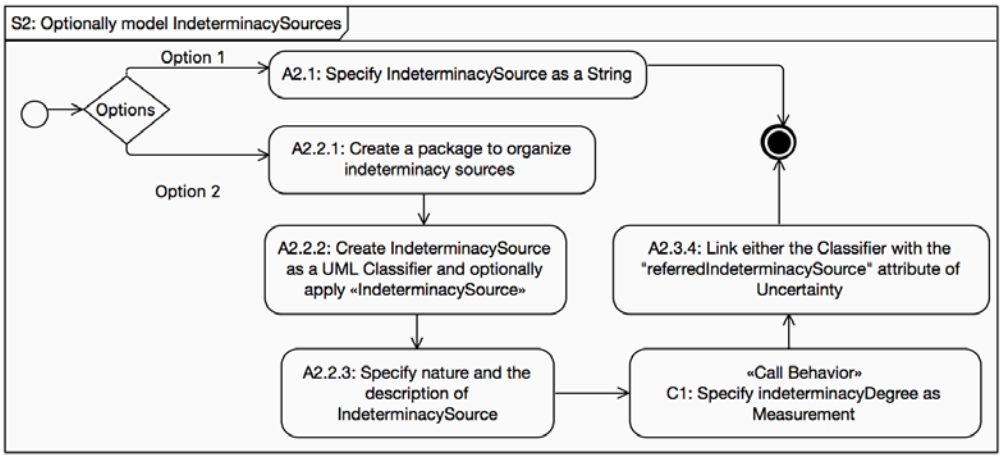


Fig. C-33. Model IndeterminacySource

C. Model Risk

A modeler can optionally associate an uncertainty to *Risk* (D12, D13). As shown in Fig. C-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).

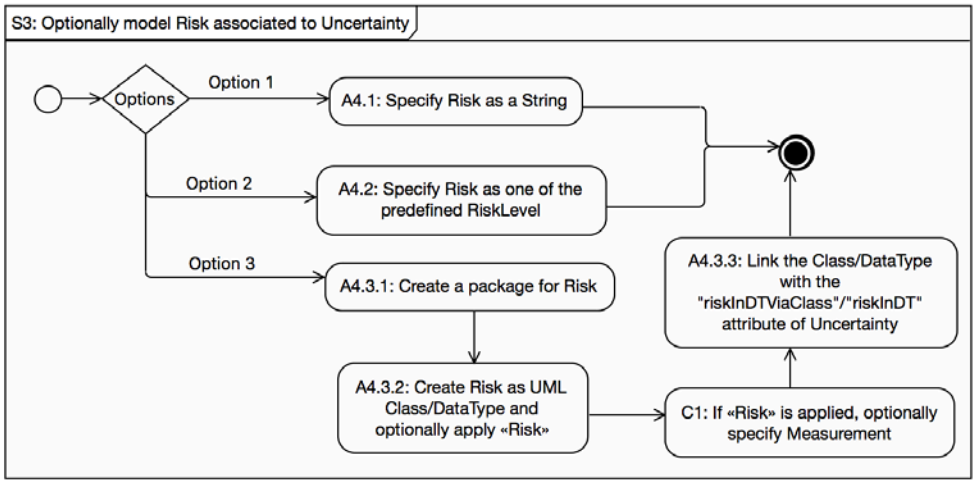


Fig. C-34. Model Risk

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 C-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 Section 1. Further, we provide a set of guidelines as an activity diagram to add UAL code on the test ready models in Section 8.1 and propose a set of recommended actions in Section 8.2, based 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. C-35) to convert test ready models that were created based on the guidelines in the last section into executable models to facilitate validation.

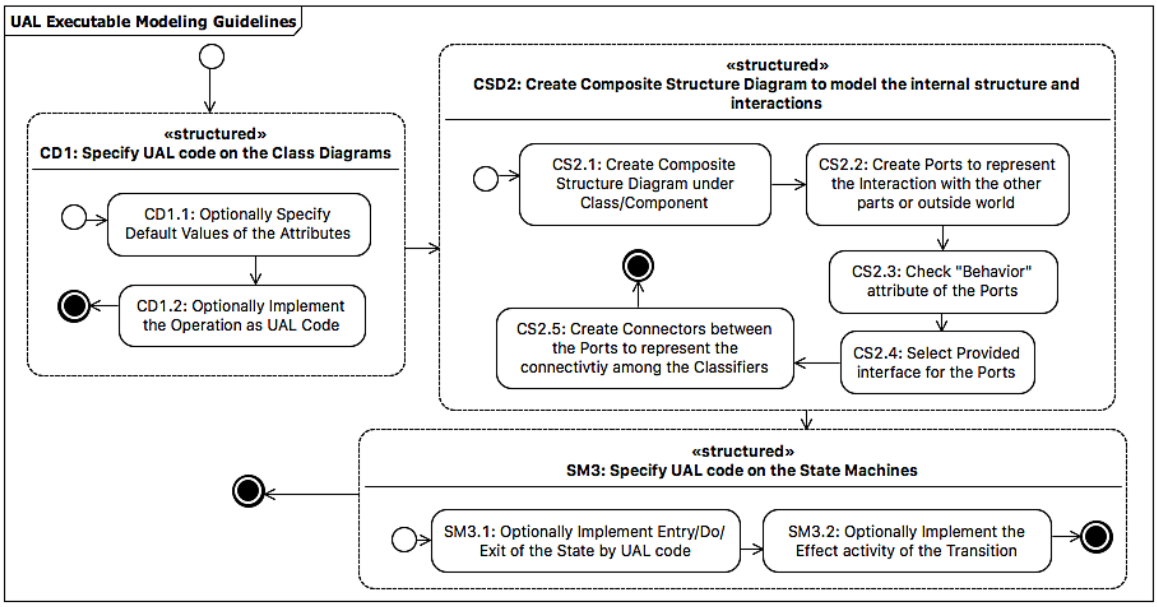


Fig. C-35. Guidelines to Create Executable Test Ready Models

As shown in Fig. C-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. C-3) is *false*, i.e., its default value. 2) As shown in the CSD2 activity in Fig. C-35, a test modeler should create a composite structure diagram (DD10, DD11 in Table C-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. C-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 *IntrusionOccured* signal and other *Signal Receptions* in Fig. C-3. Two connectors between *portSecurity* and *portSensor* (Fig. C-4) are created to enable two-way communications between *SecuritySystem* and *Sensor*. 3) As shown in SM3 in Fig. C-17, 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. C-6 is implemented with UAL as `portSensor.send(new IntrusionOccurred(this.ID))`. Since *portSensor* and *portSecurity* are connected (Fig. C-4) and provided interface *ISecuritySystem* of *portSecurity* has the capability to receive the *IntrusionOccurred* signal (in Fig. C-3), the *B.1/B.2* transition (in Fig. C-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 C-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. C-7) even when it was sent out (O4, Table C-5). One possible reason is that the *IntrusionOccurred Signal Reception* in the *ISecuritySystem* interface of *SecuritySystem* is missing (SA7).

Table C-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

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 Section 9.1, the results are described in Section 9.2, and overall discussion and limitations are presented in Section 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[18] 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 lab 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

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

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. C-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) 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. C-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. C-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 V2 in Fig. C-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.

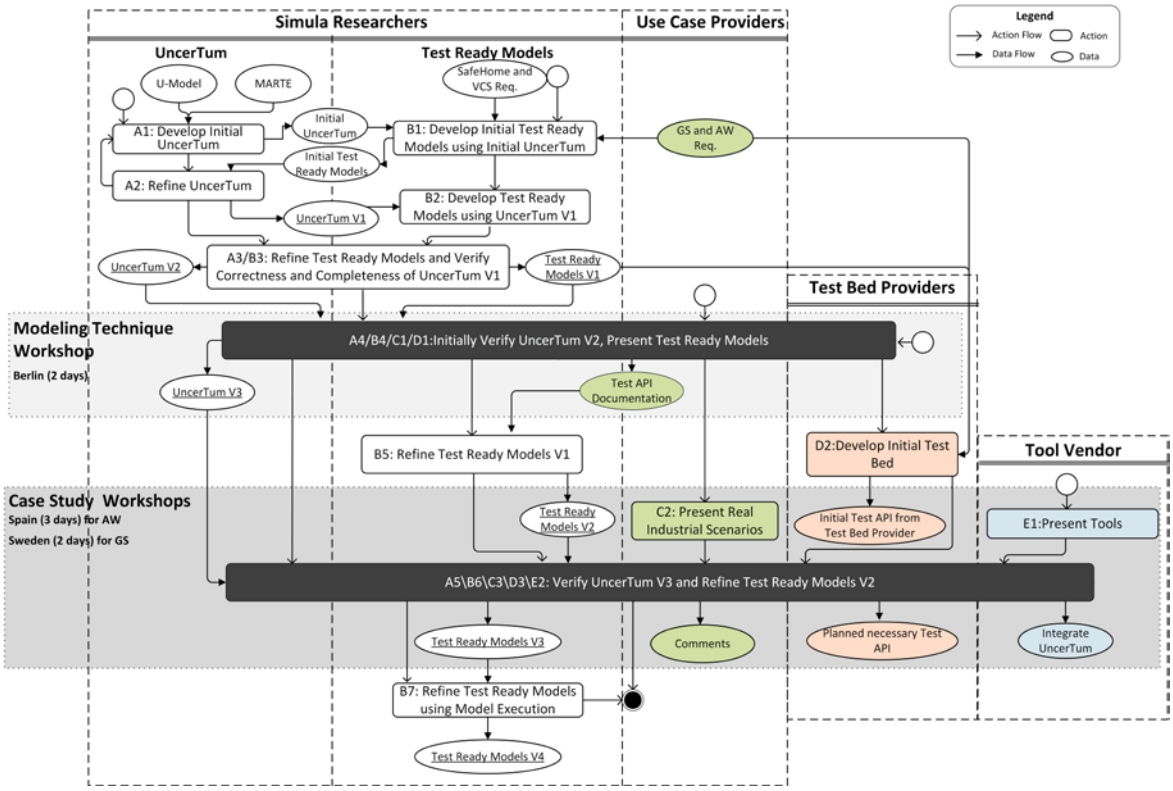


Fig. C-36. Development and Validation of UncerTum and Test Ready Models

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. C-36. Finally, we validated Test Ready Models V3 using IBM RSA Simulation Toolkit (see Section 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 C-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.

Table C-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**

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 C-6 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 the last row of Table C-6, 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 C-6 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.

Table C-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.*

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 Section 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 Section 10), we, therefore, do not have a comparison baseline. Conducting controlled experiments with test modelers could 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 are often time and monetary wise expensive.

As shown in Table C-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 «IndeterminancySource» sensor is applied to *Sensor*, see Fig. C-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 C-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 C-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 C-9, for *SafeHome*, it took the modeler 4.5 hours for modeling the UML class diagrams, whereas additional 0.5 hour was spent on applying *UUP/*Model Libraries. For the 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 C-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.

Table C-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%					

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 Section 8.

Table C-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. 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.

Table C-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

The typical problems identified include: 1) a transition between two states was fired without any event (O7 in Table C-5); 2) after firing a transition the state change did not occur or the state changed to an unexpected one (O1, O2 in Table C-5); 3) failed to send signals across concurrent state machines (O4 in Table C-5); 4) there were no non-deterministic transitions from a state (O8 in Table C-5); 5) unexpected exit, block, or deadlock were observed in a state machine (O1, O9 in Table C-5); 6) unreachable states were discovered (O3 in Table C-5); and 7) a guard condition was always true (O2, O7 in Table C-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. C-24. In the running example, «TestItem» from the *Test Context* package of UTP V.2 was applied on

SecuritySystem (Fig. C-3) and «CheckPropertyAction» from the *Arbitration Specification* package of UTP V.2 was applied to the state invariant of *IntrusionDetected* (Fig. C-7).

Table C-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 C-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 Section 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 C-8). When estimating effort in terms of time, we observed that we needed on average additional 18.5% of time to apply UUP (Table C-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 C-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 to 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 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.

Acknowledgment

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.

References

[1] E. A. Lee, "Cyber physical systems: Design challenges," in Object Oriented Real-Time Distributed Computing (ISORC), 2008 11th IEEE International Symposium on. pp. 363-369.

[2] D. B. Rawat, J. J. Rodrigues, and I. Stojmenovic, *Cyber-physical systems: from theory to practice*: CRC Press, 2015.

[3] S. Sunder, "Foundations for Innovation in Cyber-Physical Systems," in Proceedings of the NIST CPS Workshop, Chicago, IL, USA.

[4] E. Geisberger, and M. Broy, *Living in a networked world: Integrated research agenda Cyber-Physical Systems (agendaCPS)*: Herbert Utz Verlag, 2015.

[5] G. Bammer, and M. Smithson, *Uncertainty and risk: multidisciplinary perspectives*: Routledge, 2012.

[6] D. V. Lindley, *Understanding uncertainty (revised edition)*: John Wiley & Sons, 2014.

[7] M. Zhang, B. Selic, S. Ali, T. Yue, O. Okariz, and R. Norgren, "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] S. Ali, and T. Yue, "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," accessed 2017; http://www.fpx.se/.

[10] "ULMA Handling System," accessed 2017; http://www.ulmahandling.com/en/.

[11] OMG, "UML Profile For MARTE: Modeling And Analysis Of Real-Time Embeded Systems," 2011.

[12] "IBM Rational Software Architect Modeling Tool," accessed 2016; https://www.ibm.com/developerworks/downloads/r/architect/.

[13] "CertifyIt," accessed 2017; http://www.smartesting.com/en/certifyit/.

[14] "Easy Global Market," accessed 2017; http://www.eglobalmark.com/.

[15] "Nordic Med Test," accessed 2017; http://www.nordicmedtest.se/.

[16] "IK4-IKERLAN," accessed 2017; http://www.ikerlan.es/eu/.

[17] "Cisco," accessed 2017; http://www.cisco.com/.

[18] S. Ali, L. C. Briand, and H. Hemmati, “Modeling robustness behavior using aspect-oriented modeling to support robustness testing of industrial systems,” *Software & Systems Modeling*, vol. 11, no. 4, pp. 633-670, 2012.

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

[20] OMG, "UML Testing Profile," 2013.

[21] S. Ali, T. Yue, A. Hoffmann, M. F. Wendland, A. Bagnato, E. Brosse, M. Schacher, and Z. R. Dai, "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," accessed; http://utp.zen-tools.com/.

[23] OMG, "UML Testing Profile," 2016.

[24] IBM, "UML Action Language (UAL)," accessed 2017; https://www.ibm.com/support/knowledgecenter/SS8PJ7_9.6.0/com.ibm.xtools.model.ual.doc/topics/c_umlactionlanguage.html.

[25] M. Zhang, S. Ali, T. Yue, and P. H. Nguyen, *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.

[26] L. A. Zadeh, “Fuzzy sets,” *Information and control*, vol. 8, no. 3, pp. 338-353, 1965.

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

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

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

[30] M. T. Lamata, and S. Moral, “Measures of entropy in the theory of evidence,” *International Journal Of General System*, vol. 14, no. 4, pp. 297-305, 1988.

[31] R. R. Yager, “Entropy and specificity in a mathematical theory of evidence,” *International Journal of General System,* vol. 9, no. 4, pp. 249-260, 1983.

[32] M. Higashi, and G. J. Klir, “Measures of uncertainty and information based on possibility distributions,” *International Journal of General Systems,* vol. 9, no. 1, pp. 43-58, 1982.

[33] L. A. Zadeh, “Fuzzy sets as a basis for a theory of possibility,” *Fuzzy sets and systems,* vol. 1, no. 1, pp. 3-28, 1978.

[34] P. Smets, and R. Kennes, “The transferable belief model,” *Artificial intelligence,* vol. 66, no. 2, pp. 191-234, 1994.

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

[36] B. Kosko, “Fuzzy entropy and conditioning,” *Information sciences,* vol. 40, no. 2, pp. 165-174, 1986.

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

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

[39] Z. Pawlak, “Rough sets,” *International Journal of Computer & Information Sciences,* vol. 11, no. 5, pp. 341-356, 1982.

[40] J. A. Goguen, “L-fuzzy sets,” *Journal of mathematical analysis and applications,* vol. 18, no. 1, pp. 145-174, 1967.

[41] K. Atanassov, and C. Georgiev, “Intuitionistic fuzzy prolog,” *Fuzzy Sets and Systems,* vol. 53, no. 2, pp. 121-128, 1993.

[42] L. A. Zadeh, “The concept of a linguistic variable and its application to approximate reasoning—I,” *Information sciences,* vol. 8, no. 3, pp. 199-249, 1975.

[43] I. Grattan-Guinness, “Fuzzy Membership Mapped onto Intervals and Many-Valued Quantities,” *Mathematical Logic Quarterly,* vol. 22, no. 1, pp. 149-160, 1976.

[44] K. U. Jahn, “Intervall-wertige Mengen,” *Mathematische Nachrichten,* vol. 68, no. 1, pp. 115-132, 1975.

[45] W. L. Gau, and D. J. Buehrer, “Vague sets,” *Systems, Man and Cybernetics, IEEE Transactions on,* vol. 23, no. 2, pp. 610-614, 1993.

[46] A. De Luca, and S. Termini, “A definition of a nonprobabilistic entropy in the setting of fuzzy sets theory,” *Information and control*, vol. 20, no. 4, pp. 301-312, 1972.

[47] W. Feller, *An introduction to probability theory and its applications*: John Wiley & Sons, 2008.

[48] H. Song, D. B. Rawat, S. Jeschke, and C. Brecher, *Cyber-Physical Systems: Foundations, Principles and Applications*: Morgan Kaufmann, 2016.

[49] C. Talcott, "Cyber-Physical Systems and Events," *Software-Intensive Systems and New Computing Paradigms: Challenges and Visions*, M. Wirsing, J.-P. Banâtre, M. Hölzl and A. Rauschmayer, eds., pp. 101-115, Berlin, Heidelberg: Springer Berlin Heidelberg, 2008.

[50] S. Ali, L. C. Briand, M. J.-u. Rehman, H. Asghar, M. Z. Z. Iqbal, and A. Nadeem, “A state-based approach to integration testing based on UML models,” *Inf. Softw. Technol.*, vol. 49, no. 11-12, pp. 1087-1106, 2007.

[51] "Eclipse OCL," accessed 2016; http://www.eclipse.org/modeling/mdt/?project=ocl - ocl.

[52] "Dresden OCL," accessed April, 2016; https://marketplace.eclipse.org/content/dresden-ocl.

[53] "IBM RSA Simulation Toolkit," accessed 2016; http://www-03.ibm.com/software/products/en/ratisoftarchsimutool.

[54] "Use Cases - Industrial Case Studies," accessed 2017; http://www.u-test.eu/use-cases/.

[55] "Certus," accessed 2017; http://certus-sfi.no/.

[56] M. Schneider, and M.-F. Wendland, “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] A. Kerwin, “None Too Solid Medical Ignorance,” *Science Communication*, vol. 15, no. 2, pp. 166-185, 1993.

[58] OMG, "Concrete Syntax For A UML Action Language: Action Language For Foundational UML (ALF)," 2013.

[59] M.-A. Sicilia, and N. Mastorakis, “Extending UML 1. 5 for fuzzy conceptual modeling: An strictlyadditive approach,” *WSEAS Transactions on Systems*, vol. 3, no. 5, pp. 2234-2239, 2004.

[60] Z. Ma, “Fuzzy information modeling with the UML,” *Idea*, 2005.

[61] Z. M. Ma, F. Zhang, and L. Yan, “Fuzzy information modeling in UML class diagram and relational database models,” *Applied Soft Computing*, vol. 11, no. 6, pp. 4236-4245, 2011.

[62] L. Yan, and Z. M. Ma, "Extending nested relational model for fuzzy information modeling," in 2009 WASE International Conference on Information Engineering. pp. 587-590.

[63] Z. M. Ma, F. Zhang, L. Yan, and J. Cheng, “Representing and reasoning on fuzzy UML models: A description logic approach,” *Expert Systems with Applications*, vol. 38, no. 3, pp. 2536-2549, 2011.

[64] F. Zhang, and Z. M. Ma, “Construction of fuzzy ontologies from fuzzy UML models,” *International Journal of Computational Intelligence Systems*, vol. 6, no. 3, pp. 442-472, 2013.

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

[66] H. Motameni, I. Daneshfar, J. Bakhshi, and H. Nematzadeh, “Transforming fuzzy state diagram to fuzzy Petri net,” *Journal of Advances in Computer Research*, vol. 1, no. 1, pp. 29-44, 2010.

[67] V. Grassi, and R. Mirandola, "UML modelling and performance analysis of mobile software architectures," *UML 2001—The Unified Modeling Language. Modeling Languages, Concepts, and Tools*, pp. 209-224: Springer, 2001.

[68] V. Garousi, "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. pp. 92-101.

[69] OMG, "UML Profile For Schedulability, Performance, And Time," 2005.

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

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

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

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

[74] P. R. Garvey, and Z. F. Lansdowne, "Risk matrix: an approach for identifying, assessing, and ranking program risks," *Air Force Journal of Logistics*, vol. 22, no. 1, pp. 18-21, 1998.

[75] G. Klir, *Facets of systems science*: Springer Science & Business Media, 2013.

Paper D

Uncertainty-Wise Evolution of Test Ready Models

Man Zhang, Shaukat Ali, Tao Yue and Roland Norgre

Journal of Information and Software Technology (IST).
DOI: 10.1016/j.infsof.2017.03.003.

Abstract

Context: Cyber-Physical Systems (CPSs), when deployed for operation, are inherently prone to uncertainty. Considering their applications in critical domains (e.g., healthcare), it is important that such CPSs are tested sufficiently, with the explicit consideration of uncertainty. Model-based testing (MBT) involves creating test ready models capturing the expected behavior of a CPS and its operating environment. These test ready models are then used for generating executable test cases. It is, therefore, necessary to develop methods that can continuously evolve, based on real operational data collected during the operation of CPSs, test ready models and uncertainty captured in them, all together termed as *Belief Test Ready Models* (BMs)

Objective: Our objective is to propose a model evolution framework that can interactively improve the quality of BMs, based on operational data. Such BMs are developed by one or more test modelers (*belief agents*) with their assumptions about the expected behavior of a CPS, its expected physical environment, and potential future deployments. Thus, these models explicitly contain subjective uncertainty of the test modelers.

Method: We propose a framework (named as *UncerTolve*) for interactively evolving BMs (specified with extended UML notations) of CPSs with subjective uncertainty developed by test modelers. The key inputs of *UncerTolve* include initial BMs of CPSs with known subjective uncertainty and real data collected from the operation of CPSs. *UncerTolve* has three key features: 1) Validating the syntactic correctness and conformance of BMs against real operational data via model execution, 2) Evolving objective uncertainty measurements of BMs via model execution, and 3) Evolving state invariants (modeling test oracles) and guards of transitions (modeling constraints for test data generation) of BMs with a machine learning technique.

Results: As a proof-of-concept, we evaluated *UncerTolve* with one industrial CPS case study, i.e., GeoSports from the healthcare domain. Using *UncerTolve*, we managed to evolve 51% of belief elements, 18% of states, and 21% of transitions as compared to the initial BM developed in an industrial setting.

Conclusion: *UncerTolve* can successfully evolve model elements of the initial BM, in addition to objective uncertainty measurements using real operational data. The evolved model can be used to generate additional test cases covering evolved model elements and objective uncertainty. These additional test cases can be used to test the current and future deployments of a CPS to ensure that it will handle uncertainty gracefully during its operations.

Keywords. Uncertainty; Belief Model; Belief Test Ready Model; Model Evolution; Model-Based Testing.

1 Introduction

Handling the inherent uncertainty in Cyber-Physical Systems (CPSs) is a well-known challenge, which requires novel approaches for understanding, discovering and modeling uncertainty, and verifying and validating CPSs under uncertainty [5-8]. Typically, a CPS is developed by integrating various physical units (e.g., devices), which are usually black-boxes (with or without the API access) with known and uncertain assumptions on its physical operating environments and deployments. Thus, when testing a CPS, not only assumptions are made about the internal behavior of the CPS, but also its operating environments and deployments. More specifically, when performing model-based testing (MBT), the expected behavior of a CPS is modeled with the explicit consideration of uncertainty, including uncertain behaviors of its physical environments and uncertain deployments (the focus of our previous work [9]). Such models are typically created by one or more test modelers (belief agent(s)) based on his/her/their assumptions about a CPS, its operating environments, and deployments and thus the captured uncertainty is *subjective* to the test modeler(s).

Naturally, these test ready models, named as *Belief Test Ready Models (BMs)* in the rest of the paper, can be continuously evolved based on real operational data (which introduce *objective* uncertainty) of the current deployment of the CPS such that the evolved models can be used to generate new test cases to test future deployments of the CPS with both captured subjective uncertainty and evolved objective uncertainty.

1.1 Challenges and Objectives

Testing is mainly concerned with sending stimulus (via e.g., test APIs) with test data to a CPS and checking the correctness of changes of corresponding states (e.g., test oracles). In the uncertainty-wise MBT context, BMs are the key artifacts for generating executable test cases. Therefore, the quality of BMs is critical for ensuring the quality of generated test cases and consequently the quality of the CPS under various deployments. Hence, the overall scientific challenge is how to ensure the quality of BMs such that they are ready for being used to generate test cases. It is challenging because in the context of uncertainty-wise MBT for CPSs such BMs are complex (e.g., specified in multiple UML state machines) and subjective uncertainty (reflecting test modelers' belief and specified as part of BMs) need to be continuously validated with *evidence* (e.g., real operational data) continuously collected from existing deployments of the CPS.

Correspondingly, our overall objective is to address this challenge by proposing a model evolution framework, called *UncerTolve*, for evolving BMs, with real operational data collected from real CPS applications. This is feasible, as in the context of continuously deploying a CPS for various applications (details in Section 1.2), real operational data can be collected from already deployed applications of the CPS. Collected real operational data are valuable resources to enhance the initial BMs from the perspective of the correctness and completeness, including uncertainty information, test oracles, and test data specifications. Moreover, such a process can be continuous in the sense that as long as there is new operational data available, the BMs can be evolved to accommodate information contained in the data. Evolved BMs will be therefore more complete and correct. Subsequently, testing the CPS for future deployments, based on the evolved BMs, will be much better supported. We provide a clear correspondence between the sub-challenges and sub-objectives in Table D-1.

Table D-1. Sub-challenges and Sub-objectives of *UncerTolve*

Sub-challenges	Sub-objectives
How to ensure the syntactic and semantic correctness of BMs?	**Model Validation**: Validate and update (with proposed heuristics) BMs with real operational data, via model execution.
How to ensure the quality of uncertainty information captured in BMs?	**Derivation of Objective Uncertainty Measurements**: Derive objective uncertainty measurements from real operational data and enhance BMs by integrating them with subjective uncertainty measurements already specified in the BMs.
How to ensure the quality of test oracles	**Inference of Test Oracles/Test Data Specifications**: Abstract

(represented as state invariants) and test data specifications (represented as guard conditions) of BMs?	invariants (both related to test oracles and test data specifications) from real operational data, by relying on existing dynamic invariant inference techniques.
How to achieve the above sub-challenges in an integrated manner?	**Methodologies/Heuristics/Process**: Define methodologies and heuristics on how to update BMs. Suggest a practical process that integrates model validation, objective uncertainty measurement derivation, and test oracles and test data specification inferences, based on real operational data and model execution.

1.2 Context, Scope and Overview

In the context of an EU project [10], we are developing a model-based and search-based framework for testing CPSs under *known* and *unknown* uncertainties to assure that CPSs deal with uncertainty during their operation and do not harm anyone or anything. Evolving BMs in a systematic manner for preparing them for enabling the generation of executable test cases is one of the key components of the model-based and search-based framework.

The overall context is presented in Fig. D-1, where *UncerTum* [9] is a UML-based, uncertainty modeling framework for constructing BMs, and *UncerTest* [11] is a model-based and search based test case generation and minimization framework. *UncerTolve* (with its key features, inputs and outputs indicated as white boxes in Fig. D-1) is the framework we propose in the paper for evolving BMs developed with *UncerTum*. Evolved BMs are input for *UncerTest* to generate executable test cases.

A CPS may be deployed to more than one applications of the same or different application domains. For example, as discussed in [12], in the avionics industry, multiple system instances (i.e., multiple deployments) of the same CPS type can be deployed to achieve a common goal. In the context of our project, the industrial CPS of GeoSports[19] can be deployed for a variety of sports including Bandy and Ice Hockey. Each application corresponds to a unique deployment, denoted as D_1, D_2, ..D_n. *UncerTolve* evolves BMs developed for a CPS with real operational data collected from available deployments of the CPS. Test cases generated using *UncerTest* from the BM evolved with *UncerTolve* can be used to test both existing deployments (D_1, D_2, …D_n) and new ones (Fig. D-1). Note that the process is naturally iterative as the process of introducing new deployments, collecting real

[19] http://www.u-test.eu/use-cases/

operational data, based on which the BM is updated, testing new deployments based on the evolved BMs, are all iterative.

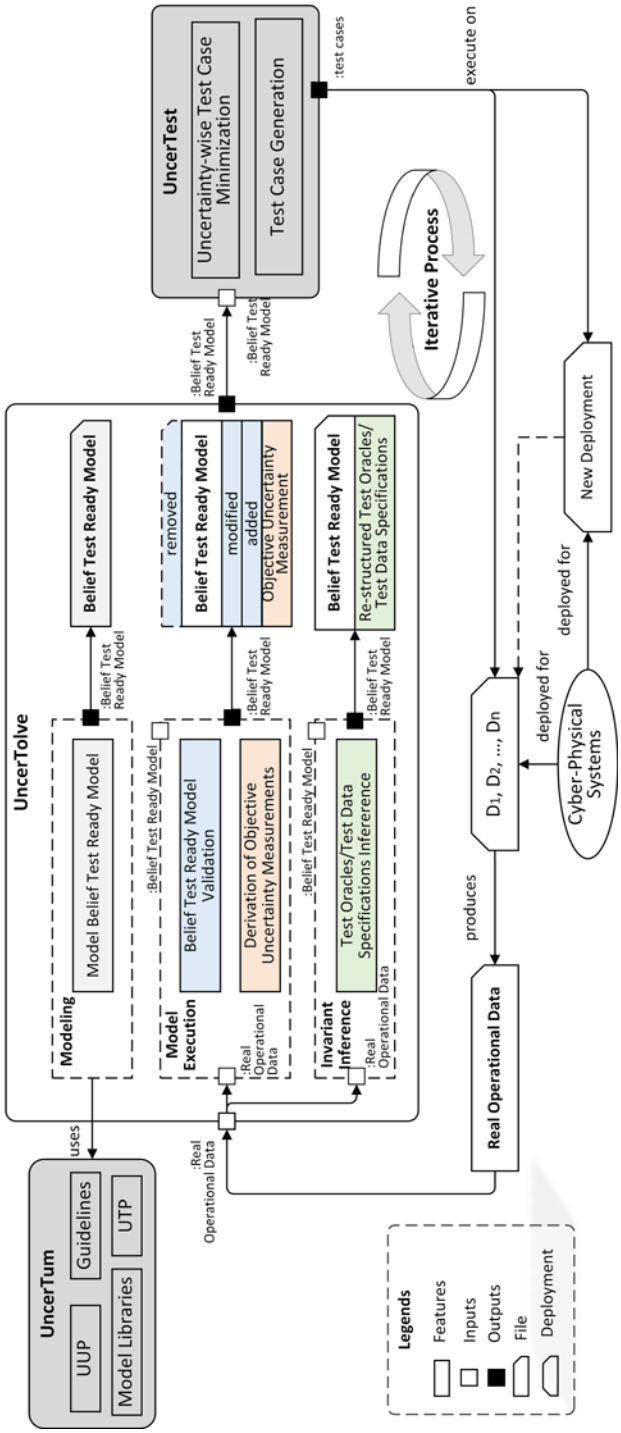


Fig. D-1. The Overall Context and Scope of *UncerTolve*

UncerTolve consists of three activities (i.e., Modeling, Model Execution, and Invariant Inference) and four components (denoted with different colors in the *UncerTolve* box), as shown in Fig. D-1.

The kickoff activity of *UncerTolve* is about modeling BM. We develop the initial BM for a CPS, specified with the Unified Modeling Language (UML) [13]. Such a UML model includes composite structure diagrams, class diagrams, constraints specified in the Object Constraint Language (OCL) [14], and state machines capturing testing interfaces and behaviors of the *application*, *infrastructure*, and *integration* levels of the CPS [9, 15]. *UncerTolve* relies on *UncerTum* [9] to explicitly model known and subjective uncertainties specified by modelers (i.e., belief agents [15]), as part of the initial BM. *UncerTum* consists of the UML Uncertainty Profile (UUP) [9] and a set of model libraries and utilizes the UML Testing Profile (UTP) V.2 [16]. To enable model execution, as part of the *UncerTolve* framework, in this paper, we also propose a modeling methodology (which extends *UncerTum*) particularly for the purpose of developing executable BMs.

The second activity is to execute BMs with real operational data. This activity involves two components: validation of BM and derivation of objective uncertainty measurements. The initial BM created in the first activity is executed to validate their syntactic correctness and conformance against real operational data. Missing or incorrect model elements might be identified during the model execution process and therefore the initial BM can be updated accordingly, based on a set of heuristics newly defined as part of *UncerTolve*. Through model execution with real operational data, *objective* uncertainty measurements can also be obtained. During this activity, existing model elements in the initial BM can be removed or modified, and new ones can be added. Obtained objective uncertainty measurements can also be appended to the BM.

In the third activity is about inferring test oracles (i.e., state invariants) and test data specifications (guard conditions) with real operational data using dynamic invariant inference techniques [4, 17, 18]. In this paper, we apply one solution, Daikon [4], which produces a set of invariants (corresponding to test oracles and test data specifications) with an implemented machine learning technique. These invariants are then merged with OCL constraints specified as part of the BM, based on newly defined heuristics, which therefore leads to another round of the updating of the BM, i.e., restructuring test oracles and test data

specifications. Numerous techniques (e.g., aka automata learning [19], data mining [20, 21]) have been proposed in the literature in the field of automated inferences of various types of information (e.g., properties, protocols, interfaces, specifications) from programs. Although, our work relies on an existing work, i.e., Daikon, our work differentiates itself from the existing works in terms of the core challenge it tackles, i.e., evolving BMs with both subjective and objective uncertainty to eventually support MBT of CPSs under *known* and *evolved* uncertainties discovered based on real operational data.

Note that the modeling activity of *UncerTolve* is the foundation of the other two activities. The other two activities are independent to each other, although we recommend to apply them sequentially as doing so will improve the overall quality of evolved BMs and this is also how our industrial case study was conducted. In summary, theoretically, the output of each activity can be used as the input to *UncerTest*; however, sequentially applying model execution and invariant inference are strongly recommended in practice for ensuring the quality of delivered BMs. This is especially important when BMs are complex, which is quite common in industrial settings.

1.3 Contributions

UncerTolve evolves BMs specified with *UncerTum* [9], which are essentially stereotyped UML class diagrams, composite structure diagrams, and state machines, and therefore contain richer information than a typical specification representation (e.g., Finite State Machines (FSMs)) that can be inferred with existing techniques (e.g., [19, 22]).

Distinguishing itself from existing works, *UncerTolve* takes into account both subjective uncertainty information specified as *belief elements* of the BM and objective uncertainty information derived from real operational data and evolves them as part of the integrated BM evolving process.

Similar to some existing dynamic inference approaches (e.g., [1-3]), *UncerTolve* uses a machine learning technique implemented in Daikon to dynamically infer state invariants (modeling test oracles) and guard conditions (modeling test data specification) of UML state machines. However, *UncerTolve* relies on real operational data collected from real applications of CPSs, instead of execution traces of programs. Note that *UncerTolve* aims to evolve BMs developed for CPSs and therefore existing approaches relying on execution

traces of programs cannot be applied or at least cannot be directly applied without adaptation for the CPS context.

In conclusion, we summarize the key contributions of *UncerTolve* as below:

1 *UncerTolve* has a modeling methodology for creating executable BMs with real operational data to support validation of the syntactic correctness of a BM modeled using *UncerTum* and checking conformance of the BM with the real operational data;
2 *UncerTolve* defines a systematic and automated process for validating a BM with both subjective and objective uncertainty and defines a set of heuristic rules (named as tolveR-E) to guide test modeler(s) to update the BM based on validation results;
3 *UncerTolve* is equipped with an automated solution for calculating and abstracting objective uncertainty measurements from the real operational data and the obtained measurements are appended to the BM;
4 *UncerTolve* applies a machine learning technique to infer test oracles (state invariants in UML state machines) and test data specifications (guard conditions in UML state machines);
5 *UncerTolve* defines a set of heuristic rules to evolve a BM with inferred state invariants, guard conditions and objective/subjective uncertainty measurements;
6 *UncerTolve*, as a proof-of-concept, is evaluated with one industrial CPS, i.e., GeoSports from the healthcare domain.

1.4 Results and the Structure of the Paper

With *UncerTolve*, we managed to evolve 51% of belief elements, 18% of states, and 21% of transitions as compared to the initial BM. Thus, we conclude that *UncerTolve* is successful in evolving BMs with subjective and objective uncertainty information.

The rest of the paper is organized as follows: Section 2 discusses the related work. Section 3 represents the background. Section 4 represents terminology and running example. Section 5 represents the overall workflow of *UncerTolve*. Section 6 represents the methodology of *UncerTolve*. Section 7 presents the evaluation and discussion, whereas we conclude the paper in Section 8.

2 Related Work

In this section, we compare *UncerTolve* with existing works in Section 2.1, whereas comparison with our own previous related works in Section 2.2.

2.1 Comparison with Existing Works

Several works (e.g., [4, 23-31]) have been published in the literature that infer, e.g., FSMs, their extensions, Live Sequence Charts (LSCs), and properties of software applications from execution traces. Most of these works rely on Daikon [4] to dynamically infer invariants from execution traces.

The work reported in [23] infers deterministic FSMs of black box components from their execution information to understand their behavior in the absence of a formal specification. The inferred FSMs are further generalized into intentional behavior models by synthesizing graph transformation rules. The process involves identifying invariant properties in a similar way as Daikon. The approach was evaluated with three different sets of classes implementing data abstractions such as Queue and MinSet.

An empirical study is reported in [24] to evaluate four strategies of inferring FSMs: 1) traces-only, 2) invariants-only, 3) invariant-enhanced-traces, i.e., inferring models from execution traces followed by enhancing them with invariants), and 4) trace-enhanced-invariants, i.e., inferring models from invariants followed by enhancing them with execution traces). Nine open-source libraries were used to compare the four strategies based on the quality of the resultant FSMs. The second and third strategies were evaluated to be the best ones.

Lo et al. [25] proposed an approach with a tool to enhance the precision of mining FSMs from code and traces by inferring temporal properties and incrementally merging equivalent states. Similarly, Walkinshaw and Bogdanov [26] proposed an approach to allow additional inference of state machines, based on temporal logic formulas and an extra capability to introduce new formulas during the inference process. The proposed approach was evaluated with two software applications. Gabel and Du [32] presented a general specification mining framework (Javert) for learning complex temporal properties (specified as specification patterns in FSMs) from execution traces.

Krka et al. [18] proposed an automated approach to infer object-level FSMs from execution traces and program invariants. First, it derives an FSM that captures legal invocation sequences of an object's public interfaces based on inferred data-value invariants. Second, it uses collected dynamic invocation traces to refine the invariant-based FSM to an object-level FSM.

Tonella et al [27] [28] proposed an approach to infer FSMs for supporting MBT based on a combination of clustering, invariant inference and genetic algorithm (GA). GA was used to optimize the quality attributes of inferred FSMs. The approach was evaluated with a small e-commerce application.

Walkinshaw and Taylor [33] proposed an approach to infer deterministic Extended FSMs (EFSMs) with WEKA [34] and Daikon and evaluated the approach with five Java and Erlang programs.

An algorithm is proposed in [26] to extract FSMs with parameters (FSAMs) from interaction traces: sequences of method invocations. FSAMs put constraints on the values of parameters. The algorithm has three sequential steps: merging similar traces, deriving constraints with Daikon, and merging equivalent states. The Builder design pattern was used to evaluate the proposed approach.

In [29], an approach was proposed to infer communicating FSMs (CFSMs) from execution traces of concurrent programs that has three steps: 1) mining temporal properties (invariants), 2) creating an initial CFSM model, and 3) refining the CFSM model. The proposed approach was evaluated with three networked systems. The authors of [31] proposed an approach to infer resource-aware FSMs from execution logs of the software application by following similar steps. The proposed approach was evaluated with a case study on the TCP protocol.

Berg et al. [30] proposed a way to adapt regular inferences of FSMs from observations of component behaviors to construct models of communication protocol entities. The challenge that the authors tried to tackle is to infer state machines where messages have arbitrary parameters; however, it only handles Boolean parameters. Later on, Berg et al. [19] also made an effort to infer state machines with an infinite state space. First, the proposed approach infers finite-state Mealy machines by observing the behavior of a communication

protocol from a small domain. Second, it transforms them into infinite-state Mealy machines by folding the inferred finite-state Mealy machines into compact symbolic models.

Lo et al. [20] presented an approach to mine specifications as restricted LSCs from execution traces that are transformed into UML sequence diagrams with a modal profile applied. Later on, Lo and Maoz [21] made an effort to integrate the value-based specification mining approach of Daikon with a sequence-based approach to mine specifications as LSCs. A scenario-based slicing technique was applied to obtain sliced traces. Value-based invariants mining is then applied to both on the original traces and the sliced traces to identify scenario-specific invariants. Four software applications were used to evaluate the proposed approach.

Beschastnikh et al. [35] proposed an automated approach to infer invariant constrained models from system execution logs, by intentionally reducing the involvement of developers. First, the approach mines temporal invariants from logs and generates trace models, from which it generates initial models (in the form an authors-defined, edge and node style graphical representation). These initial models are then refined and coarsened to explore the space of models.

Raz et al. [36] proposed a way to infer invariants based on the observations of the behavior of dynamic data feeds (i.e., a time-ordered sequence of observations) to detect semantic anomalies in online data sources. The approach relies on an augmented Daikon and Mean (i.e., a statistical method for estimating a confidence level for the mean of a distribution). The approach was evaluated with real-world data. In [37], the authors proposed a heuristics based algorithm to scale up dynamic inferences of properties/invariants of software applications from execution traces. The approach was evaluated on JBoss and the Windows kernel. Hangal and Lam [38] proposed an approach (similar to Daikon) to detect program invariants from program executions. It also reports detected dynamic invariant violations.

The work reported in this paper builds on an existing work, i.e., Daikon to infer invariants based on execution information. However, in our case, real operational data was used from real applications of CPSs. To compare with these related works, we distinguish *UncerTolve* from the following four aspects. First, most of the related work directly take programs as input to infer, e.g., specifications and API. *UncerTolve*, however, takes test ready UML models together with explicitly captured subjective uncertainty as input and evolve them

based on model execution using real operational data, based on dynamic inference with Daikon. Second, *UncerTolve* aims to handle CPSs, not just programs. This means that we not only evolve models of applications but also infrastructures and their interactions. Third, *UncerTolve* evolves belief models including discovering previously unknown belief elements (in stereotypes), states, and transitions. Fourth, the ultimate objective of *UncerTolve* is to facilitate MBT of CPSs under known and unknown uncertainty, instead of program comprehension and bug detection like most of the related works do.

2.2 Comparison with Our Previous Works

To understand uncertainty in general, in our previous work [15], we developed a generic conceptual model called U-Model. Our aim was to precisely define uncertainty and its associated concepts for CPSs. The U-Model was implemented in two ways: 1) As an extension of an existing restricted use case specification language (named as RUCM) [35], to specific uncertainty in use case specifications called as U-RUCM [39], 2) Implementation of U-Model as a UML profile—the UML Uncertainty Profile (UUP) to enable MBT of CPSs under uncertainty. UUP together with other related profiles and model libraries were implemented as a modeling framework—*UncerTum* [9]. With *UncerTum*, test modelers can create BMs with explicit consideration of subjective uncertainty. As shown in Fig. D-1, BMs created with *UncerTum* [9] are the key inputs of *UncerTolve*—the key contribution of this paper. *UncerTum* only focuses on creating BMs for test case generation and cannot be used to further enhance BMs into executable ones such that these models can be validated with real operational data. In the context of *UncerTolve*, we propose an extension to *UncerTum* for converting BMs developed with *UncerTum* into executable ones.

In [11], we reported *UncerTest* [11], an uncertainty-wise testing framework. *UncerTest* implements various uncertainty-wise test case generation and minimization strategies that can be used to generate test cases from BMs developed with *UncerTum* [9]. *UncerTest* [11] can be used to generate test cases from BMs evolved with *UncerTolve* to test CPSs. However, we may need to define additional test strategies in *UncerTest* [11] to focus specifically on the evolved parts of the evolved models. We plan to implement these test strategies in our future work.

3 Background

In this section, we present the background that is necessary to understand the rest of the paper. In Section 3.1, we define a CPS at a generic level, along with three logical levels, at which uncertainty may occur. In Section 3.2, we introduce UTP, which is one of the key profiles applied to BMs to enable MBT. Section 3.3 presents U-Model, a conceptual model defining uncertainty and its associated concepts. Section 3.4 introduces *UncerTum*, an uncertainty-wise modeling framework to create BMs of CPSs with subjective uncertainty, i.e., the key input of *UncerTolve*. Section 3.5 presents *UncerTest*, an uncertainty-wise test case generation, and minimization framework to generate test cases from BMs created with *UncerTum* and evolved with *UncerTolve*.

3.1 Cyber-Physical Systems and Uncertainty Levels

A CPS is defined as [15]: *"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. D-2. Uncertainty in a CPS can occur at the following three logical levels [15]. First, uncertainty at the *Application level* is due to events/data originating from an application (one or more software components) of a physical unit of the CPS. An example of application-level uncertainty is the indeterminate behavior of a human interacting with a CPS, e.g., not wearing a device sensing heart rate properly which leads to uncertain heart rate readings. Second, uncertainty at the *Infrastructure level* is due to data transmission via information network enabled through networking infrastructure and/or cloud infrastructure. An example of infrastructure level uncertainty includes the uncertain behavior of a CPS due to packet loss in an information network. The third level is the *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 (e.g. the abnormal heart rate captured by heart rate sensor due to the loss of the connection between heart rate sensor and a computer system analyzing the heart rate assuming the heart rate sensor and a computer system are connected via wireless network). More details and examples are provided in [15].

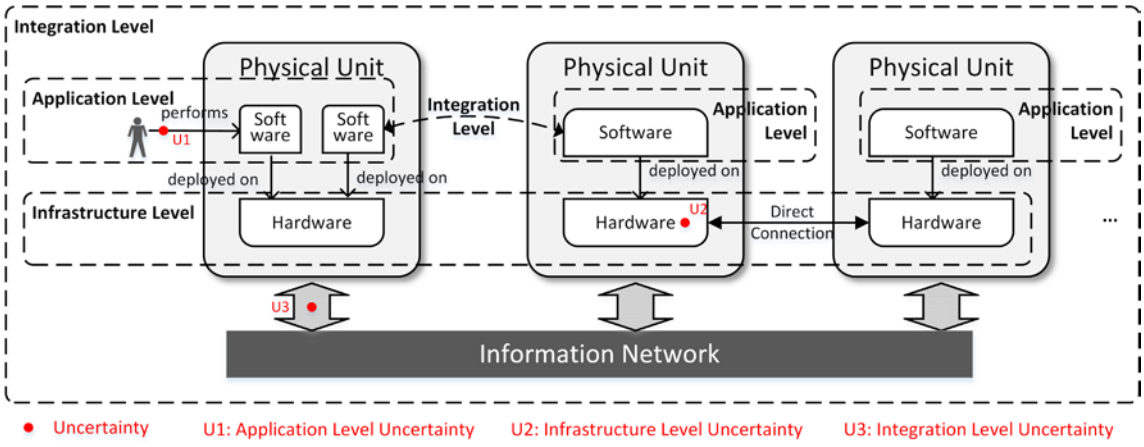


Fig. D-2. The Overall Context and Scope of *UncerTolve*

3.2 UML Testing Profile

UML Testing Profile (UTP) [39] 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 on 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 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. 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 UTP V.2 and the team can be found in [27; 38].

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

3.3 U-Model

To understand uncertainty in the general context of software engineering, we developed a conceptual model called U-Model [15] to define uncertainty and its associated concepts. The 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 case studies.

The U-Model takes a subjective approach to represent uncertainty, which is modeled as a state (i.e., worldview) of some agents (called *BeliefAgent*s), who, for whatever reason, do not have complete and fully accurate knowledge about some subjects of interest. A *Belief* is an abstract concept that can be expressed in the concrete form via one or more explicit *BeliefStatement*s (a concrete and explicit specification of some *Belief* held by a *BeliefAgent* about possible phenomena or notions belonging to a given subject area). Uncertainty (i.e., lack of confidence) represents "a state of affairs whereby a *BeliefAgent* does not have full confidence in a belief that it holds" [15]. This may be due to several factors: lack of information, inherent variability in the subject matter, ignorance, or even due physical phenomena such as the Heisenberg uncertainty principle [21]. While uncertainty itself is an abstract concept, it can be quantified by a corresponding *Measurement*, which expresses in some concrete form the subjective degree of uncertainty that the agent ascribes to a *BeliefStatement*. As the latter is a subjective notion, a *Measurement* should not be confused with the degree of validity of a *BeliefStatement*. Instead, it merely indicates the level of confidence that the agent has in a statement. Further details on U-Model may be consulted in [15].

3.4 UncerTum

UncerTum [9] (Fig. D-1) is uncertainty-wise modeling framework to support the development of BMs of CPSs, which consists of specialized UML-based modeling notations (named as UUP) for specifying uncertainties to enable MBT of CPSs under uncertainty.

UUP is at the core of *UncerTum* and implements U-Model [15] (Section 3.3). UUP consists of three parts (i.e., *Belief*, *Uncertainty*, and *Measurement* profiles) and an internal library containing enumerations required in the profiles. To ease the development of BMs with uncertainty, *UncerTum* additionally defines four sets of UML model libraries: *Pattern*,

Time, *Measure*, and *Risk* libraries, by extending an existing UML profile for Modeling and Analysis of Real-Time and Embedded Systems (MARTE) [40]. *UncerTum* also includes a small UML profile called the CPS testing levels profile to allow stereotyping (labeling) test ready model elements with three CPS test levels (e.g., integration level). The purpose is to differentiate model elements from different levels and facilitate defining level specific test strategies. Moreover, *UncerTum* relies on UTP V.2 (Section 3.2) to model BMs for the purpose of enabling MBT. Finally, *UncerTum* defines a set of concrete guidelines (i.e. *Measurement Modeling*) on how to use its modeling notations to construct BMs with uncertainty explicitly specified.

3.5 UncerTest

UncerTest [11] (shown in Fig. D-1) consists of two main part: test case generation and uncertainty-wise test case minimization. *Test case generation* takes the BM using *UncerTum* as input to automatically and systematically generate abstract test cases, according to two proposed test case generation strategies: *All Simple Paths* (No Loops) and *All Paths with a Fixed Maximum Length*. These two strategies were inspired from the ones reported in [41], but were extended for BMs specified in *UncerTum* and considered various uncertainty aspects such as the number of uncertainties in a test path and overall uncertainty of a test path, defined based on Uncertainty Theory [42]. *Uncertainty-wise test case minimization* was proposed because the number of abstract test cases generated by *Test Case Generation* is typically very large for any non-trivial CPS and it is impossible to execute all of them. The uncertainty-wise test case minimization problem is a multi-objective search problem with four objectives: 1) The average number of uncertainties covered by the subset of the test cases after minimization; 2) The average percentage of *uncertainty space* (defined in Uncertainty Theory [42]) covered by the subset of the test cases after minimization; 3) The average *uncertainty measure* (defined in Uncertainty Theory [42]) of the subset of test cases after minimization; and 4) The average number of unique uncertainties covered by the subset of test cases after minimization.

4 Terminologies And Running Example

In this section, we will briefly present a running example together with relevant terminologies. The running example will be used in the rest of the paper to explain the key steps of *UncerTolve*. The *UncerTolve* itself will be presented in Sections 5-6.

4.1 Belief Test Ready Model

A *belief test ready model* (BM) consists of three types of models: a Composite Structure (CS) diagram, a set of class diagrams (CDs), and a set of Belief State Machines (BSMs). The belief aspect of BMs is from the perspective of modelers (i.e., belief agents), who create the BMs and therefore the BMs reflect their (subjective) beliefs on the information specified in the models.

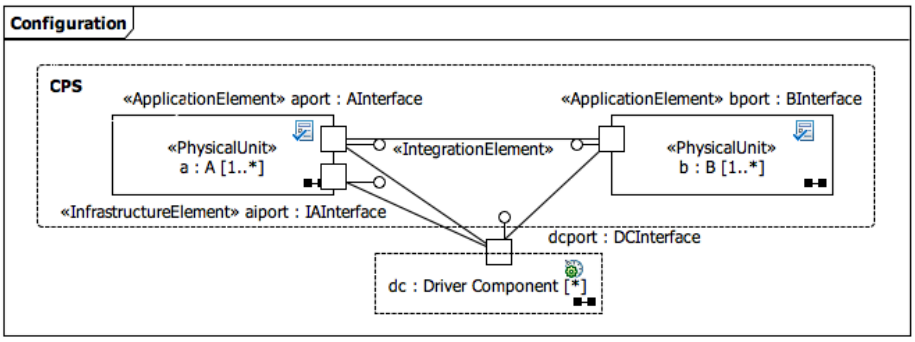


Fig. D-3. Composite Structure Diagram of BM (Running Example)

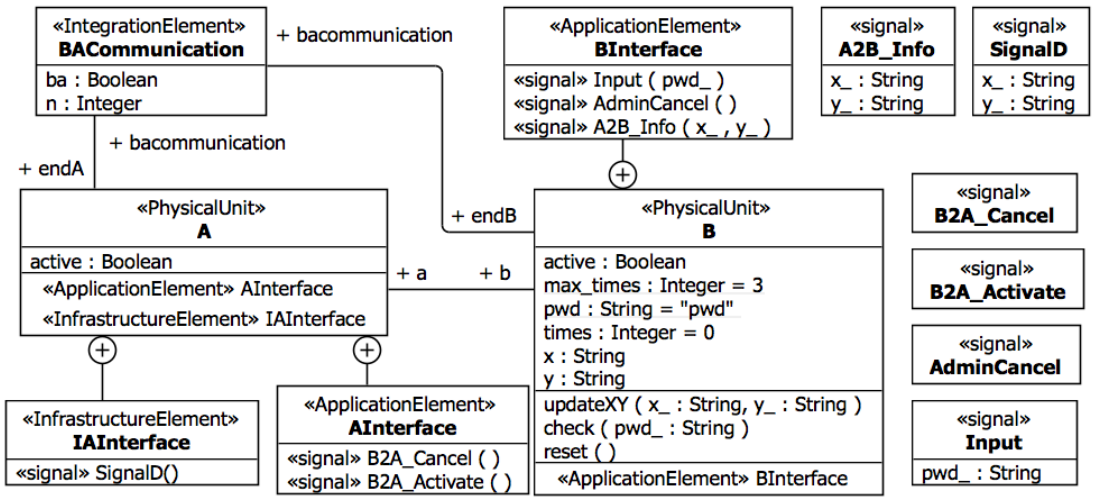


Fig. D-4. Class Diagram of BM (Running Example)

The CS diagram of a BM model represents a high-level test/model evolution configuration (referred as *Configuration* in Fig. D-3) of a CPS under Test. It captures various physical units that constitute the CPS, such as components *A* and *B* with «PhysicalUnit» applied. The stereotype is defined in the CPS profile of *UncerTum* [9]. Application and infrastructure level testing ports and interfaces of each physical unit are also explicitly modeled in the CS diagram. For example, as shown in Fig. D-3, *A* has one application-level port (*aport :: AInterface*) and one infrastructure level port (*aiport :: IAInterface*), which are stereotyped with «ApplicationElement» and «InfrastructureElement», respectively. Each port has a corresponding interface specified in the class diagram (Fig. D-4) such as *AInterface*. The integration level interface is stereotyped with «IntegrationElement» (represented as class *BACommunication* in Fig. D-4) and it is associated with *A* and *B*.

The structure of physical units is modeled as a set of UML class diagrams. Classes in the UML class diagrams capture various types of information required for testing, including APIs (e.g., the *reset()* operation of *B* in Fig. D-4), state variables (e.g., the *active:Boolean* attribute of *A* in Fig. D-4), test configuration parameters (e.g., the cardinality of instances of *A* in Fig. D-3), signals (e.g., *AdminCancel* in Fig. D-4), and signal receptions (e.g., *AdminCancel()* in *BInterface* in Fig. D-4). The class diagrams have the CPS profile (Section 3.1) applied to distinguish model elements of the three different levels. For example, *AInterface* is stereotyped as «ApplicationElement» to signify that it is an application level interface.

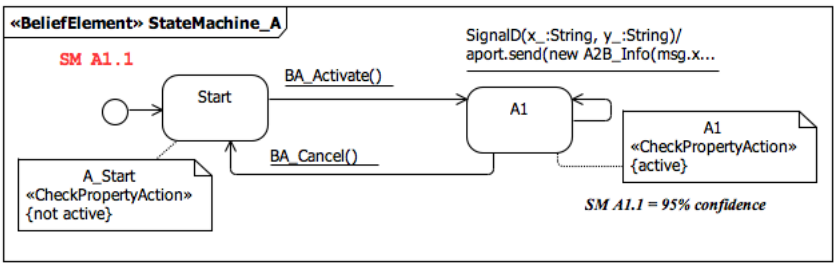


Fig. D-5. Belief State Machine of *A* (V1.1) (Running Example)

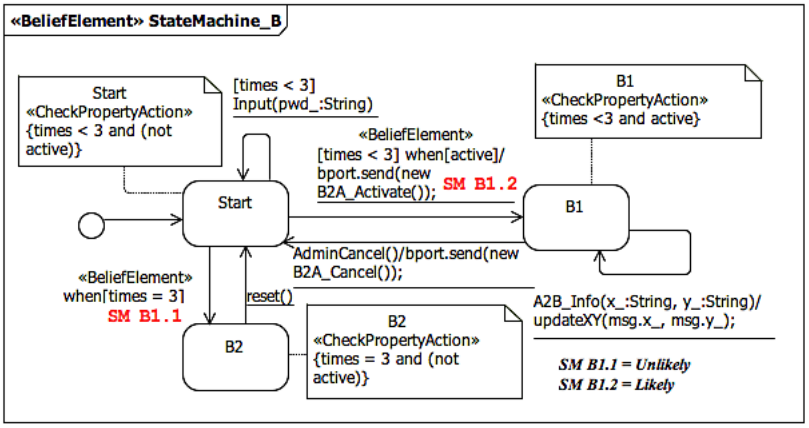


Fig. D-6. Belief State Machine of *B* (V1.1) (Running Example)

Each physical unit's test behavior is modeled as one or more BSMs using *UncerTum* (Section 3.4), e.g., as shown in Fig. D-5 and Fig. D-6 for *A* and *B* respectively. For example, as shown in the BSM for physical unit *A* (Fig. D-5), «BeliefElement» from UUP is applied to the state machine for *A*, where the confidence of the belief agent about this state machine is specified as 95%. Two key types of OCL constraints are defined in BSMs. Each basic state in a BSM is precisely defined with a state invariant (e.g., *not active* associated with the *Start* state of *A*, Fig. D-5) specified as an OCL constraint based on state variables defined in the CDs (e.g., attribute *active* of class *A* in Fig. D-4). These state invariants serve as test oracles and can be checked at runtime using existing OCL evaluators such as Eclipse OCL [43]. Second, each guard condition (e.g., guard *[times<3]* on the transition from *Start* to *B1* in *B*, Fig. D-5) is specified as an OCL constraint on the input parameters of the associated trigger, which defines the valid range of inputs. These constraints in our case are used to automatically generate test data to trigger transitions. An OCL solver (e.g., EsOCL [44]) can take these constraints as input and automatically generate test data [45]. BSMs are further enriched with UAL such that they can be directly executed with the IBM Rational Simulation Toolkit [46]. For example, the self-transition of State *A1* (Fig. D-5) has an effect whose body is written in UAL as below:

```
aport.send(new A2B_Info(msg.x_, msg.y_));
```

This effect simply sends signal *A2B_Info* from *A* to *B* via *aport*. Notice that *A2B_Info* is a signal reception in *BInterface*. Notice that signals in UML are typically used for modeling communications across state machines. In our running example, for instance, the state machine of physical unit *A* (Fig. D-5) communicates with the state machine of physical unit

B (Fig. D-6) via the UAL code of the effect of self-transition of state *A1* (Fig. D-5) as also shown in the last paragraph. Similarly, the state machine of physical unit *B* (Fig. D-6) communicates with the state machine of physical unit *B* (Fig. D-5) via the UAL code written in the effect of the transition from *B1* to *Start* in Fig. D-6.

4.2 Executable Belief Test Ready Model

An executable BM is a Java code, which is semantically equivalent to a BM discussed in Section 4.1. Executable BM Java code can be executed either directly with the IBM Rational Simulation Toolkit or as a standalone application by simply introducing a *main()* method. In Section 6.1, how to develop executable BM will be discussed in details.

4.3 Driver Model

In order to apply *UncerTolve*, we need to develop a component called *Driver Component* (e.g., *dc: Driver Component* in the composite structure diagram in Fig. D-3). A *Driver Component* is connected to physical units of a CPS via UML ports and connectors (Fig. D-3). A *Driver Component* has its own class diagram (e.g., Fig. D-7) and state machine (*Driver State Machine (*DSM*)*, e.g., Fig. D-8). The class diagram contains attributes and operations that are specifically needed to model DSM such as attribute *isCorrectInput* of *Driver Component* in Fig. D-7. A DSM is a UML state machine that is specifically defined to trigger the execution of BSMs based on real operational data. Data types of the real operational data (e.g., *x* and *y*) are specified in the class diagrams of BM. Data on signals (e.g., *SignalD* of Fig. D-4) are sent via ports (e.g., *dcport* in Fig. D-3) from DSM to BSMs. Such data includes the input of a system actor, environment changes, etc. A DSM can also be enriched with UAL to make it executable. The class diagram and DSMs are all together called Driver Model (DM).

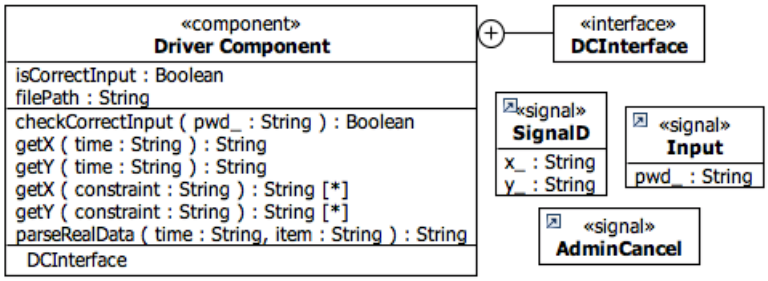


Fig. D-7. Class Diagram of DM (Running Example)

The DSM of our running example is shown in Fig. D-8. The DSM has two regions, i.e., the top region is used to communicate with *A*, whereas the bottom region is for communicating with *B*. In the top region, there is only one state called *Sending Data* having a self-transition “after 0.01s”. This means that every 0.01 seconds, data is sent from *Driver Component* to *A*. The following UAL code is embedded inside the entry activity of the *Sending Data* state:

```
dcPort.send(new SignalD(getX(time), getY(time)));
```

The above code obtains values of *x* and *y* at a given point in *time* (from real operational data) and sends them to *A* via *SignalD* through *dcPort*. In the bottom region, in the *Sending Input Data* state, the following UAL code is added:

```
String pwd_ = parseRealData("input_pwd");
      dcPort.send(new Input(pwd_));
isCorrectInput = checkCorrectInput(pwd_);
```

The above code obtains the password from real operational data (the first line), sends it to *B* via the *Input* signal through *dcPort* (the second line), and checks whether or not the password is correct with the *checkCorrectInput (pwd_)* operation in the class diagram of *Driver Component*.

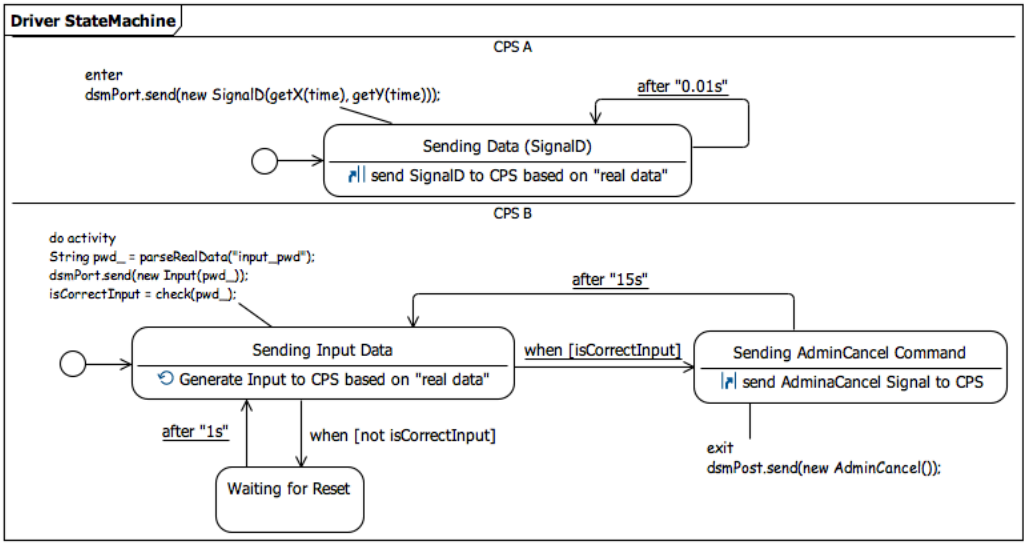


Fig. D-8. State Machine of DM (Running Example)

5 Architecture and Current Implementation of *UncerTolve*

In the rest of the section, we first present the overall architecture of *UncerTolve* (Section 5.1), followed by the current implementation of *UncerTolve* (Section 5.2).

5.1 Architecture

The architecture of *UncerTolve* is represented in Fig. D-9. The key input of the architecture is real operational data collected from existing deployments. Real operational data can be collected continuously; therefore a process of using *UncerTolve* for evolving BMs can be iterative. As long as new operational data available, *UncerTolve* can be used to evolve the current BM and therefore the evolved BM can be used to generate new test cases for testing new/future deployments. Real operational data are invoked by *Driver Model* and *Executable Belief Test Ready Model* (executable UML) for enabling model execution. Model execution results (i.e., discovered previously unknown objective uncertainty measurements and errors in the initial BM) are used to evolve the BM with a set of heuristics (i.e., *tolveR-E*). Real operational data are also used to support the dynamic invariant inference, which produces *Invariants*, representing either test oracles (i.e., state invariants) or test data specifications (i.e., guard conditions). Results of the inference are used to further evolve the belief BM, based on another set of heuristics: *tolveR-D*.

Fig. D-9 shows the necessary components of the *UncerTolve* architecture. An initial *Belief Test Ready Model*, semantically equivalent *Executable Belief Test Ready Model*, *Driver Model*, and *Real Operational Data* are key artifacts that need to be constructed in order to use *UncerTolve*. Definitions and examples of these artifacts are presented in Section 4 for references. For each of the activities (i.e., modeling, model execution and invariant inference), a set of guidelines (i.e., *S1*, *S2*, and *S3*) is also defined to guide users through a non-trivial model evolution process. As part of the guidelines, *tolveR-E* and *tolveR-D* are the two sets of heuristics defined for refining the initial BM based on model execution and invariant inference results.

There are three evolution ports defined on *Belief Test Ready Model*: 1) following *tolveR-E*, based on *Execution Log* (output of model execution), to evolve UML class diagrams, composite structure diagrams and BSMs, 2) following *tolveR-D*, based on invariants derived via Dynamic Invariant Inference, to evolve test oracle and test data specifications specified as state invariants and guard conditions of BSMs, and 3) appending objective uncertainty measurements derived from model execution to the BM. Though, the *UncerTolve* architecture provides these three evolution ports, it is not necessary to use them all at once. Depending on needs and contexts, which one(s) to use and how to use them can be customized.

Note that this architecture is generic since it can be integrated with different technologies (e.g., different invariant inference engines) to achieve the same or similar objectives. Section 5.2 discusses the current implementation of this architecture.

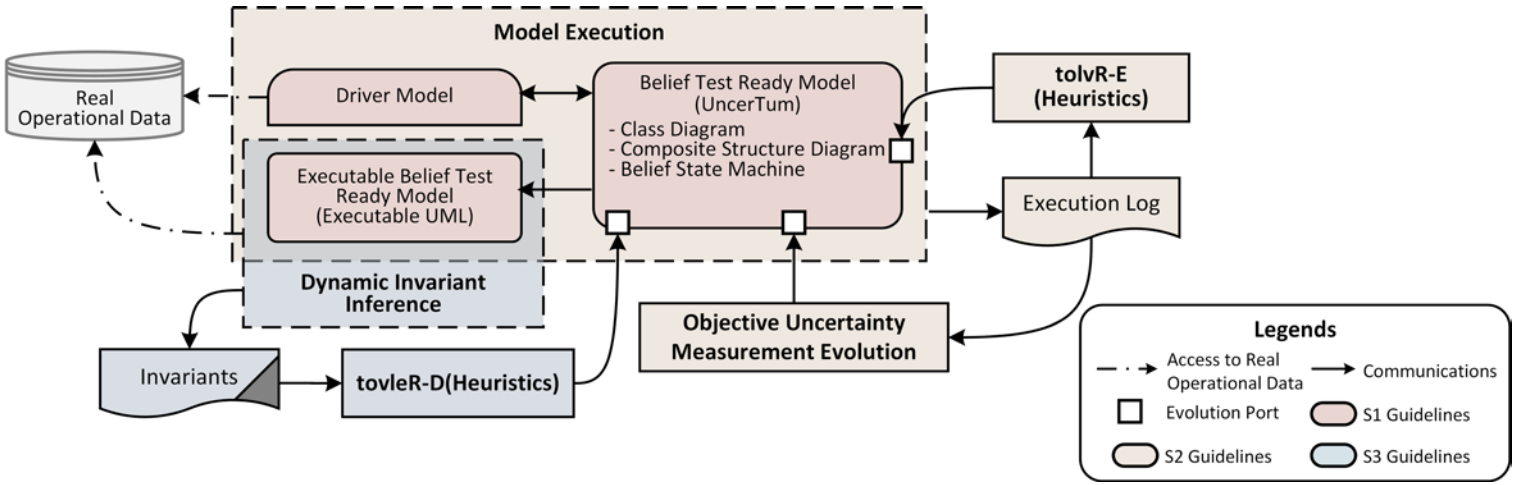


Fig. D-9. The Overall Architecture of *UncerTolve*

5.2 Current Implementation of UncerTolve

In this section, we discuss our current implementation of *UncerTolve*, focusing on the selection of technologies and the integration of them. The overall workflow of the current implementation of *UncerTolve* is presented in Fig. D-10. The selection of technologies and corresponding justifications are summarized in Table D-2. The recommended methodology for using the current implementation of *UncerTolve* is however discussed in Section 6.

Table D-2. Steps, techniques/tools/languages, and corresponding justifications of the current implementation of *UncerTolve*

Step	**Techniques/tools/languages**	**Justification of using selected techniques/tools/languages**
S1, S2	IBM Rational Software Architect (RSA) IBM Simulation Toolkit UML Action Language (UAL) Eclipse OCL Java Implementation of heuristics *tolveR-E*: *Execution Logger* and *Log Analyzer*	The overall approach of the U-Test project is implemented in the CertifyIt[22] tool, which is a plug-in to IBM RSA. UAL is implemented based on the OMG Alf standard and is used by the IBM RSA Simulation Toolkit. Thus, IBM RSA, Simulation Toolkit, and UAL were selected in the current implementation of *UncerTolve*. Eclipse OCL is one of the commonly used OCL evaluation tools, which is built on EMF and fits well with the tooling of our overall technical solution. Given that execution log cannot be used automatically to modify BMs, heuristics *tolveR-E* are implemented to propose a set of actions to the user to modify BMs with uncertainty.
S2	Java Implementation of *Objective Uncertainty Measurement Analyzer*	Our approach is based on *subjective* uncertainty. To further validate it, we calculate the frequency (objective uncertainty measurements) based on the real operational data.
S3	Daikon Invariant Detector *Invariant Converter* (Java Implementation planned)	Several dynamic inference tools exist in the literature [1-3]; however, we decided to use Daikon because it implements a set of optimizations that facilitates its applications to complex problems [4].
S3	Java Implementation of heuristics *tolveR-D*: *Invariant Analyzer*	Daikon outputs invariants. Links must be established between the inferred invariants and models elements of BMs. Thus, we developed heuristics *tolveR-D* to link Daikon invariants with state invariants and guard conditions (specified as OCL constraints and representing test oracles and test data specifications) of the BMs.
S1-S3	Java implementation of the integrated solution (Fig. D-10)	None

The first activity is to develop and execute BMs (S1/S2 in Fig. D-10). This activity takes place in IBM's Rational Software Architect (RSA) and its Simulation Toolkit plugin [46]. As shown in Fig. D-10, *UncerTum* (Section 3.4) is currently implemented in IBM RSA. A user of *UncerTolve* develops BMs (S1 in Fig. D-10) using the guidelines developed for *UncerTum* (see [9]). To validate BMs, such models must be executed with real operational data. To achieve so, we extended *UncerTum* to provide a set of new guidelines to convert BMs into executable ones. The methodology for creating executable models is described in

Section 6.1. Corresponding to BMs, equivalent Java code is automatically generated by the Simulation Toolkit [46], which can either be executed with the Simulation Toolkit or as a standalone Java application. The user executes the developed BMs with the real operational data (S1 in Fig. D-10) using the Simulation toolkit [46].

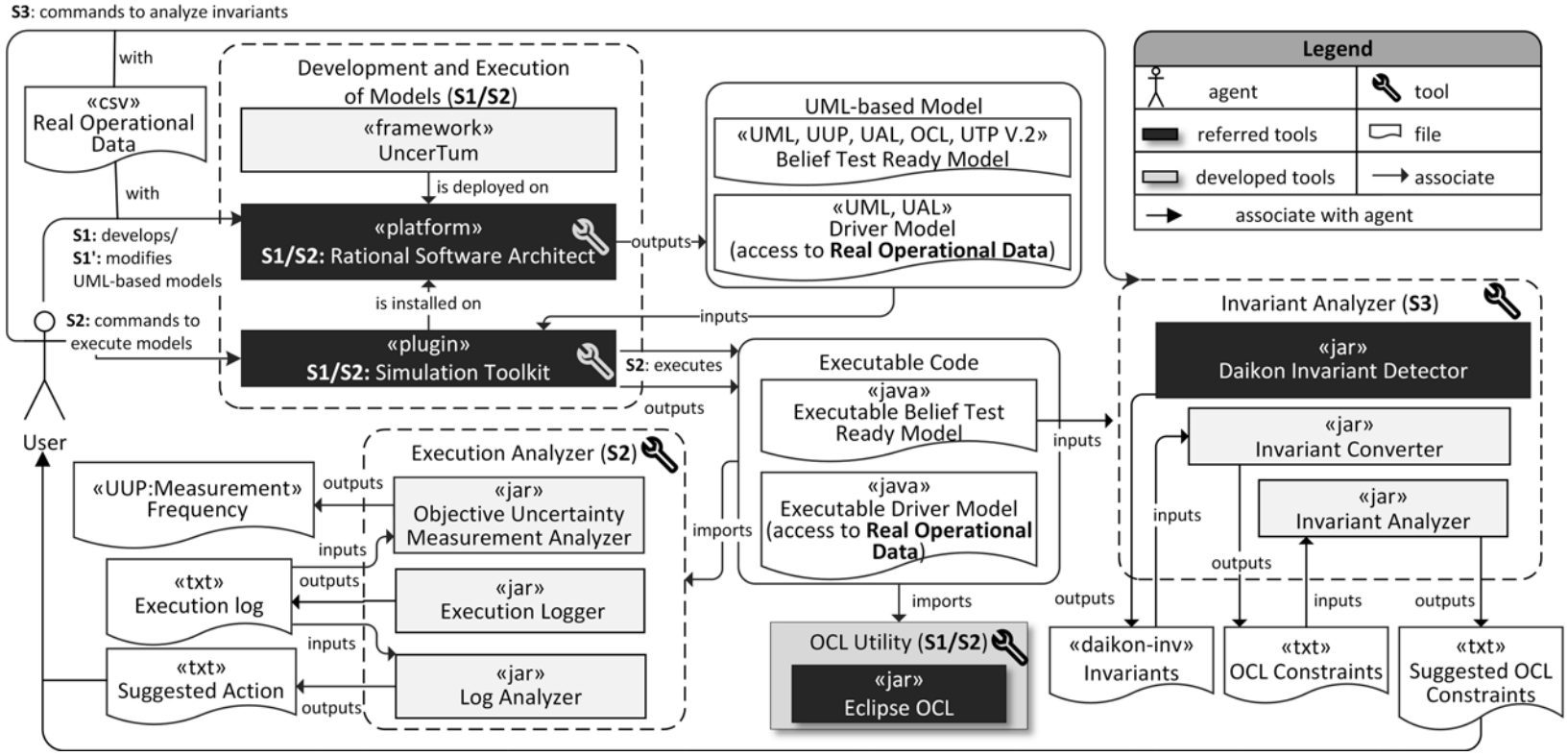


Fig. D-10. The Overall Work Flow of the Current Implementation of *UncerTolve*

The second activity of is *Execution Analyzer* (S2), which is used to analyze the results of the execution of BMs based on real operational data. *Execution Analyzer* is composed of *Execution Logger*, *Log Analyzer*, and *Objective Measure Analyzer*. Once the BMs are executed, *Execution Logger* logs the execution as execution log. The execution log includes information such as at one point of time, which trigger was fired with which data. Such log is used by *Log Analyzer* as an input to suggest various actions for the user to update BMs based on the set of heuristics of *tolveR-E* (Section 6.2.1). Based on suggested actions, the user may update BMs (S1' in Fig. D-10). This log is also used by *Objective Uncertainty Measurement Analyzer* to calculate conditional probabilities, e.g., the frequency of occurrence of a particular transition (details in Section 6.2.2).

The third activity is about the analysis of invariants using a machine learning technique implemented in the Daikon Invariant Detector [4]. The user may command (S3 in Fig. D-10) to infer invariants based on real operational data using the API we developed to invoke Daikon and access the real operational data. As a result, Daikon outputs a set of invariants, which are converted to OCL constraints by the *Invariant Converter* that we implemented in Java. The converted OCL constraints are inputted to *Invariant Analyzer* to further evolve invariants in BMs based on the set of heuristics of *tolveR-D* (Section 6.3), and the output is suggested OCL constraints as a feedback to the user. The user may accept, reject, or modify the suggested OCL constraints.

6 Recommended Methodology

The recommended methodology for applying the current implementation of *UncerTolve* is presented in Fig. D-11, from which one can see that it is iterative and has three sequential steps. In the rest of the section, each of these steps is discussed in details. Section 6.1 presents our proposed modeling methodology to create executable BMs including activities for creating BMs and UML models for *Driver Component*; Section 6.2 presents a set of activities for validation BMs and DMs and evolve objective uncertainty measurements (S2); Section 6.3 presents the process of evolving BMs in terms of invariants using dynamic invariant inference (S3).

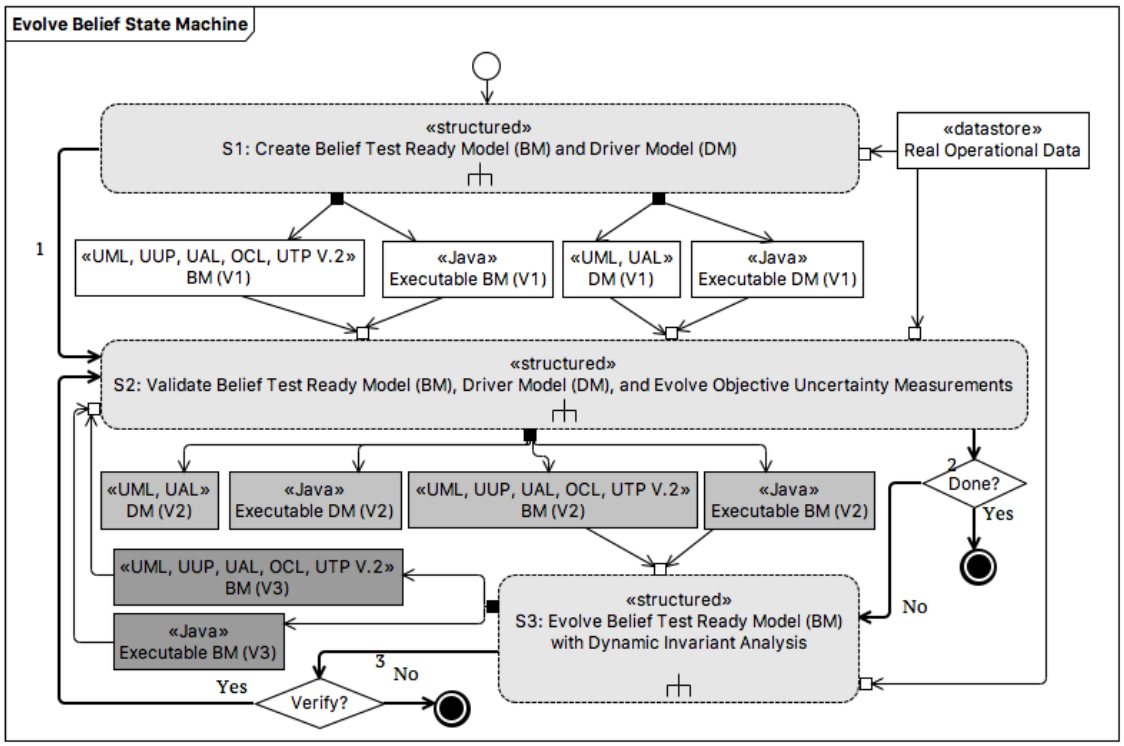
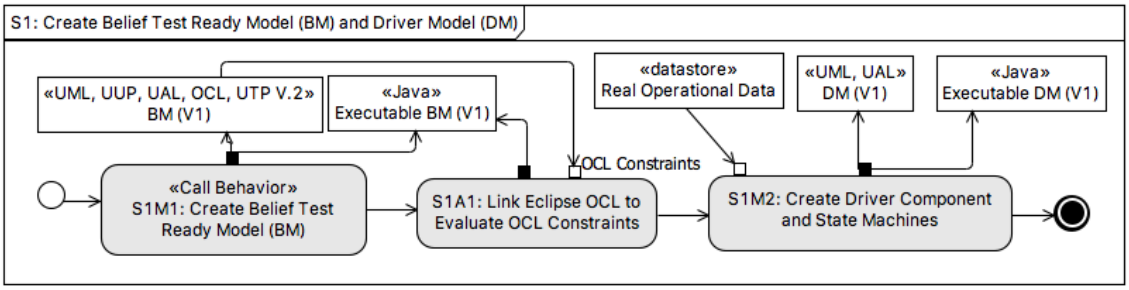


Fig. D-11. Recommended Methodology of using the Current Implementation of *UncerTolve*

6.1 Creating BM and Driver Model (S1)

As we previously discussed, with *UncerTum* [9], BMs can be created. Using *UncerTest*, executable test cases can be generated from the BMs specified in *UncerTum*. However, an extension of *UncerTum* is required to make BMs executable such that they can be validated against real operational data. This section only focuses on the aspects that are required to make BMs executable and other details on *UncerTum* are provided in [9]. As shown in Fig. D-12, S1 is broken down into three activities: S1M1, S1A1, and S1M2.

As the first step of the *UncerTolve* methodology, a modeler (belief agent) needs to apply *UncerTum* (which integrates the modeling notations of UML, UUP for specifying uncertainty, UAL for model execution, OCL for specifying constraints, and UTP V.2 for capturing testing aspects, e.g., «BeliefElement» in Fig. D-5 and Fig. D-6) to create BM (V1), i.e., the initial version of BM for a CPS under test (S1M1). UUP and UTP V.2 profiles are implemented in IBM RSA. As discussed in Section 4.1, the BM created by the modeler based on her/his subjective opinions and the BM is composed of a set of class diagrams, a composite structure diagram and a set of BSMs.



S1M1, S1M2 – manual action; S1A1 -- automated action

Fig. D-12. The Structured Activity of Create Belief Model and Driver Model

To make the BM executable, UAL code should be added to relevant model elements of BSMs of the BM such as *entry*, *exit*, and *do* activities of a *state* and *effect* on a *transition*. The second output of S1M1 is Executable BM (V1), which is Java code automatically generated from the initial version of BM (V1) by IBM RSA, and can be automatically executed by the IBM Simulation Toolkit [46] or as a standalone program. For example, Fig. D-5 and Fig. D-6 present the diagrams of the BM (V1) of the running example. The key elements of the model have been discussed in Section 4.

Note that a modeler can specify a *subjective* uncertainty measurement as part of the applied «BeliefElement» on a model element on the BM model. For example, as shown in Fig. D-6, the subjective uncertainty measurement (denoted as *SM B1.2*) for «BeliefElement» applied on the transition from *Start* to *B1* is '*Likely*'. The transition from *Start* to *B2* however '*Unlikely*' occurs (see *SM B1.1*). Note that *SM* means Subjective Measurement and encoding of *BX.Y* means that the *X* round of the derivation of subjective uncertainty measurement for the *Y* element with «BeliefElement» applied.

Since the executable UML implemented in IBM RSA doesn't support converting OCL constraints into Java code and consequently cannot evaluate constraints at the runtime, we implemented *OCLUtility* in Java. This utility links IBM Simulation Toolkit with the Eclipse OCL library to evaluate OCL constraints at runtime (S1A2). Using this activity together with S1M1, executable BM (V1) is developed. Notice that *OCLUtility* is generic and needed to be developed once.

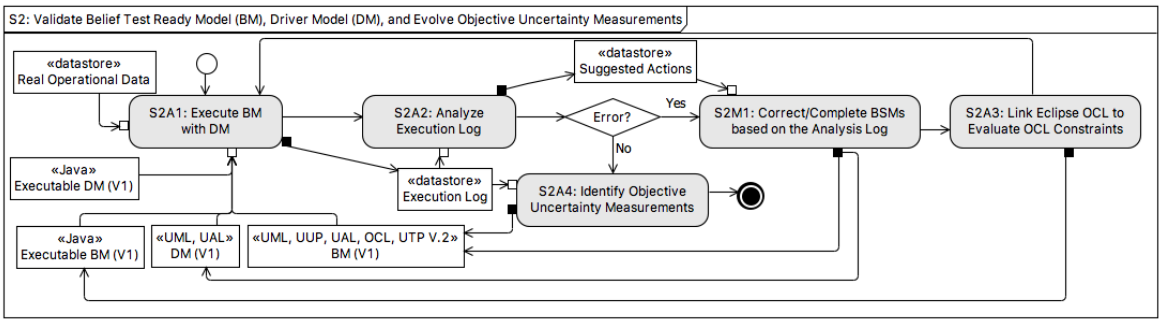
Activity S1M2 is to connect the *Driver Component* to the BM using the same composite structure diagram developed for the BM (e.g., Fig. D-3), create class diagrams to keep information required to create a DSM, and create DSMs to drive the execution of BSMs for

the purpose of validating and evolving them (Section 4.3). Recall that all these models together are called DM. The outputs of this activity are then DM (V1) and its equivalent Java code Executable DM (V1).

For our running example, we show the composite structure diagram of the BM in Fig. D-3 (which is shared with the DM), the class diagram in Fig. D-7, particularly developed for the *Driver Component*, and the DSM in Fig. D-8. Please refer to Section 4.3 for a detailed discussion of the DM model.

6.2 Validate BM and Driver Model, and Evolve Objective Uncertainty Measurements (S2)

The second step (Fig. D-13) is to validate the BM and DM against real operational data and evolve *objective* uncertainty measurements on the BSMs based on the real operational data. Note that *subjective* uncertainty measurements are the ones defined by the belief agent when the initial version of the BM was created.



* S2M1 – manual action; S2A1~S2A4 – automated action

Fig. D-13. The Structured Activity of Validating BM, DM and Evolving Objective Uncertainty Measurements

The first step (S1A1) automatically executes the BM with real operational data using the DM. Results of the execution are stored in the *Execution Log*. Note that the UAL code for generating the execution log is added in the DSM and BSMs for this purpose. The second step (S2A2) automatically analyzes the generated execution log to identify errors and obtain objective uncertainty measurements such as the frequency of the occurrences of a transition in a BSM. If an error is obtained, manual correction and completion of BSMs (S2M1), based on the analysis results obtained in S2A2 are then required. Sequentially, *UncerTolve* automatically establishes the link with Eclipse OCL (S2A3). The process of identifying errors continues until no error is identified, in which case *UncerTolve* automatically adds

discovered objective uncertainty measurements to the BM (S2A4). Notice that the whole process of keeping updating the BM (V1) is continued until the validation is finished. At this moment, the BM (V2), along with the Executable BM (V2), DM (V2) and Executable DM (V2) are generated for S3 to take them as input to evolve BM (Fig. D-11). In the rest of the section, we discuss the key steps of S2.

6.2.1 Analysis of Errors and Fixing Models (S2A2, S2A3, and S2M1)

In S2A2, *UncerTolve* systematically and automatically checks the execution log for various types of errors. We classify errors into two high-level categories: *Syntactic* and *Semantic* errors. Syntactic errors are related to missing, incorrect, and redundant model elements in the BM and DM. For example, a redundant state means that its state invariant is subsumed by the state invariant of another state. A semantic error occurs when the models are syntactically correct, but the semantics of the models introduced using the UAL code have logical errors.

We proposed a set of heuristics for the validation purpose (i.e., *tolveR-E*, 0) in *UncerTolve*. We provide below a subset of *tolveR-E* as examples:

O 1. If the state invariant of a state in a BSM evaluates to be *false*, then it leads to three possible fixing scenarios: adding a new state, changing an existing one, and/or deleting an existing one.

O 2. If a guard condition evaluates to be *false*, then it leads to two options: adding a new transition with an unknown trigger to an unknown state and changing an existing transition.

O 3. If a signal is sent from the DSM to a BSM (which is supposed to transit to a known state) but the signal is not received by the BSM, then this indicates that one or more model elements (e.g., connector) are missing from the BM model.

O 4. If a signal is sent from the DSM to a BSM (which it is supposed to transit to a known state) but the BSM transits to an unexpected state, it means that one or more model elements (e.g., the expected state) are incorrect.

O 5. If a signal is sent from the DSM to a BSM and more than one states of the BSM become active in one region at the same time, this may suggest redundant states.

Regarding the running example, one can observe the following changes to the *StateMachine_B* BSM of the BM (V1) (Fig. D-6): 1) adding new state *B3* (along with the definition of its state invariant as an OCL constraint), 2) adding two new transitions (between states *Start* and *B3*) and 3) applying «BeliefElement» on the two new transitions. The changes are reflected in the new version of the BSM (blue in Fig. D-14). This series of changes were triggered because, in S2A2, *UncerTolve* identified that the real operational data reflects the situation that from state *Start*, under the condition of *times=4*, the systems ends up at the *B3* state.

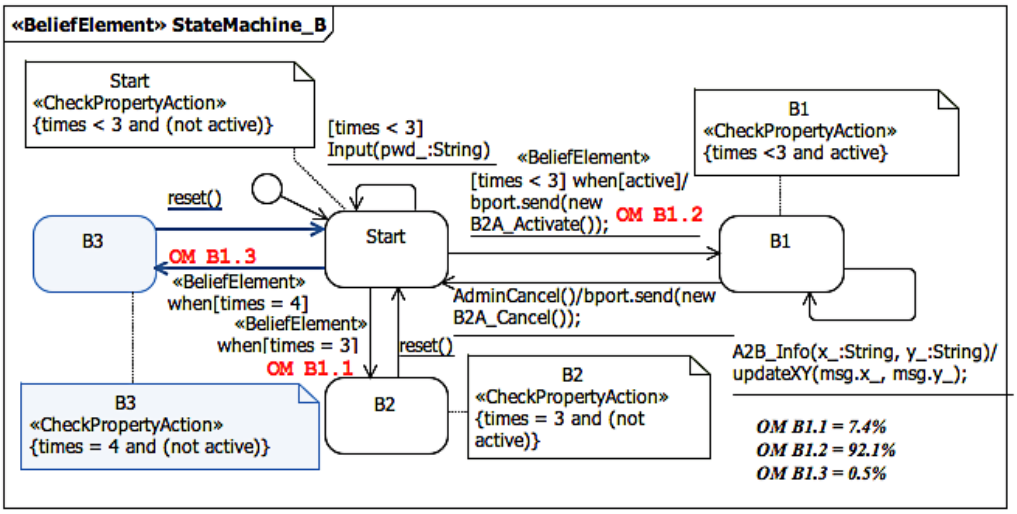


Fig. D-14. Belief State Machine of B (V2.1)

As discussed in Section 4, OCL constraints are used to specify state invariants (serving as test oracles) and guard conditions, which are for generating test data for the input parameters of associated triggers. Based on the real operational data, these OCL constraints are validated by executing the executable BM and as the result, new constraints may be added or existing ones are changed by a user based on the suggested actions provided by *UncerTolve*. For example, as shown in Fig. D-14, a new OCL constraint is added to state B3 as its state invariant.

6.2.2 Identifying Objective Uncertainty Measurements (S2A4)

UncerTolve analyzes the execution log and calculates the frequency of traversing a state or transition, based on which it defines an *objective* uncertainty measurement for the state or transition. Especially for transitions, *UncerTolve* calculates conditional probabilities of the transitions leaving from the same state. For example, as shown in Fig. D-14, the

StateMachine_B BSM of the BM (V2) contains three objective uncertainty measurements (i.e., *OM B1.1*, *OM B1.2* and *OM B1.3*). Note that *OM* means Objective Measurement and encoding of *BX.Y* means that the *X* round of the derivation of the objective uncertainty measurement for the *Y* element with «BeliefElement» applied. *OM B1.2=92.1%* implies that based on the real operational data, the probability of transiting from *Start* to *B1* via the transition is *92.1%*. Note that the subjective uncertainty measurement for this transition was initially defined as '*Unlikely*' by the modeler (Fig. D-6). In this case, the objective uncertainty measurement conforms to the subjective uncertainty measurement. In the case that a nonconformity is observed, *UncerTolve* alerts the modeler, but the evolving process of the models continues, as in steps S3 and S4, the objective uncertainty measurements might be updated, which provides more evidence to the modeler. The modeler can then decide whether or not to adjust her/his belief on the subjective uncertainty measurement in the next or future rounds of S2. Notice that more real operational data used in the evolving process leads to the higher precision of derived objective uncertainty measurements.

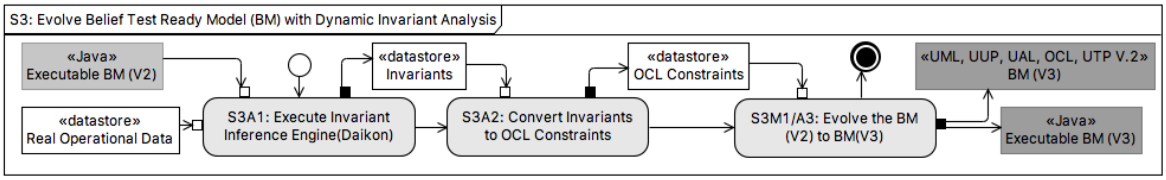
Intermediate versions of the subjective and objective uncertainty measurements can be saved such that different types of analyses can be performed and eventually advanced test generation strategies can be derived, which is one of the items of our future work.

6.3 Evolve Belief State Machines with Dynamic Invariant Analysis (S3)

In the first step (S3A1), *UncerTolve* executes the Executable BM (V2), together with the real operational data in the Daikon tool, which produces a set of invariants (Fig. D-15). In S3A2, *UncerTolve* automatically converts Daikon invariants into OCL constraints. The obtained OCL constraints are then taken as the input of S3M1/A3 to evolve the BM (V2) to BM (V3). *UncerTolve* implements a set of heuristics for this step (see details in Appendix D), some of which are listed below as examples:

Problem 1. If an invariant inferred by Daikon supersedes an existing constraint, then there are three options for the modeler to manually evolve the models: 1) keep the original constraint, 2) split the original constraint such that one or more states (transitions) are newly introduced, or 3) keep the original state (transition) but update the constraint.

Problem 2. If an invariant inferred by Daikon subsumes a set of existing constraints (named as *EConstraints*), there are three options for the modeler to manually evolve the models: 1) keep things unchanged (if the invariant inferred by Daikon is irrelevant), 2) merge the invariant inferred by Daikon with a set of existing states and transitions, corresponding to *EConstraints*, 3) create a composite state to group a set of existing states and transitions that are associated to *EConstraints*.



* S3A1 – manual action; S3A2 -- automated action; S3M1/A3 – semi-automated action

Fig. D-15. The Structured Activity of Evolve BM with Dynamic Invariant Analysis

In the running example, the input of S3 is Fig. D-14, and the output of S3 is Fig. D-16. In Fig. D-16, newly added and changed model elements are highlighted as green. Note that in the figure that state *B4* is newly introduced to the *StateMachine_B* BSM of the BM (V3). As a result, two transitions are added between *B4* and *Start*. Introducing the transition from *Start* to *B4* leads to the updates of a list of objective uncertainty measurements: *OM B1.1*-*OM B1.3*. This is because the sum of the objective uncertainty measurements for all the four transitions leaving state *Start* (to states *B1*, *B2*, *B3*, and *B4*) is *100*%; therefore, introducing a new transition triggers the change of *OM B1.2* (*=92.1%* in Fig. D-14) to *OM B2.2* (*=91.0%* in Fig. D-16). The objective uncertainty measurement for the newly added transition from *Start* to *B4* is calculated as: *OM B2.4 = OM B1.2 – OM B2.2 = 92.1% - 91.0% =1.1%*. The rationale behind the calculation is that in S3, *UncerTolve*, based on the real operational data, evolves the state invariant of *B1* by adding clause '*a.active*' to it, which leads to the discovery of the new state *B4* (whose state invariant contains the clause '*not a.active*', the negation of the newly added clause of *B1*'s state invariant). Therefore, *OM B2.2* and *OM B2.4* are the results of the splitting of *OM B1.2*. As also shown in Fig. D-16, the state invariants of states *B2* and *B3* are also updated in S3 by introducing the same clause '*not a.active*' to each of the invariant. The objective uncertainty measurements of *OM B1.1* and *OM B1.3* remain unchanged.

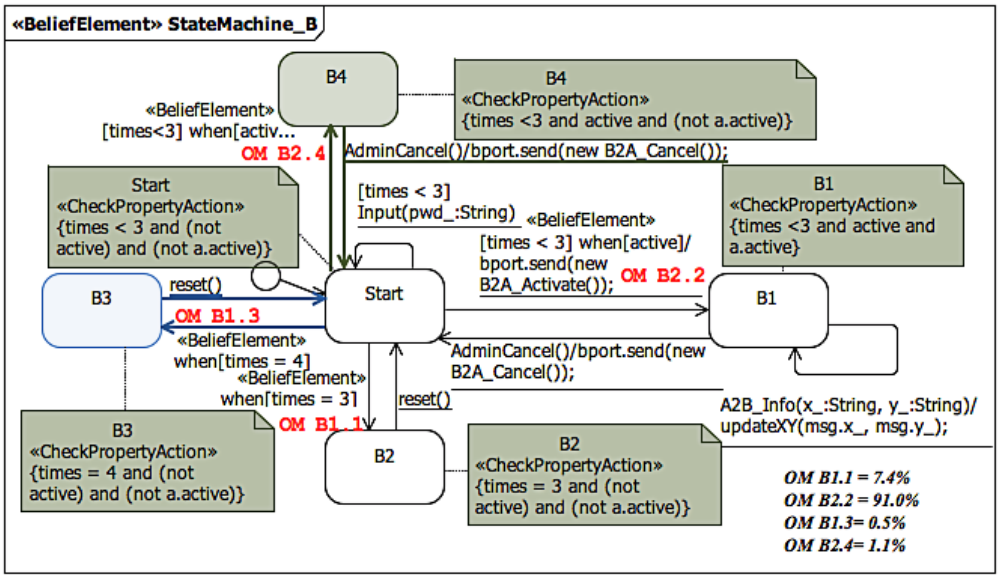


Fig. D-16. Belief State Machine of *B* (V3.1)

7 Evaluation

In this section, we present the evaluation of *UncerTolve* as a proof-of-concept using the industrial case study available to us as part of the project. The case study is called GeoSports (GS) from the healthcare domain provided by Future Position X, Sweden[20]. The GeoSports case study is about monitoring Bandy players for their performance and health conditions during the game for early intervention and prevention. Coaches use data produced by the GeoSports system to improve the performance of individual players and the team together. We had access to real operational data of five real games that were used to evaluate *UncerTolve*. The first versions of the BMs with uncertainty were developed with *UncerTum,* together with the industrial partner during the four workshops hosted at its site. Below, we present the results of evaluation according to each key activity (i.e., S1, S2 and S3, Section 7.1 to Section 7.3). Section 7.4 presents the results of the overall validation of the final evolved models. Section 7.5 presents discussion and experiences. In 7.5, we report the effort required to build the test ready model of the GS case study, and the possibility of adopting UncerTolve in a commercial tool setting. The threats to validity are discussed in Section 7.7.

[20] www.fpx.se

7.1 Results of Creating BM and DM (S1)

Table D-3 presents the descriptive statistics of the initial versions (V1) of BMs and driver models for the case study. The #C column shows the total number of classifiers (including classes, components, signals, interfaces and data types) defined in a BM/DM. The #R column presents the total number of relationships among the classifiers such as associations and compositions. The #RP column presents the total number of signal receptions specified in all the class diagrams of a BM/DM. Similarly, for the composite structure diagram (CSD) developed for a BM, we present the total number of ports (#P) and connectors (#CN). For the state machines, we present the total number of states (#S) and transitions (#T). The #BE column presents the total number of model elements in a BM, where the «BeliefElement» stereotype was applied. For each driver model, we present the total number of classes and components (#C), states (#S) and transitions (#T).

Based on the descriptive statistics shown in Table D-3, GS has the belief model with 62 classifiers, 56 relationships and 37 signal receptions in the class diagrams, 10 ports and 11 connectors in the composite structure diagram, and 82 states and 106 transitions in all the state machines of its BM.

Table D-3. Descriptive statistics of the initial BMs (V1)* of the GS case study

#	Class Diagram			Composite Structure Diagram		State Machine		#BE
	#C	#R	#RP	#P	#CN	#S	#T	
Belief Model (BM)	62	56	37	10	11	82	106	49
Driver Model (DM)	1	0	0	2	NA	5	11	NA

C: Classifiers, R: Relationships, RP: Signal Receptions, P: Ports, CN: Connectors, BE: Belief Elements

7.2 Results of Validation and Evolution via Model Execution (S2)

Table D-4 summarizes the results of S2 for the GS case study. We provide the total number of missing model elements (#MS), incorrect model elements (#IN), and redundant model elements (#RD). In addition, we report the total number of errors discovered in the semantics of the models (#SM). We report these descriptive statistics for the model elements of the BSMs of a BM: states (S), transitions (T), and elements with «BeliefElement» (BE) applied. Similarly, we report the statistics for the DSM of a BM, communications between the BSMs and the DSM and vice versa (BSM2DSM and DSM2BSM). In addition, we present the percentage of the elements evolved as compared to V1 in the % row with the

following formula: (#MS - #RD)/(#V1 + #MS - #RD), where #V1 is the total number of the model elements of a BM V1 (the initial version of the BM, comparison baseline).

Table D-4. Results of BM V2 and DM V2*

	Belief Model						Driver Model			
	BSM			BSM2DSM			DSM		DSM2BSM	
	#BE	#S	#T	#RP	#P	#CN	#S	#T	#RP	#P
#Missing	9	11	12	1	0	1	0	0	0	0
#Incorrect	0	3	3	2	1	2	0	1	0	0
# Redundant	0	1	2	1	0	0	0	0	0	0
#Semantic Problems	0	2	1	0	0	0	1	1	0	0
%	37%	11%	9%	0%	0%	8%	0%	0%	0%	0%

* BSM: Belief State Machine, DSM: Driver State Machine, RP: Signal Receptions, P: Ports, CN: Connectors, BE: Belief Elements, BSM2DSM: BSM to DSM communication, DSM2BSM: DSM to BSM communication, MS: Missing Model Elements, IN: Incorrect Model Elements, RD: Redundant Model Elements, SM: Semantic Problems, %: Percentage of evolved elements as compared to V1.

As it can be seen from Table D-4, *UncerTolve* evolved 37% of the belief elements, 11% and 9% of states and transitions in the BM V2 as compared to the BM V1. Notice that the loop inside the S2 activity (S2A1→S2A2→S2M1→S2A3→S2A1, Fig. D-14) was executed seven times until no further errors were discovered. In addition, 8% of connectors for enabling the communications from the BSMs to the DSM (BSM2DSM) were evolved as shown in Table D-4.

Table D-4 also presents the errors discovered in the BMs and DMs of GS. For example, as shown in Table D-4 (#SM row, DSM column in GS block), *UncerTolve* found two semantic errors in its DSM, one error state, and one error transition. These two semantic errors are located in the UAL code in the entry/do/exit activity of the error state and the effect of the error transition. Notice that since the semantic errors were located in the UAL code of the DSM, it does not result in the evolution of any BM model element. This is why the % row in the DSM column for GS shows 0% for both #S and #T.

7.3 Results of Dynamic Inference (S3)

Table D-5 shows the results of activity S3 for the case study. The #EP column presents the total number of model elements in the BSMs of a BM. These model elements were the points of evolution (e.g., State S4 in Fig. D-16). The #RF column presents the total number of refined model elements (e.g., state invariants in OCL and belief elements). The #ES column presents the total number of states that were newly added to or deleted from BSM

V3 as the result of evolution. The #ET column represents the total number of transitions that were added to and deleted from BSM V3 as the result of evolution. Finally, the #EB column represents the total number of belief elements that were added to or deleted from BSM V3. The % row for #EP provides the percentage of model evolution points as compared to the total number of model elements in BSM, i.e., #EP/(#S+#T). Similarly, the percentage of #RF is calculated as #RF/(#S+#T). For ES, the percentage is calculated as #ES/#S, for ET as #ET/#T, and for EB as #EB/(#S+#T). In these formulas, #S and #T represent the total number of states and transitions in BSM V3.

Table D-5. Results of the evolution of BSM V3*

Model Element Type	# Evolution points	# Modified/ Refined Elements	# Evolved States	# Evolved Transitions	# Evolved Belief Elements
#	5	56	8	18	32
%	2%	27%	8%	13%	29%

*#Evolved States: the total number of evolved states excluding the modified ones, #Evolved Transition: total number of evolved transitions excluding the modified ones, #EB: total number of evolved belief elements excluding the modified ones

As shown in Table D-5, *UncerTolve* identified 5 model elements (2%) that can be evolved, whereas 27% of the existing model elements were refined. In the case of states, transitions, and belief elements, 8% of states, 13% of transitions, and 29% of belief elements were added/deleted to BSM V3 as compared to V2.

7.4 Overall Validation

Once the evolved version V3 of the BSMs was obtained after the S3 activity, we verified it with the real operational data by performing the S2 activity once again. Results are shown in Table D-6. As one can see from the table, we found 11 validation problems in belief elements, whereas we discovered 3 validation problems with states and 2 with transitions. In total, we verified 99 belief elements. For states, we verified in total 100 states. Similarly, for transitions, we verified in total 134 transitions, but we couldn't verify 5 transitions once again due to unavailability of real operational data.

Table D-6. Results of the validation of BSM V3*

Model Element Type	#Belief Elements	#States	#Transitions
#Missing	0	0	0
#Incorrect	0	0	0

#Redundant	11	0	0
#Semantic Problems	0	3	2
#Model Elements	99	100	134

Table D-7. Overall results of the evolution across the versions (%)

Model Element Type	% Belief Elements	%States	%Transitions	%Evolution Points
V1				
V2	37%	11%	9%	19%
V3	29%	8%	13%	11%
V3'	-11%	0%	0%	1%
M= (#V3'-#V1)/#V3'	51%	18%	21%	

* V3': Verified version of V3 with S2, M is (#V3'-#V1)/#V3', - means not applicable.

Table D-7 shows the results of the percentage increase in the number of evolved model elements of BM across the three versions (V1 to V3). In addition, we also show the percentages for the verified version of V3, i.e., V3'. The last column shows the mean percentage of increase in the number of model elements in V3' as compared to V1 and is calculated as M=(#V3'-#V1)/#V3', where #V3' is the number of model elements in V3' and #V1 is the number of model elements in V1. For EP, we also show the mean percentage of increase in the evolution points in BM from V1 to V2, from V2 to V3 and from V3 to V3'.

As shown in Table D-7, in the stage of V3', 51% of belief elements, 18% of states, and 21% of transitions were evolved as compared to the first version (V1). For EP, 19% of new evolution points were discovered in V2, 11% in V3, and 1% in V3'.

7.5 Effort to Build Belief Test Ready Models and Adoption of UncerTolve

The BMs of GS were initially built by Simula researchers (the first three authors of the paper). These models were further confirmed with the industrial partner (last author of the paper). First, the first author (second year Ph.D. candidate) created the first version of the models, which were iteratively discussed with the second (a senior scientist) and third (a chief scientist) authors. Second, two workshops (2 days each) were held to present and discuss the models with the industrial partner to check their conformance with real scenarios. Third, the Simula researchers modified the models and as a result, the final version of models was produced that was used as input of *UncerTolve* (Fig. D-11). Table D-8 shows the rough estimate of efforts for developing the models and presenting them to the industrial partner.

We classify effort in terms of how much time it took to build the models using standard UML notations and additional effort to apply various profiles and model libraries defined in

UncerTum. As shown in Table D-8, for standard Class/Composite structure diagrams, it took 37.5 hours (about a week), whereas it took additional 3.5 hours to apply *UncerTum* profiles and model libraries. For standard UML state machines, it took 52.5 hours and an additional 12.5 hours for *UncerTum* modeling. The last column shows additional effort required with *UncerTum* as compared to standard UML, i.e., roughly 15%. The last row of Table D-8 shows the effort we spent to present the models to our industrial partner.

Table D-8. Efforts in terms of time (hours) to develop and present BMs

	Class/Composite Structure Diagrams		**State Machine**		**% of Time**
	Standard UML Modeling	UncerTum Modeling	Standard UML Modeling	UncerTum Modeling	
Effort to develop	37.5	3.5	52.5	12.5	15%
Effort to present	7.5		15		-

In the project [10], we have a dedicated tool vendor (Easy Global Market[21]) responsible for implementing research results including *UncerTolve* into Smartesting's commercial model-based testing tool called CertifyIt[22] and transfer of the results to the industrial partners. Such adoption of the *UncerTum*, *UncerTest*, and *UncerTolve* is on-going and will be completed by the end of the project.

7.6 Discussion and Experiences

In this section, we present discussion and our experiences of applying *UncerTolve* to the industrial case study, based on the results presented in Sections 7.1-7.4.

Based on our experience of designing drivers for model execution and evolution, we discovered that the design of a driver is highly dependent on the characteristics of a CPS. For example, in our case, we have no direct access to its testing API or internal states. In addition, GS doesn't provide feedback to its users, i.e., Bandy players. It only records the readings from the Bandy players and transmits these via radio connections to the central system, where these data are processed. Because of these two characteristics of the CPS, the driver for GS was simpler since there was less information available for GS. In addition, the feedback from the CPS to the driver was not required to be modeled in GS. This might not

21 www.eglobalmark.com
22 www.smartesting.com/en/certifyit/

be the case for other CPS case studies where we may have direct access to testing APIs and there is feedback from CPS, which will consequently lead to complex driver design.

In our case study, time events were used in models to capture timing aspects. Consequently, this had an impact on designing BMs, DMs and model evolution. However, GS only sampled data after a fixed interval of time and thus the design of BMs was simpler, which may not be the case for other case studies that have much more complex timing constraints.

In terms of the generalization of *UncerTolve*, theoretically speaking, as long as BMs of a CPS is specified in *UncerTum* [9] (a generic modeling methodology to create BMs of CPSs with subjective uncertainty) and real operational data are available, it is applicable to any case study. Our proposed modeling methodology (reported in Section 6.1) to create executable BMs is also generic and can, therefore, be tailored. In addition, our heuristics rules to update models based on the results of validation of executable models (Section 6.2), the process of calculating and abstracting objective uncertainty and reflecting them in BMs (Section 6.2.2), and rules to evolve BMs (Section 6.3) are not specific to any case study and are thus generic. At its current state, we assessed *UncerTolve* with one CPS case study from the EU project as a proof-of-concept. In order to provide further evidence related to the generalization of *UncerTolve*, we indeed need to conduct additional case studies, which will be the focus of our near future work.

7.7 Threats to Validity

Internal validity threats in our context are due to the use of existing tools, including Daikon and IBM Rational Simulation Toolkit. Notice that Daikon has been extensively used in the literature for dynamic inference of invariants as we discussed in Section 2 and thus the chances of results being impacted by its use are minimum. IBM Rational Simulation Toolkit is a commercial product that we used for model execution and has a well-tested implementation. Therefore, it is highly unlikely that the results were impacted by its use as well. As part of an academic initiative by IBM, we were able to use fully functional version free of any cost.

Currently, we evaluated *UncerTolve* with only one industrial case study; however, to generalize the results, *UncerTolve* must be evaluated with other case studies. We plan to

conduct additional industrial CPS case studies in addition to using the same case study with additional real operational data.

8 Conclusion

Given that Cyber-Physical Systems (CPSs) are tested with the assumptions on its internal behavior, its operating environment, and potential deployments, it is necessary that belief test ready models (BMs) developed to test the CPSs are continuously evolved using their real operational data including observed uncertainties. Such evolved models can be used to generate additional test cases to be executed on the current and future deployments of the same CPS. To this end, we proposed a test ready model evolution framework called *UncerTolve*. The framework was specially designed and developed to evolve BMs of CPSs with explicitly captured subjective uncertainty. Our aim is to not only improve the quality of BMs and evolve captured uncertainty, but also potentially discover unspecified uncertainty.

UncerTolve used several methods to evolve the models and uncertainty measurements including validation and evolution using model execution with real operational data collected from the application of a CPS and evolving constraints with a machine learning technique implemented in Daikon—dynamic invariant detection tool based on real operational data. *UncerTolve* was evaluated as a proof-of-concept with one industrial CPS case study from the healthcare domain, where we managed to evolve 51% of belief elements, 18% of states, and 21% of transitions. In the future, we are planning to use the evolved models to generate additional test cases by defining new test strategies focusing on the evolved parts of BM. Such test strategies will be implemented in our uncertainty-based test case generation and minimization framework called *UncerTest*. In addition, we are planning to conduct further case studies to evaluate *UncerTolve*.

Acknowledgment

This research was supported by the EU Horizon 2020 funded project (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, RCN funded

MBT4CPS project, RCN funded Certus SFI and the EU COST action MPM4CPS.

References

[1] M. Boshernitsan, R. Doong, and A. Savoia, “From daikon to agitator: lessons and challenges in building a commercial tool for developer testing,” in Proceedings of the 2006 international symposium on Software testing and analysis, Portland, Maine, USA, 2006, pp. 169-180.

[2] S. Hangal, N. Chandra, S. Narayanan, and S. Chakravorty, “IODINE: a tool to automatically infer dynamic invariants for hardware designs,” in Proceedings of the 42nd annual Design Automation Conference, Anaheim, California, USA, 2005, pp. 775-778.

[3] C. Ackermann, R. Cleaveland, S. Huang, A. Ray, C. Shelton, and E. Latronico, "Automatic Requirement Extraction from Test Cases," *Runtime Verification: First International Conference, RV 2010, St. Julians, Malta, November 1-4, 2010. Proceedings*, H. Barringer, Y. Falcone, B. Finkbeiner, K. Havelund, I. Lee, G. Pace, G. Roşu, O. Sokolsky and N. Tillmann, eds., pp. 1-15, Berlin, Heidelberg: Springer Berlin Heidelberg, 2010.

[4] M. D. Ernst, J. Cockrell, W. G. Griswold, and D. Notkin, “Dynamically discovering likely program invariants to support program evolution,” *IEEE Transactions on Software Engineering*, vol. 27, no. 2, pp. 99-123, 2001.

[5] D. B. Rawat, J. J. Rodrigues, and I. Stojmenovic, *Cyber-physical systems: from theory to practice*: CRC Press, 2015.

[6] S. Sunder, "Foundations for Innovation in Cyber-Physical Systems," in Proceedings of the NIST CPS Workshop, Chicago, IL, USA.

[7] E. Geisberger, and M. Broy, *Living in a networked world: Integrated research agenda Cyber-Physical Systems (agendaCPS)*: Herbert Utz Verlag, 2015.

[8] S. Hangal, and M. S. Lam, "Tracking down software bugs using automatic anomaly detection," in Proceedings of the 24th International Conference on Software Engineering. ICSE 2002. pp. 291-301.

[9] M. Zhang, S. Ali, T. Yue, and R. Norgre, *An Integrated Modeling Framework to Facilitate Model-Based Testing of Cyber-Physical Systems under Uncertainty,*

Technical report 2016-02, Simula Research Laboratory, 2016; https://www.simula.no/publications/integrated-modeling-framework-facilitate-model-based-testing-cyber-physical-systems.

[10] S. Ali, and T. Yue, "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.

[11] M. Zhang, S. Ali, T. Yue, and M. Hedman, *Uncertainty-based Test Case Generation and Minimization for Cyber-Physical Systems: A Multi-Objective Search-based Approach,* Technical report 2016-13, Simula Research Laborabory, 2016; https://www.simula.no/publications/uncertainty-based-test-case-generation-and-minimization-cyber-physical-systems-multi.

[12] M. Daun, J. Brings, T. Bandyszak, P. Bohn, and T. Weyer, “Collaborating multiple system instances of smart cyber-physical systems: a problem situation, solution idea, and remaining research challenges,” in Proceedings of the First International Workshop on Software Engineering for Smart Cyber-Physical Systems, Florence, Italy, 2015, pp. 48-51.

[13] OMG, "Unified Modeling Language (UML)," June 2015.

[14] O. M. Group, "Object Constraint Language (OCL)," February 2014.

[15] M. Zhang, B. Selic, S. Ali, T. Yue, O. Okariz, and R. Norgren, "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.

[16] OMG, "UML Testing Profile," April, 2013.

[17] C. Csallner, N. Tillmann, and Y. Smaragdakis, “DySy: dynamic symbolic execution for invariant inference,” in Proceedings of the 30th international conference on Software engineering, Leipzig, Germany, 2008, pp. 281-290.

[18] I. Krka, Y. Brun, D. Popescu, J. Garcia, and N. Medvidovic, "Using dynamic execution traces and program invariants to enhance behavioral model inference," in 2010 ACM/IEEE 32nd International Conference on Software Engineering. pp. 179-182.

[19] T. Berg, B. Jonsson, and H. Raffelt, "Regular Inference for State Machines Using Domains with Equality Tests," *Fundamental Approaches to Software Engineering: 11th International Conference, FASE 2008, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2008, Budapest, Hungary, March 29-April 6, 2008. Proceedings*, J. L. Fiadeiro and P. Inverardi, eds., pp. 317-331, Berlin, Heidelberg: Springer Berlin Heidelberg, 2008.

[20] L. David, M. Shahar, and K. Siau-Cheng, “Mining modal scenario-based specifications from execution traces of reactive systems,” in Proceedings of the twenty-second IEEE/ACM international conference on Automated software engineering, Atlanta, Georgia, USA, 2007.

[21] D. Lo, and S. Maoz, “Scenario-based and value-based specification mining: better together,” *Automated Software Engineering,* vol. 19, no. 4, pp. 423-458, 2012.

[22] I. Beschastnikh, Y. Brun, J. Abrahamson, M. D. Ernst, and A. Krishnamurthy, “Using Declarative Specification to Improve the Understanding, Extensibility, and Comparison of Model-Inference Algorithms,” *IEEE Transactions on Software Engineering,* vol. 41, no. 4, pp. 408-428, 2015.

[23] G. Carlo, M. Andrea, and M. Mattia, “Synthesizing intensional behavior models by graph transformation,” in Proceedings of the 31st International Conference on Software Engineering, 2009.

[24] I. Krka, Y. Brun, and N. Medvidovic, "Automatic mining of specifications from invocation traces and method invariants," in Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering. pp. 178-189.

[25] D. Lo, L. Mariani, and M. Pezzè, "Automatic steering of behavioral model inference," in Proceedings of the the 7th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering. pp. 345-354.

[26] L. Davide, M. Leonardo, and P. Mauro, “Inferring state-based behavior models,” in Proceedings of the 2006 international workshop on Dynamic systems analysis, Shanghai, China, 2006.

[27] D. Lorenzoli, L. Mariani, and M. Pezzè, "Automatic generation of software behavioral models," in Proceedings of the 30th international conference on Software engineering. pp. 501-510.

[28] P. Tonella, C. D. Nguyen, A. Marchetto, K. Lakhotia, and M. Harman, "Automated generation of state abstraction functions using data invariant inference," in Automation of Software Test (AST), 2013 8th International Workshop on. pp. 75-81.

[29] I. Beschastnikh, Y. Brun, M. D. Ernst, and A. Krishnamurthy, "Inferring models of concurrent systems from logs of their behavior with CSight," in Proceedings of the 36th International Conference on Software Engineering. pp. 468-479.

[30] T. Berg, B. Jonsson, and H. Raffelt, "Regular Inference for State Machines with Parameters," *Fundamental Approaches to Software Engineering: 9th International Conference, FASE 2006, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2006, Vienna, Austria, March 27-28, 2006. Proceedings*, L. Baresi and R. Heckel, eds., pp. 107-121, Berlin, Heidelberg: Springer Berlin Heidelberg, 2006.

[31] O. Tony, H. Michael, F. Sebastian, H. Armand, P. Marc, B. Ivan, and B. Yuriy, “Behavioral resource-aware model inference,” in Proceedings of the 29th ACM/IEEE international conference on Automated software engineering, Vasteras, Sweden, 2014.

[32] G. Mark, and S. Zhendong, “Javert: fully automatic mining of general temporal properties from dynamic traces,” in Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of software engineering, Atlanta, Georgia, 2008.

[33] N. Walkinshaw, R. Taylor, and J. Derrick, “Inferring extended finite state machine models from software executions,” *Empirical Software Engineering*, pp. 1-43, 2015.

[34] I. H. Witten, and E. Frank, *Data Mining: Practical machine learning tools and techniques*: Morgan Kaufmann, 2005.

[35] I. Beschastnikh, Y. Brun, S. Schneider, M. Sloan, and M. D. Ernst, "Leveraging existing instrumentation to automatically infer invariant-constrained models," in

Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering. pp. 267-277.

[36] O. Raz, P. Koopman, and M. Shaw, "Semantic anomaly detection in online data sources," in proceedings of the 24th International Conference on Software Engineering. pp. 302-312.

[37] J. Yang, D. Evans, D. Bhardwaj, T. Bhat, and M. Das, "Perracotta: mining temporal API rules from imperfect traces," in Proceedings of the 28th international conference on Software engineering. pp. 282-291.

[38] S. Hangal, and M. S. Lam, "Tracking down software bugs using automatic anomaly detection," in Proceedings of the 24th international conference on Software engineering. pp. 291-301.

[39] "U-RUCM: Specifying Uncertainty in Use Case Models," accessed 2017; http://zen-tools.com/rucm/U_RUCM.html.

[40] OMG, "UML Profile For MARTE: Modeling And Analysis Of Real-Time Embeded Systems," June, 2011.

[41] "JGrapht," accessed 2016; http://jgrapht.org/.

[42] B. Liu, *Uncertainty theory*: Springer, 2015.

[43] "Eclipse OCL," accessed 2016; http://www.eclipse.org/modeling/mdt/?project=ocl - ocl.

[44] S. Ali, M. Z. Iqbal, A. Arcuri, and L. C. Briand, “Generating Test Data from OCL Constraints with Search Techniques,” *IEEE Transactions on Software Engineering*, vol. 39, no. 10, pp. 1376-1402, 2013.

[45] S. Ali, M. Z. Iqbal, A. Arcuri, and L. Briand, "A Search-Based OCL Constraint Solver for Model-Based Test Data Generation," in 2011 11th International Conference on Quality Software. pp. 41-50.

[46] "IBM RSA Simulation Toolkit," accessed 2016; http://www-03.ibm.com/software/products/en/ratisoftarchsimutool.

Paper E

Uncertainty-wise Test Case Generation and Minimization for Cyber-Physical Systems: A Multi-Objective Search-based Approach

Man Zhang, Shaukat Ali, Tao Yue

Journal paper that has been submitted to ACM Transactions on Software Engineering and Methodology (TOSEM)

Abstract

Cyber-Physical Systems (CPSs) typically operate in highly indeterminate environmental conditions, which require the development of testing methods that must explicitly consider uncertainty in test design, test generation, and test optimization. Towards this direction, we propose uncertainty-wise test case generation and test case minimization strategies that rely on test ready models explicitly specifying subjective uncertainty. We propose two test case generation strategies and four test case minimization strategies based on Uncertainty Theory and multi-objective search. These strategies include a novel methodology for designing and introducing indeterminacy sources in the environment during test execution and a novel set of uncertainty-wise test verdicts. We performed an extensive empirical study to select the best algorithm out of eight commonly used multi-objective search algorithms, for each of the four minimization strategies, with five use cases of two industrial CPS case studies. The minimized set of test cases obtained with the best algorithm for each minimization strategy were executed on the two real CPSs. The results showed that our best test strategy managed to observe 51% more uncertainties due to unknown indeterminate behaviors of the physical environment of the CPSs as compared to the rest of the test strategies. Also, the same test strategy managed to observe 118% more unknown uncertainties as compared to the unique number of known uncertainties.

CCS Concepts. Software and its engineering → Software creation and management → Software verification and validation → Software defect analysis → Software testing and debugging.

Keywords. Uncertainty, Cyber-Physical Systems, Test Case Generation, Test Case Minimization, Multi-Objective Search, Uncertainty Theory.

1 Introduction

Cyber-Physical Systems (CPSs) are destined to face uncertainty in their operation due to close interactions with the physical environment [1]. Thus, classical testing methods (e.g., regression testing [2], conformance testing [3, 4]) must be extended to consider uncertainty explicitly. There exist a few methods in the literature that explicitly take uncertainty into

account while designing methods for testing CPSs [5, 6]. We present, in this paper, one such work but mainly focus on uncertainty-wise test case generation and minimization.

Our test case generation and minimization approaches are model-based, in the sense that these rely on test ready models explicitly specifying *subjective* uncertainty, which is defined as "lack of knowledge" [7, 8] about the expected behavior of a CPS in the presence of uncertainty in its operating environment. Such test ready models are developed with the Uncertainty Modeling Framework (UncerTum) [9, 10], which defines a set of UML Profiles (e.g., the UML Uncertainty Profile (UUP)) and model libraries. With UncerTum, one can create test ready models, called *Belief Test Ready Model*s (BMs). These models are composed of two types of UML diagrams: 1) *Belief Class Diagrams* (BCDs) capturing testing interfaces (e.g., observable states and operations to send stimulus) and 2) *Belief State Machines* (BSMs) modeling the expected behavior of a CPS with explicitly captured *subjective* uncertainty. Note that both BCDs and BSMs are standard UML class diagrams and state machines with stereotypes from the UUP applied.

We developed two test case generation strategies, named as *All Simple Belief Paths* (No Loops) and *All Specified Length Belief Paths* on BSMs. These two strategies are inspired from the ones reported in [11], but are extended for BSMs and considered various uncertainty aspects such as the number of uncertainties in a test path and overall uncertainty of a test path based on Uncertainty Theory [12]. Moreover, we take into account the advanced features of standard UML state machines such as composite states, submachine states, and orthogonal regions. Using the tool developed for our approach, test cases satisfying a selected test case generation strategy can be automatically and systematically generated.

Depending on the complexity of a CPS and a chosen test case generation strategy, the number of generated test cases might be huge. Automatically executing all generated test cases, especially for complex CPSs, is impractical since test execution may require setting up special hardware, simulators, and emulators. Therefore, we need an approach that can minimize the number of test cases to be executed and maximize the coverage of transitions, meanwhile maximizing the following four uncertainty related objectives: 1) the number of uncertainties covered, 2) the number of unique uncertainties covered, 3) the overall uncertainty (computed based on the Uncertainty Theory [12]) of all the selected test cases,

and 4) the coverage of uncertainty space (from the Uncertainty Theory [12]). To achieve this, we decided to benefit from the commonly-applied, eight multi-objective search algorithms from the Evolutionary Algorithm, Hybrid Algorithm, and Swarm Algorithm classifications of such algorithms [13]. Also, we used Random Search (RS) to assess the complexity of the problem at hand, i.e., if our problem can be solved with RS, it means that our problem is simple and doesn't need a complicated search algorithm to solve it. Based on the above four uncertainty related objectives, we defined four uncertainty-wise multi-objective test case minimization strategies, which share the objectives of minimizing the number of test cases and maximizing the transition coverage.

To assess the cost-effectiveness of the proposed test case generation and test case minimization strategies, we performed an empirical evaluation using two industrial case studies: GeoSports (GS) [14] (with one use case) and Automotive Warehouse (AW) [15] (with four use cases). Regarding the comparison across the test strategies to discover uncertainties in the behaviors of CPSs, our best strategy managed to discover 51% more uncertainties as compared to the rest of the test strategies due to unknown indeterminacy sources in the physical environments of the two industrial case studies. Also, the same test strategy observed 118% more unknown uncertainties due to unknown indeterminate behaviors of the physical environments as compared to the already known uncertainties.

The rest of the paper is organized as follows. In Section 2, we briefly summarize UncerTum [10] and the Uncertainty Theory. The overview of the proposed approach is presented 3. In Section 4, we describe details of the test case generation and minimization. The evaluation is discussed in Section 5, followed by the tool implementation (Section 6), related work (Section 7) and conclusion (Section 8).

2 Background

Section 2.1 presents the Uncertainty Modeling Framework (UncerTum) [9, 10] for developing test ready models to support Model-based Testing (MBT). Section 2.2 introduces Uncertainty Theory [12], for calculating uncertainty related objectives, and Section 2.3 presents the running example.

2.1 Uncertainty Modeling Framework (UncerTum)

UncerTum [9, 10] was proposed to develop test ready models for enabling MBT of CPSs in the presence of environmental uncertainty. UncerTum is equipped with specialized modeling notations (named as the UML Uncertainty Profile (UUP)) for specifying uncertainties. UUP is the core of UncerTum and UUP implements an uncertainty conceptual model, named as U-Model [16]. U-Model was developed to understand uncertainties in CPSs by defining, characterizing and classifying uncertainties and associated concepts (e.g., *Belief*, *BeliefStatement*, *IndeterminacySource*, *Measure*, and *Measurement*), and their relationships at a conceptual level.

UncerTum additionally defines four sets of UML model libraries: *Pattern*, *Time*, *Measure*, and *Risk* libraries, by extending the existing UML profile: Modeling and Analysis of Real-Time and Embedded Systems (MARTE) [17]. The purpose of defining these libraries is to ease the development of test ready models with uncertainty.

In summary, key notations used in UncerTum are standard UML state machines and class diagrams with UUP stereotypes and the model libraries applied. Such diagrams are referred as BMs. Details of UncerTum with examples can be found in our previous work [9, 10].

2.2 Uncertainty Theory

2.2.1 Probability Theory vs. Uncertainty Theory

Probability Theory is commonly used to measure uncertainty based on a long-run experiment [18]. However, in the context of testing, it is quite common that observed data is not ready (i.e., being “close enough to the long-run frequency” [18]) at the initial stage of a test design for enabling MBT, due to, for example, economic reasons and/or technical difficulties [18]. Therefore, Probability Theory is not ideal for measuring uncertainty in such a context to guide the testing phases, e.g., test design, test execution, and test results. Although we acknowledge that there exist testing techniques (e.g.,[19, 20]) that are built on Probability Theory that are described in the related work section of this paper.

Uncertainty Theory is an attempt for weakening the prerequisite of applying Probability Theory [18]—not having sufficient observed data for developing an uncertainty-wise MBT technique. Uncertainty Theory is defined by Liu [12] as “a branch of mathematics for modeling human uncertainty” to deal with uncertainty in the situation of lacking observed

data [18]. Notably, Uncertainty Theory has been applied to solve various problems, including optimal control [21], optimal scheduling (the train timetable problem [22]), risk assessment [23] and the maximum flow problem of the network [24]. In Uncertainty Theory, uncertainty is considered as the degree of the belief of a belief agent about a particular "thing," estimated by one or more domain experts (i.e., the belief agent) [12, 18]. This definition well fits the situation in the test design phase. Notably, our definition of uncertainty in U-Model [16] conforms to this definition, on which UncerTum was proposed. Therefore, our testing technique UncerTest being presented in this paper is established on Uncertainty Theory.

2.2.2 Summary of Uncertainty Theory

Uncertainty Theory defines a term called *Uncertainty Measure (UM)*, which captures a specific uncertainty value (a number) related to an event. This number assigns the *belief degree* [16] of a *belief agent* [16] to the event, to indicate her/his confidence about the occurrence of the event [12]. *UM* is represented as the $\mathcal{M}$ symbol. As Liu suggested in [12], $\mathcal{M}$ respects the following three axioms:

Axiom 1. (Normality) $\mathcal{M}(\Gamma) = 1$, (Γ is the universal set).

Axiom 2. (Duality) $\mathcal{M}\{\Lambda\} + \mathcal{M}\{\Lambda^c\} = 1$, where Λ shows a particular event, whereas Λ^c shows all the elements in the universal set excluding Λ.

Axiom 3. (Subadditivity) $\mathcal{M}\{\bigcup_{i=1}^{\infty} \Lambda_i\} < \sum_{i=1}^{\infty} \mathcal{M}\{\Lambda_i\}$ (every *countable* sequence of events $\Lambda_1, \Lambda_2, ...$).

Below, we define *Uncertainty Space* and the related theorem, which are relevant to our work. Readers may consult [12] for more details about the theory.

Uncertainty Space: A triplet $(\Gamma, \mathcal{L}, \mathcal{M})$, where Γ is the universal set, $\mathcal{L}$ is a σ-algebra [37] over Γ, and $\mathcal{M}$ is *UM*.

Theorem: Let $(\Gamma_k, \mathcal{L}_k, \mathcal{M}_k)$ be uncertainty spaces and $\Lambda_k \in \mathcal{L}_k$, for $k = 1, 2, \ldots n$. Then $\Lambda_1, \Lambda_2, \ldots \Lambda_n$ are always independent of each other if they are from different uncertainty spaces.

2.3 Example of the Application of UncerTum and Uncertainty Theory

This section presents a running example of the application of UncerTum and Uncertainty Theory. The example will also be used in the rest of the paper to explain various concepts.

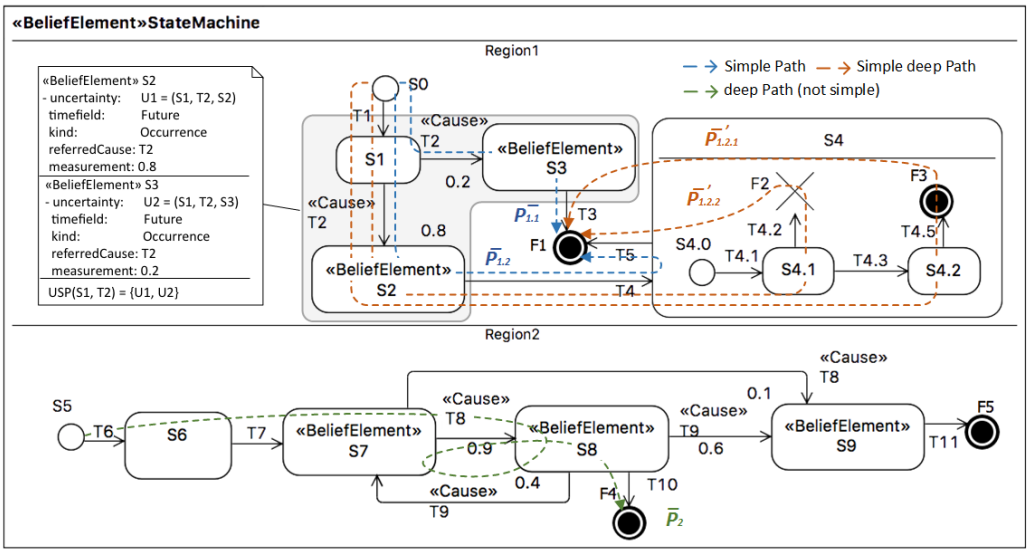


Fig. E-1. Belief State Machines (BSMs) of the Running Example

Fig. E-1 shows a standard UML state machine of a CPS with UUP stereotypes applied. The state machine is stereotyped as «BeliefElement» to indicate that it is a belief state machine, i.e., test modeler lacks complete knowledge about at least one model element in the state machine. The *S2* state also has the «BeliefElement» stereotype applied to present the situation that the test modeler («BeliefAgent») lacks the confidence that whether the CPS being model will transit to the *S2* state by the*T2* transition from the *S1* state. From State *S1*, the transition *T2* also has the chance to trigger the occurrence of State *S3*. Thus, two uncertainties are captured in this case, *U1* (*S1*, *T2*, *S2*) and *U2* (*S1*, *T2*, *S3*). To model uncertainty specified by UncerTum, the kind of uncertainty, related cause, evidence, and measurements are recommended to be specified in the testing phase [9, 10]. In terms of the state machine (event-driven) [26], the kind of uncertainty is recognized as an *occurrence*, which is caused by transition *T2*. The measurement of *Uncertainty* can be measured by the different way. In this paper, we apply *Uncertainty Theory* to measure uncertainty, which allows the modeler to specify the measurement of uncertainty from the subjective perspective of test modeler(s) according to their experience and knowledge that can be supported by evidence or not as discussed in Section 2.2. For example, *Uncertainty Measure* of *U1* is 0.8 as a measurement of uncertainty (in Fig. E-1), which implies that the test modeler

(«BeliefAgent») believes that the occurrence of the State *S2* is possible to be triggered by transition *T2* from State *S1* with the 80% confidence. Since *U1* and *U2* are oriented from the same state *S1* and triggered by the same transition *T2*, we recognize that *U1* and *U2* belong to the same uncertainty space.

In the context of the testing phase, we developed strategies to generate abstract test cases from test ready models developed with UncerTum. Accordingly, a set of uncertainty related properties, such as *Uncertainty Measure,* are calculated for each abstract test case (Section 4.1.1), and those properties can be regarded as objectives of the test case minimization using multi-objective search algorithms (Section 4.2).

3 Overview

An overview of UncerTest is presented in Fig. E-2. The only input for the test case generation is BMs developed using UncerTum (Section 2.1). Two test generation strategies are proposed in UncerTest: 1) All Simple Belief Paths (*ASiBP*): A set of all simple paths (no loops) in a BSM, each of which contains unique states and transitions; and 2) All Specified Length Belief Paths (*ASlBP*): A set of all paths in a BSM, the maximum length of each of which can be set to any positive number. Each path is an abstract test case.

For each abstract test case, UncerTest automatically calculates *UM* (Def17), based on the Uncertainty Theory (Section 2.2). Followed by that, it applies the *Uncertainty-wise Test Minimization* approach as the number of automatically generated abstract test cases is often large for any non-trivial CPS and it is practically impossible to execute all of them. Test case minimization strategies of UncerTest can be formulated as multi-objective search problems, and thus we opted for multi-objective search algorithms (e.g., NSGA-II) to address them. To reduce a number of test cases and maintain the coverage of a test ready model, all the search problems are developed, aiming to minimize the number of test cases and maximize the transition coverage. Regarding uncertainty, we futher proposed four uncertainty related objectives: 1) maximizing the average number of uncertainties covered by the selected test cases, which aims to test more defined uncertainties; 2) maximizing the average percentage of uncertainty space covered by the selected test cases, which aims to test more uncertainties from the different uncertainty space; 3) maximizing the average *UM* of the selected test cases, which aims to test the paths with high confidence; and 4) maximizing the average

number of unique uncertainties covered by selected test cases, which aims to test more different defined uncertainties. A minimized set of abstract test cases is then converted into executable test cases (Section 4.3) with the consideration of the source of uncertainties (indeterminacy source), which are executed to test a CPS.

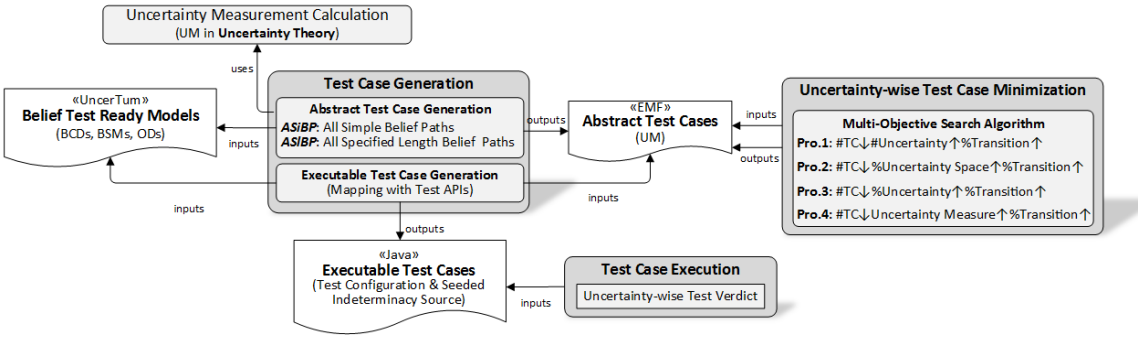


Fig. E-2. Overview of UncerTest

4 Test Case Generation and Minimization

First, we present the test case generation approach of UncerTest (Section 4.1), followed by its test minimization strategies (Section 4.2). Section 4.3 discusses the process of executable test case generation; and Section 4.4 discusses our test execution and reporting mechanisms.

4.1 Abstract Test Case Generation

UncerTest automatically generates abstract test cases from BMs, based on the test case generation strategies that are applied on BSMs. In the rest of the section, we first provide necessary definitions (Section 4.1.1), followed by the test case generation strategies (Section 4.1.2).

4.1.1 Definitions

To measure uncertainties specified by UncerTum and apply them and their measurements (Section 2.3), we formalize related concepts (e.g., *Path*, *UM*) in Table E-1, and exemplify them with the running example presented in Fig. E-1. In Table E-1, we formalized the state machine (BSM), region, states, and transitions (Def1 and Def2). To adopt Uncertainty

Theory in UncerTest to specify and calculate measurements of uncertainties, we further define *U* (Def3), *UM* (Def4) and *USP* (Def5) in terms of the elements in BSM.

Table E-1. Definitions

Def#	Name	Definitions
Def1	*R*	Suppose a region of a BSM can be represented as a tuple *R*={*is, ST, TR, FS*}, where *is* is the initial state of *R*; *ST* = $\{st_i \mid 0 < i \leq nst\}$ is the set of states (i.e., simple, choice, submachine, and composite states), each of which may have a UUP stereotype (Section 2.1) applied; *TR* = $\{tr_i \mid 0 < i \leq ntr\}$ is the set of transitions, which may have UUP stereotypes (Section 2.1) applied; and *FS* = $\{fs_i \mid 0 < i \leq nfs\}$ is the set of final points (i.e., final state, exit point, and terminate) in *R*. **Example:** The *Region 1* in the top of Fig. E-1 is an *R*.
Def2	*BSM*	A BSM can be viewed as a set of orthogonal regions [27]: *BSM* =$\{R_i \mid 0<i\leq nr, \forall R_i \perp \forall R_j,$ *iff nr>1, i≠j*$\}$. If a state is composite or a submachine, it is equivalent to a state machine. **Example:** The state machine in Fig. E-1 is a *BSM*.
Def3	*U*	*Uncertainty* (*U)* of (st_x, tr_y, st_z) is a situation whereby the belief agent does not have full confidence that st_x transits to st_z through the tr_ytransition in the BSM. **Example:** In Fig. E-1, *U1* = (*S1*, *T1*, *S2*) indicates that the belief agent does not have full confidence that *S1* will transit to *S2* by *T2*.
Def4	*UM(U)*	In Uncertainty Theory, *U* can be measured by *Uncertainty Measure* (*UM(U)*), which is a belief degree ranging from 0 to 1 and is represented as $UM(st_x, tr_y, st_z) = \mathcal{M}\{(st_x, tr_y, st_z)\}$. **Example:** In Fig. E-1, *UM* = $\mathcal{M}${(*S1*, *T2*, *S2*)}=0.8 indicates that the belief agent believes that *S1* will transit to *S2* by *T2* with a probability of 0.8.
Def5	*USP*	*Uncertainty Space (USP)* of (st_x, tr_y) is a triplet: $USP(st_x, tr_y) = (\Gamma, \mathcal{L}, \mathcal{M})$, where Γ is the universal set that contains all the options (i.e. (st_x, tr_y, st_{z1}), (st_x, tr_y, st_{z2})) of transiting from st_x via tr_y, about which a belief agent hold beliefs; $\mathcal{L}$ is a σ-algebra over Γ; and $\mathcal{M}$ is the uncertainty measure of the elements in $\mathcal{L}$. **Example:** *USP*(S1, T2) is $(\Gamma, \mathcal{L}, \mathcal{M})$, where Γ = {(S1, T2, S2), (S1, T2, S3)}; $\mathcal{L} = \{\emptyset, \{(S1, T2, S2)\}, \{(S1, T2, S3)\}, \Gamma\}$; $\mathcal{M}\{\emptyset\} = 0$, $\mathcal{M}\{(S1, T2, S2)\}$ =0.8, $\mathcal{M}\{(S1, T2, S3)\} = 0.2$ and $\mathcal{M}\{\Gamma\}$ =1.
Def6	*P*	A *Belief Path (P)* in a region of belief state machine (BSM) is a sequence of states and transitions represented as $P = (e_0, \dots, e_i, \dots, e_{np})$, where e_0 is *is* (the initial state of the region); e_{np} is an element from *FS*, $(e_i, e_{i+1}) = \{(st_x, tr_y) \mid i < np, i \text{ is even}, st_x$ is the source state of *of* $tr_y\}$; and $(e_i, e_{i+1}) = \{(tr_x, st_y) \mid i < np, i$ is odd, st_y is target state of $tr_x\}$. **Example:** As shown in Fig. E-1, $\bar{P}_{1.1}$= (*S0, T1, S1, T2, S3, T3, F1*) is a path in Region 1. In Region 2 (Fig. E-1), $\bar{P}_2$ = *(S5, T6, S6, T7, S7, T8, S8, T9, S7, T8, S8, T10, F4)*, which can be seen as a sequence of (*S5, T6, S6*), (*S6, T7, S7*), (*S7, T8, S8*), (*S8, T9, S7*), (*S7, T8, S8*) and (*S8, T10, F4*).
Def7	*Us(P)*	*Us* in a belief path *P* presents a multiset[23] [28] of uncertainties that appear along the *P*. **Example:** *U*($\bar{P}_2$) ={(*S7, T8, S8*), (*S8, T9, S7*), (*S7, T8, S8*)}.
Def8	*USP(P)/ NUSP(P)*	*USP* in a *P* $(e_0, \dots, e_i, \dots, e_{np})$ is a set of uncertainty spaces covered by it, which can be represented as: *USP(P)*=$\{USP_j(e_i, e_{i+1}) \mid j < nusp, i < np, i$ *is even*$\}$.The number of *Uncertainty Spaces* (Def5) *NUSP* in a *P* $(e_0, \dots, e_i, \dots, e_{np})$ is the number of uncertainty spaces that appear along the *P*. **Example:** *USP*($\bar{P}_2$) = {*USP(S7, T8), USP(S8, T9)*}; *NUSP*($\bar{P}_2$) *= 2*.

[23] A multiset is a generalization of the concept of a set that allows multiple instances of the multiset's elements.

Def9	*NU(P)/ NUU(P)*	The number of *Uncertainties* (Def3) *NU* of a $P = (e_0, \dots, e_i, \dots, e_{np})$ is the number of uncertainties that appear along the *P*. Further, *NUU* represents the number of unique uncertainties in the *P*. **Example:** *NU($\bar{P}_2$) =3, NUU ($\bar{P}_2$) = 2.*
Def10	*UM(P)*	Uncertainty Measure of a belief path (*UM(P)*) $(e_0, \dots, e_i, \dots, e_{np})$ is a belief degree, with which a belief agent believes that e_0 arrives e_{np} by following the sequence of $(e_1 \dots e_{np-1})$. It can be represented as $UM(P) = \mathcal{M}\{\cap_{i=0}^{(np-1)/2}\{(e_{2i}, e_{2i+1}, e_{2i+2})\}\}$. **Example:** Along the path, two uncertainty spaces are encountered: *USP(S7, T8)* and *USP(S8, T9). UM($\bar{P}_2$)* is therefore calculated as $\mathcal{M}\{\{S5, T6, S6\} \cap \dots \cap \{S8, T10, F4\}\}$. Since each $(st_i, tr_{i+1}, st_{i+2})$ is from different USPs, *UM($\bar{P}_2$)* = $\mathcal{M}\{S5, T6, S6\} \wedge \dots \wedge \mathcal{M}\{S8, T10, F4\}$ = 0.4.
Def11	*PP*	A *Parallel Belief Path* (*PP*) is a sequence of *Ps* represented as $PP = \{BP_i | 0 < i \leq npp, \forall P_i \perp \forall P_j, i \neq j\}$. **Example:** As shown Fig. E-1, $\bar{P}_{1.1}$ is a belief path in Region 1, and $\bar{P}_2$ is a belief path in Region 2. *PP_1* = $\{\bar{P}_{1.1}, \bar{P}_2\}$
Def12	$\overline{P}$	A *Simple Belief Path* ($\overline{P}$) is a sequence of unique states and transitions represented as $\overline{P} = (e_0, \dots, e_i, \dots, e_{np})$, where $\forall e_i \neq \forall e_j$ ($i \neq j$). **Example:** $\bar{P}_2$*= (S5, T6, S6, T7, S7, T8, S8, T9, S7, T8, S8, T10, F4)* is a path in Region 2, and $\bar{P}_2$ is not deep since S7 and S8 appear more than one times (Fig. E-1).
Def13	*P′*	A *Deep Belief Path (P′)* is a *P* that does not contain any composite or submachine states, which can be represented as $P' = (e_0, \dots, e_i, \dots, e_{np})$, where $\forall e_i$ is not a composite or submachine state. **Example:** In Fig. E-1. Belief State Machines (BSMs) of the Running Example , $\bar{P}_{1.2}$*= (S0, T1, S1, T2, S2, T4, S4, T5, F1)* is a belief path in Region 2, and $\bar{P}_{1.2}$ is not simple since it contains the composite state S4. Further, we flatten the composite state S4, which leads to two simple belief paths (i.e., (*S4.0, T4.1, S4.1,T4.2,F2)* and (*S4.0,T4.1,S4.1,T4.3,S4.2,T4.5,F3)*). We, thus, extend $\bar{P}_{1.2}$ with these two simple belief paths into two deep simple paths ($\bar{P}'_{1.2.1}$and $\bar{P}'_{1.2.2}$ in Fig. E-1).
Def14	*t*	A test case *t* in a state machine is a *deep* belief path (*P′*), which can be *parallel* (*PP′*) or *simple ($\bar{P}'$).* **Example:** *PBP_1* = $\{\bar{P}_{1.1}, \bar{P}_2\}$ is a test case.
Def15	*Us(t)/ NU/ NUU(t)*	*Us* in a test case presents a multiset[23] of uncertainties covered by it, calculated as: $Us(t) = \begin{cases} \cup_{i=0}^{npp} U(P_i), & t = PP' \\ U(P), & t = P' \end{cases}$. *NU/NUU* in a test case represents the total number of uncertainties/unique uncertainties covered by it, calculated as: $NU(t) = \begin{cases} \sum_{i=0}^{npp} NU(P_i) & t = PP' \\ NU(P) & t = P' \end{cases}$, $UNU(t) = \begin{cases} \sum_{i=0}^{npp} NUU(PP_i) & t = PP' \\ NUU(P) & t = P' \end{cases}$. **Example:** *Us($PP_1$)* = {(*S1, T1, S2*), (*S7, T8, S8*), (*S8, T9, S7*), (*S7, T8, S8*)}, *NU(PP_1)* = 4 and *NUU(PP_1)* = 3
Def16	*USP(t)*	*USP* in a test case presents a set of uncertainty spaces covered by it, calculated as: $USP(t) = \begin{cases} \cup_{i=0}^{npp} USP(P_i), & t = PP' \\ USP(P), & t = P' \end{cases}$. **Example:** USP(*$PP_1$*) = {USP(*S1, T1*), USP(*S7, T8*), USP(*S8, T9*)}
Def17	*UM(t)*	*UM* of a test case *t* is a number, indicating the belief degree, with which a belief agent believes that the test case will be executed successfully. *UM* is calculated as $UM(t) = \begin{cases} \wedge_i^{npp} UM(P_i) & t = PP' \\ UM(P) & t = P' \end{cases}$. **Example:** *TR($PBP_1$)* = {*T1, T2, T3, T6, T7, T8, T9, T10*}
Def18	*TR(t)*	*TR* in a test case is the set of transitions covered by it, which can be represented as *TR(t)* = $\{tr_i | 0 < i < ntr_t\}$. **Example:** *UM(PBP_1) = UM($\bar{P}_{1.1}$) ∧ UM($\bar{P}_2$) = 0.2* ∧ 0.4 = 0.2

Def19	T	A test set T is a set of test cases derived from a BSM using a test case generation strategy: $T = \{t_i \mid 0 < i \leq nt\}$.

We also designed a class diagram shown in Fig. E-3 to conceptually describe how the defined concepts are related to each other. From a BM, a test set can be generated, based on the test case generation strategies. A test set is composed of a set of test cases, which can be a path or a parallel path. A path is characterized by two properties: *isSimple* and *isDeep*. A parallel path is a special type of paths, which should be composed of at least two paths. A deep (simple) parallel path has all its contained paths being all deep (simple) paths. Each test case is a deep path. A test case can be an abstract or executable test case. For each belief path, one can obtain information such as values for *NU*, *UM*, and *TR*, as shown in Fig. E-3.

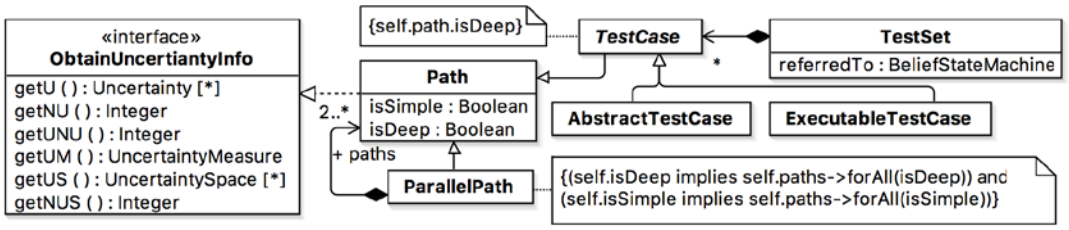


Fig. E-3. Key concepts of UncerTest and Their Relationships

4.1.2 Strategies

In the literature, some state machine-based test case generation strategies have been proposed, including *All Transitions*, *All States*, and *All Predicates* [29-32]. For UncerTest, we propose two test case generation strategies, inspired by *Prime Path Coverage* [11] and *Specified Path Coverage* presented in [11].

All Simple Belief Path Coverage (ASiBP). Test set *T* satisfies *ASiBP* on BSM if and only if any belief simple deep path $\bar{P}'$ from *initial state* to one of *final state*s in BSM is in *T*.

As said in [11], "*One useful aspect of the simple path is that any path can be created by composing simple paths*". We propose *ASiBP* to cover all minimal paths based on which any path-based coverage criterion can be defined by extending a path generated with *ASiBP* (i.e., side trips and detours [11]). The test set generated using this strategy is the cross product of all the possible *simple deep* belief paths across all the regions. For example, in Region 1, there are two simple paths: $\bar{P}_{1.1}$= *(S0, T1, S1, T2, S3, T3, F1)* and $\bar{P}_{1.2}$= *(S0, T1, S1, T2, S2, T4, S4, T5, F1)*. $\bar{P}_{1.1}$ is *deep*, but $\bar{P}_{1.2}$ is not. Further, we flatten the composite state S4, which leads to two simple paths (i.e., (*S4.0, T4.1, S4.1, T4.2, F2)* and (*S4.0, T4.1, S4.1, T4.3, S4.2,*

T4.5, F3)). We, thus, extend $\bar{P}_{1.2}$ with these two simple paths into two deep simple paths ($\bar{P}'_{1.2.1}$and $\bar{P}'_{1.2.2}$, Fig. E-1). The total number of deep paths in Region 1 is therefore three. In Region 2, there are three simple paths that are also deep. In total, the number of test cases generated with this strategy is $3 \times 3 = 9$. Table E-2 presents all generated test cases by applying *ASiBP*. Note that with *ASiBP* it is impossible to cover all uncertainties if any uncertainty is in any existing loop (Fig. E-1). For example, as shown in Fig. E-1, the uncertainty of (*S8*, *T9*, *S7*) is embedded in the side trip of (*S7*, *T8*, *S8*, *T9*, *S7*).

All Specified Length Belief Path Coverage (ASlBP). Test set *T* satisfies *ASlBP* on belief state machine (BSM) if and only if any belief simple deep path $\bar{P}'$ of length less than *specified length* from *initial state* to one of *final state*s in BSM is in *T*.

We propose *ASlBP* because it can be configured 1) for specific needs (e.g., saving cost by generating less number of test cases), 2) to subsume *All Transitions*, *All States*, and *All Predicates* when needed, 3) to generate a larger size of test set from a BM (which are more diverse in terms of attached uncertainty information) to form a better pool for test minimization, and 4) to subsume the *All Uncertainty* coverage, which we define as covering all states and transitions with uncertainty. The test set generated with this strategy consists of all possible *deep* belief paths with loops allowed, and all the lengths of these paths should not be longer than the maximum allowed length, which is configurable (as discussed above) and should be pre-defined before applying the test generation strategy. For example, one way of defining the maximum allowed length for generating paths for a region is to calculate the total number of states and transitions contained in the region. For example, in Region 2, the maximum allowed length is 15.

After applying the test case generation strategies, each abstract test case t_i in *T* (Def19) has the following associated attributes: 1) the multiset of uncertainties in t_i (*Us(t_i)*, *NU(t_i,)*, Def15);); 2) the set of uncertainty spaces in t_i (USP(t_i), Def16); 3) the uncertainty measure (*UM*) of t_i (*UM(t_i)*, Def17); and 4) the set of unique transitions (Def18) in t_i, $TR_{t_i} = \{tr_j' | 0 < \mathrm{j} < ntr_{t_i,}\}$.

Table E-2. Example of Abstract Test Case Generation*

BSM	#	Abstract Test Case	Us	UM	NUSP
***U and UM*:**	1	R1:(S0, T1, ***S1, T2, S3***, T3, F1) R2:(S5, T6, S6, T7, S7, T8.1, ***S8, T9, S9***, T11, F5)	U2 U6	0.2	2

U1=(S1, T2, S2), UM(U1) = 0.8; U2=(S1, T2, S3), UM(U2) = 0.2; U3=(S7, T8, S8), UM(U3) = 0.9; U4=(S7, T8, S9), UM(U4) = 0.1; U5=(S8, T9, S7), UM(U5) = 0.4; U6=(S8, T9, S9), UM(U6) = 0.6 ***USP***: USP(S1, T2) = {U1, U2}; USP(S7, T8) ={U3, U4}; USP(S8, T9) ={U5, U6}	2	R1:(S0, T1, ***S1, T2, S3***, T3, F1) R2:(S5, T6, S6, T7, ***S7, T8, S8***, T10, F4)	U2 U3	0.2	2
	3	R1:(S0, T1, ***S1, T2, S3***, T3, F1) R2:(S5, T6, S6, T7, ***S7, T8, S9***, T11, F5)	U2 U4	0.1	2
	4	R1:(S0, T1, ***S1, T2, S2***, T4, S4.0, T4.1, S4.1, T4.3, S4.2, T4.5, F3, T5, F1) R2:(S5, T6, S6, T7, ***S7, T8, S8, T9, S9***, T11, F5)	U1 U3 U6	0.2	3
	5	R1:(S0, T1, ***S1, T2, S2***, T4, S4.0, T4.1, S4.1, T4.3, S4.2, T4.5, F3, T5, F1) R2:(S5, T6, S6, T7, ***S7, T8, S8***, T10, F4)	U1 U3	0.2	2
	6	R1:(S0, T1, ***S1, T2, S2***, T4, S4.0, T4.1, S4.1, T4.3, S4.2, T4.5, F3, T5, F1) R2:(S5, T6, S6, T7, ***S7, T8, S9***, T11, F5)	U1 U4	0.1	2
	7	R1:(S0, T1, ***S1, T2, S2***, T4, S4.0, T4.1, S4.1, T4.2, F2) R2:(S5, T6, S6, T7, ***S7, T8, S8, T9, S9***, T11, F5)	U1 U3 U6	0.2	3
	8	R1:(S0, T1, ***S1, T2, S2***, T4, S4.0, T4.1, S4.1, T4.2, F2) R2:(S5, T6, S6, T7, ***S7, T8, S8***, T10, F4)	U1 U3	0.8	2
	9	R1:(S0, T1, ***S1, T2, S2***, T4, S4.0, T4.1, S4.1, T4.2, F2) R2:(S5, T6, S6, T7, ***S7, T8, S9***, T11, F5)	U1 U4	0.1	2
	Summary: *uncertainty coverage* = 5/6 = 83.3%, *uncertainty space coverage* = 100%, *transition coverage* = 17/18 = 94.4%				

* R1 and R2 present two parallel test paths, generated from Region 1 and Region 2, respectively.

4.2 Uncertainty-Wise Test Case Minimization

4.2.1 Problem Representation

Depending on which test case generation strategy to apply, how it is configured (for *ASlBP*) and how complex a CPS under test is, the number of generated abstract test cases can potentially be very large and it would be practically impossible to execute executable test cases generated from all of the abstract test cases within a limited time budget. It is, therefore, important to minimize the number of abstract test cases to be executed based on various attributes associated with each test case.

$T = \{t_i \mid 0 < i < nt\}$ is a test set derived from BSM using the UncerTest generation strategies, each test case t has four uncertainty related attributed (Section 4.2.2). $S = \{s_1, \ldots, s_{ms}\}$ presents a set of potential solutions, i.e., a subset of T, where ms is the total number of potential solutions and ms is calculated as 2^{nt}-1 except that the solution selects none. As the number of generated test cases increases, the search space will increase exponentially. For any test case minimization problem, the solution s contains a set of selected test cases, formalized as $T_{sub} = \{t_j' \mid 0 < j < mt, t_j' \in T\} \subseteq T$, where mt is the number of selected test cases. Each solution s is characterized by a set of values of cost and effectiveness measures. In UncerTest, we defined six objectives and four uncertainty-wise multi-objective

minimization problems with consideration of three aspects: cost, effectiveness, and uncertainty.

4.2.2 Definitions and Functions of the Six Minimization Objectives

Six minimization objectives are defined in this section.

(1) Cost Measure

O1. Percentage of Test Case Minimization (PTM)

PTM measures the percentage of the selected test cases in a solution T_{sub}, which is calculated as:

$$PTM = \frac{mt}{nt} \times 100\%$$

where nt is the number of test cases in *T;* and mt is the number of test cases in T_{sub}.

(2) Uncertainty-related Measure

O2. Average Normalized Number of Uncertainties Covered (ANU)

ANU measures the average normalized number of uncertainties covered by the selected test cases of a solution. For each test case t_i' , the number of uncertainties covered is $NU(t_i')$, which can then be normalized [1] as: $nor(NU(t_i')) = \frac{NU(t_i')}{NU(t_i')+1}$. The *ANU* for the selected test cases is calculated as:

$$ANU = \frac{\sum_{i=1}^{mt} nor(NU(t_i'))}{mt}$$

O3. Percentage of Uncertainty Space Covered (PUS)

PUS measures the percentage of the total set of uncertainty spaces of a BSM covered by the selected test cases of a solution. Suppose, the set of uncertainty space of the BSM is $USP_{SM} = \{US_i | \ 0 < i \leq nusp\}$ and the set of uncertainty spaces of the selected test cases is the intersection of the uncertainty spaces across each test case t_i' , $USP_{sub} = \bigcap_i^{mt} USP_{t_i'} = \{US_i' | \ 0 < i \leq musp\} \subseteq USP_{SM}$. *PUS* is then defined as:

$$PUS = \frac{musp}{nusp} \times 100\%\square$$

O4. Average Overall Uncertainty Measure (AUM)

AUM is the overall uncertainty measure of the selected test cases of a solution. Note that for test case t_i', $UM(t_i')$ is calculated using Uncertainty Theory (Section 4.1.1). The overall average uncertainty is thus calculated as:

$$AUM = \frac{\sum_{i=1}^{mt} UM(t_i')}{mt}$$

O5. Percentage of Unique Uncertainties Covered (PUU)

PUU measures the percentage of the total number of unique uncertainties covered by the selected test cases of a solution. Suppose that the set of unique uncertainties in a BSM is $UU_{SM} = \{U_i | \ 0 < i \leq nuu\}$ and the set of unique uncertainties of the selected test cases is the interaction of the unique uncertainties across each test case t_i' , $UU_{sub} = \bigcap_i^{mt} UU_{t_i'} = \{U_i' | \ 0 < i \leq muu\} \subseteq UU_{SM}$, then PUU is calculated as:

$$PUU = \frac{muu}{nuu} \times 100\%$$

(3) Effectiveness Measure

O6. Percentage of Transition Coverage (PTR)

PTR measures the percentage of the total number of transitions in a BSM covered by the selected test cases of a solution. According to Def1, *ntr* is the total number of transitions in a BSM. Suppose that *mtr* is the number of transitions in the selected test cases (the size of the interactions among the transition sets of each selected test case t_i' , $TR_{sub} = \bigcap_i^{mt} TR_{t_i'} = \{tr_i' | \ 0 < i \leq mtr\}$). *PTR* is calculated as:

$$PTR = \frac{mtr}{ntr} \times 100\%$$

4.2.3 Uncertainty-wise Test Case Minimization Problems

To reduce the number of test cases to execute and maximize the coverage of transitions in test ready models, *PTM* and *PTR* are the necessary objectives for test case minimization. Further, we defined the following four test case minimization problems that minimize PTM, maximize PTR, and at the same time achieve four distinct uncertainty-related concerns.

Problem 1. Search for a solution T_{sub} to achieve: 1) low *PTM*; 2) high *ANU*; and 3) high *PTR*. We defined *Problem 1* to select the minimum number of test cases to cover the maximum number of known uncertainties possible. We aim to observe the reaction of the CPS in the presence of maximum uncertainties with the minimum possible test cases.

Problem 2. Search for a solution T_{sub} to achieve: 1) low *PTM*; 2) high *PUS*; and 3) high *PTR*. We defined *Problem 2* to select the minimum number of test cases to cover at least one uncertainty from each uncertainty spaces. We aim to observe the reaction of the CPS in the

presence of uncertainties from all known uncertainty spaces with the minimum possible test cases.

Problem 3. Search for a solution T_{sub} to achieve: 1) low *PTM*; 2) high *AUM*; and 3) high *PTR*. We defined *Problem 3* to select the minimum number of test cases to maximize the coverage of the parts of the system with high degree of confidence.

Problem 4. Search for a solution T_{sub} to achieve: 1) low *PTM*; 2) high *PUU*; and 3) high *PTR*. We defined *Problem 4* to select the minimum number of test cases to maximize the coverage of different uncertainties. We aim to test the behavior of a CPS under diverse uncertainties with the minimum number of test cases.

4.3 Executable Test Case Generation

In our context, generating executable test cases from abstract test cases (Section 4.1) is mainly concerned with how to enable indeterminacy sources (i.e., sources of uncertainties) that are specified as part of the test ready models and how to generate test data.

4.3.1 Enabling Indeterminacy

Since we focus on testing a CPS in the presence of environmental uncertainties, we need to introduce uncertainties in the physical environment that lead to uncertain behaviors of the CPS. To achieve this, we need to model such environmental uncertainties (named as "Indeterminacy Sources" for being more precise) in the environment that lead to observe uncertainties in the CPS.

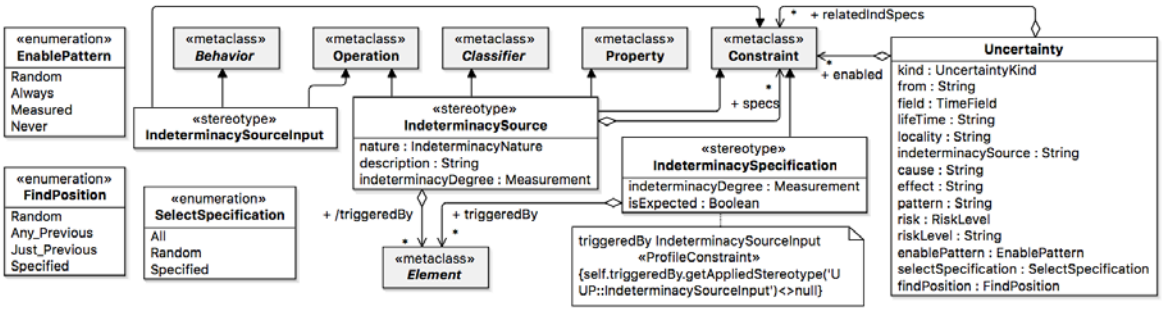


Fig. E-4. Profile Diagram of IndeterminacySource (Partial)

Fig. E-4 shows part of the UUP profile (Section 2.1) for modeling indeterminacy sources. We provide a set of options to model indeterminacy sources, e.g., as a UML Operation and a constraint specified in Object Constraint Language (OCL) [33]. An indeterminacy source

always has 1..* indeterminacy specifications, i.e., «IndeterminacySpecification» (conditions) that must be true for an indeterminacy source to occur. «IndeterminacySourceInput» specifies the action that triggers the occurrence of «IndeterminacySource».

It is possible to model these indeterminacy-related concepts in different ways. Therefore, to ease the modeling process, we summarize our recommendations for applying this part of the profile in Table E-3, based on our experience. For example, in the first situation (as described as *Smi1* in Table E-3), we recommend modeling an indeterminacy source as a UML *Property*, when states of a CPS or its environment can be directly accessed and are indeterminate. For example, as shown in Fig. E-5, the *batteryStatus* attribute in the *Alarm* class is an indeterminacy source. Its indeterminacy specification is modeled as an OCL constraint: *"self.batteryStatus = BatteryLevel::Low"*, whereas it can be triggered by *setLowBattery()* (the indeterminacy source input). Fig. E-5 also shows that «BeliefElement» is applied to the *Alarm Not Ringing* state and it is linked to the indeterminacy source of *batteryStatus* (via the *Referred Indeterminacy Source* attribute of «BeliefElement») to signify that it is one of the sources that lead to the uncertainty associated with the *Alarm Not Ringing* state.

Note that for the first and third situations (*S1* and *S3* in Table E-3), we recommend specifying an indeterminacy source input either as an Operation without parameters (*Op1*) or as an Operation with parameter(s) constrained with a OCL constraint (*Op2*). Also, for *Smi1* and *Smi3*, an indeterminacy source can be specified as a property (*R1*) or constraint (*R2*). If it is R2, its corresponding indeterminacy specification(s) can then be simply specified as *FALSE* by default and must be switched to *TRUE* to enable the related indeterminacy source.

Table E-3. Recommendations For applying the Indeterminacy Source part of the UUP profile

#	Stereotype Applied		Base Element
S1: States of the environment of the CPS are indeterminate, such as the batteryStatus example shown in Fig. E-5 *and described in Section 4.3.1.*			
R1	«IndeterminacySource» «IndeterminacySpecification»		Property Constraint
	Op1 *Op2*	«IndeterminacySourceInput» «IndeterminacySourceInput»	Operation Operation, Constraint
R2	«IndeterminacySource» «IndeterminacySpecification»		Constraint FALSE (default)

	Op1	«IndeterminacySourceInput»	Operation
	Op2	«IndeterminacySourceInput»	Operation, Constraint
S2: Input data is indeterminate.			
R1	«IndeterminacySource»		Operation
	«IndeterminacySpecification»		Constraint
	«IndeterminacySourceInput»		Constraint
S3: Occurrences of an event from the environment (e.g., "pressing the button") are indeterminate.			
R1	«IndeterminacySource»		Property
	«IndeterminacySpecification»		Constraint
	Op1	«IndeterminacySourceInput»	Operation
	Op2	«IndeterminacySourceInput»	Operation, Constraint
R2	«IndeterminacySource»		Constraint
	«IndeterminacySpecification»		FALSE (default)
	Op1	«IndeterminacySourceInput»	Operation
	Op2	«IndeterminacySourceInput»	Operation, Constraint

In addition, we propose three mechanisms (i.e., *EnablePattern, SelectSpecification* and *FindPosition*), discussed below, to enable an indeterminacy source associated with a specific uncertainty, their corresponding indeterminacy specifications and inputs during test execution.

EnablePattern provides four ways of enabling an indeterminacy source: 1) *Random* – the indeterminacy source is introduced randomly (from the uniform random distribution) during execution; 2) *Always* - the indeterminacy source is always enabled during execution; 3) *Measured* - the indeterminacy source is enabled during execution by a specified measurement, e.g., with a normal distribution; and 4) *Never* - the indeterminacy source is never enabled during the execution. Choosing which option is dependent on how much knowledge information (e.g., experience, historical data) one has about the system.

SelectSpecification provides three ways of selecting which indeterminacy specification(s) of an indeterminacy source to be enabled during test execution: 1) *All* – all associated indeterminacy specifications are enabled; 2) *Random* – enable a random number of randomly selected specification(s) from all the specifications associated with the indeterminacy source during test execution; and 3) *Specified* – the indeterminacy specification(s) specified with the "enabled" attribute is enabled during the test execution. Similarly, which option to take is highly dependent on users' experience, knowledge and available historical data.

FindPosition is about finding a position of a path generated by the UncerTest abstract test case generation strategy, in which an indeterminacy source should be enabled. We define four options for *FindPosition*: 1) *Random* - the position is generated randomly; 2)

Any_Previous – the position can be any previous position before the occurrence of the associated uncertainty; 3) *Just_Previous* – the position is exactly the position right before the occurrence of the associated uncertainty; and 4) *Specified* – the exact position is modeled in the test ready model. Option 1 is recommended when we have no particular preferences or guidance. Option 2 is recommended when one wants to test, if possible, whether the uncertainty is actually due to the indeterminacy source enabled. Option 3 should be used when one wants to know whether the occurrence of the uncertainty is due to its previous step. Option 4 should be used when one has a specific position in mind, based on for example previous experience or historical data.

Note that the three mechanisms can be configured by users to form a concrete strategy (as part of an overall test strategy) for enabling an indeterminacy source associated with an uncertainty and all or part of its associated indeterminacy specifications, at a particular position of a path, which is eventually transformed into executable test cases and executed. In Fig. E-5, we show an example of such configurations for enabling the indeterminacy source of *batteryStatus*, that is, the *SelectSpecification::Specified* indeterminacy specification (i.e., *Low Battery*) should be enabled by following the *EnablePattern::Random* pattern at the *FindPosition::Any_Previous* position during the execution.

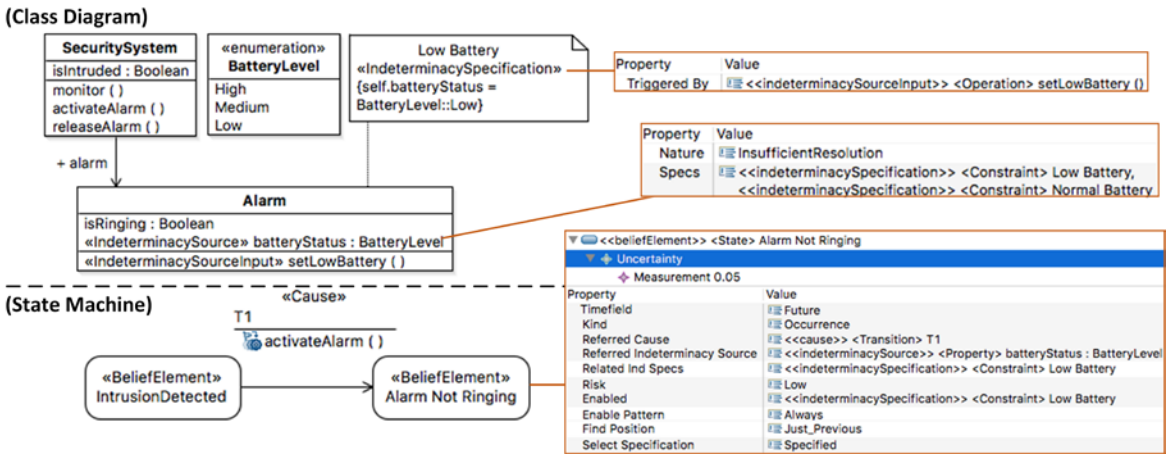


Fig. E-5. An IndeterminacySource Modeling Example for the SafeHome (Partial)

4.3.2 Test Setup and Test Data Generation

When generating executable test cases, test configuration and concrete test data are needed. When applying UncerTum, test configuration is recommended to be specified as a

UML object diagram organized in a package. All the objects and their relationships in this test configuration package will be instantiated before executing test cases.

First, test data generation is needed for triggering call events on transitions. In this case, a guard condition (an OCL constraint) on a transition specifies the valid set of values, with which the call event can be invoked. We used an existing test data generation tool called EsOCL [34], which takes an OCL constraint as an input and generates a set of values that satisfy the constraint. These values are then used as test data in executable test cases.

Second, test data might be needed to trigger occurrences of indeterminacy sources. For any indeterminacy source input that is specified as a stereotyped Constraint or as a stereotyped Operation with its parameters constrained with a constraint, we rely on the EsOCL tool [34] to solve the constraint to generate test data. For any indeterminacy source input specified as an operation with no any parameter, no data needs to be generated to trigger the operation and hence the indeterminacy input.

4.4 Test Execution and Reporting

In addition to test verdicts for test cases, to evaluate occurrences of uncertainties during test execution, we define uncertainty-wise test verdicts as shown in Fig. E-6 (the conceptual model) and Table E-4 (definitions).

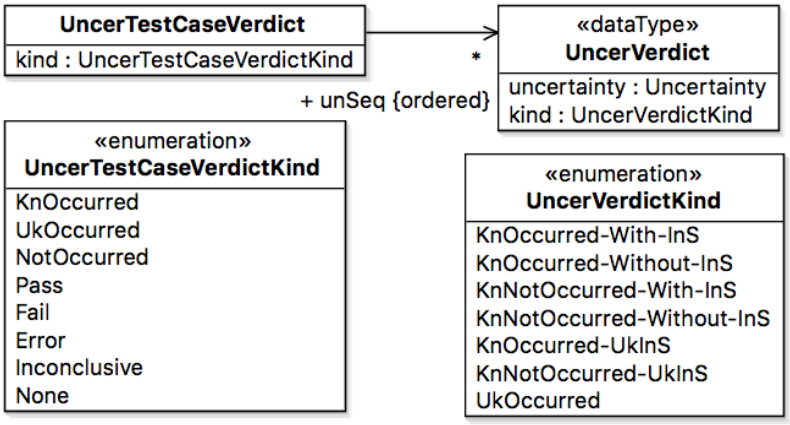


Fig. E-6. Uncertainty-wise Test Verdicts – Conceptual Model

As shown in Fig. E-6 an *UncerTestCaseVerdict* is modeled as a sequence of *UncerVerdict*s for specifying a set of possible evaluations of a test case, including the uncertainty aspect (e.g., known uncertainty occurred (i.e., *KnOccurred*)) and classical test

case verdicts (e.g., *Pass*). An *UncerVerdict* specifies a set of possible evaluations of a test oracle in terms of a specific uncertainty. The seven kinds of uncertainty verdicts are listed as the seven literals of enumeration *UncerVerdictKind*. Their definitions are provided in Table E-4.

Table E-4. Uncertainty-wise Test Verdicts – Definitions of the Literals of the Enumerations (Fig. E-6)

Literal	**Definition**
UncerVerdictKind: It presents the kinds of verdicts for an uncertainty.	
KnOccurred-With-InS	Known uncertainty occurred under the occurrence of a specified indeterminacy source.
KnOccurred-Without-InS	Known uncertainty occurred under the non-occurrence of any specified indeterminacy source.
KnNotOccurred-With-InS	Known uncertainty did not occur under the occurrence of any specified indeterminacy source, but at least one of alternative uncertainty occurred.
KnNotOccurred-Without-InS	Known uncertainty did not occur under the non-occurrence of any specified indeterminacy source, but at least another uncertainty within the same uncertainty space occurred.
KnOccurred-UkInS	Known uncertainty occurred, but its related indeterminacy source is unknown.
KnNotOccurred-UkInS	Known uncertainty did not occur, and its related indeterminacy source is unknown.
UkOccurred	Known uncertainty did not occur, and none of the other uncertainties in the same uncertainty space occurred.
UncerTestCaseVerdictKind: It presents the kinds of the verdicts for a test case.	
KnOccurred	At least one known uncertainty (with any of the three KnOccurred types of UncerVerdictKind) occurred but no UkOccurred.
UkOccurred	At least one UkOccurred.
NotOccurred	All uncertainties are evaluated to be any of the three KnNotOccurred kinds of UncerVerdictKind.
Pass	The execution result of the test case, for which no uncertainty is specified, adheres to the expectations.
Fail	The execution result of the test case, for which no uncertainty is specified, differs from the expectations.
Error	An error is detected.
Inconclusive	The test case execution result cannot be classified as Pass, Fail, Error, KnOccurred, UkOccurred or NotOccurred.
None	A test case has not been executed yet.

5 Evaluation

Section 5.1 introduces case studies. Section 5.2 presents research questions. Section 5.3 presents the design of our evaluation. Results are presented in Section 5.4, the overall discussion is presented in Section 5.5, and threats to validity are presented in Section 5.6.

5.1 Case Study

To assess the cost-effectiveness of UncerTest, we selected two industrial CPS case studies.

The first case study is GeoSports, and the system monitors the performance (e.g., speed and position) and health conditions of players both individually and as a team during a game with the ultimate objective of improving their performance. The GeoSports application that we tested is deployed for Bandy (a type of ice hockey commonly played in northern Europe) and uses the Quuppa system [35]. The testing infrastructure for Bandy is shown in Fig. E-7. Instead of using real players to execute test cases, our industrial partner, Nordic Med Test [36] has deployed a set of test rigs for replacing players. Each test rig has one Quuppa device attached to it. The device communicates its position with one or more locators (antennas) via Bluetooth connections and the locators receive those positions and send them to the Quuppa Server (QPE). The access to the devices, locators, and the QPE server are available as REST APIs. Also, a set of test APIs was implemented by the partner as REST APIs for controlling the test rigs. Notice that we only tested the positioning system in this paper, i.e., collecting the positions from Quuppa tags and transmitting them to the QPE server via locators.

The second case study is Automated Warehouse (AW) provided by ULMA Handling Systems [15], 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) 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. Instead of using real devices to test these scenarios, ULMA [15] and IK4-Ikerlan [37] developed and provided relevant simulators and emulators (Fig. E-7). For example, two handling systems are deployed at two different sites (Site 1 and Site 2). For each site, the local superior monitors software and all types of devices and services and

upload the data to the cloud superior through the network. Each physical device is developed as a simulator where the software, i.e., WMS and MFC, are deployed on. Also, a set of emulators are developed for manipulating the real physical environment, e.g., putting a pallet on the conveyor. To access the devices, software, and environment, the test APIs were implemented by the partner for controlling the physical device, sending requests to the software, and manipulating the physical environment. Further details on the case studies can be consulted in [38].

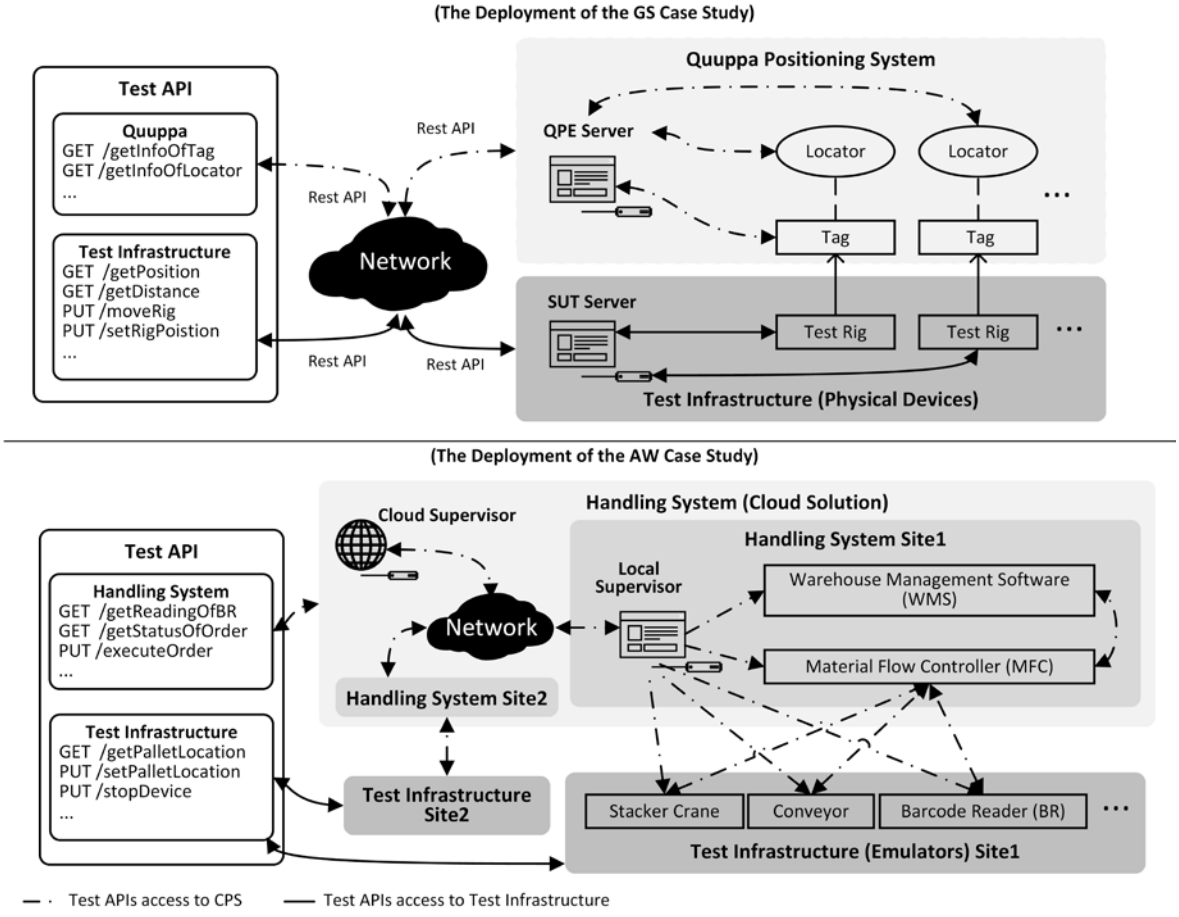


Fig. E-7. The Test Execution Solution of the GS and AW Case Studies

The descriptive statistics of the test ready models of GS and AW are given in Table E-5. We selected one use case for GS and four use cases for AW. For each use case, we selected the number of elements stereotyped as «BeliefElement» (#Belief), uncertainties (#U), known indeterminacy sources (#IndS), known indeterminacy source specifications (#IndSpec), states (#State), and transitions (#Transition). For AW, the percentage of uncertainties specified in the test ready model is more than 50%, which reflects that more than 50%

behavior specified in the test ready model is uncertain. This value is higher than the one for GS since the behavior and environment of AW is relatively complex, e.g., large number of devices.

Table E-5. Descriptive statistics of the case studies

Case	UC	#Belief	#U	#IndS	#IndSp	#State	#Transition	#U/#State	#U/#Transition
AW	AW1	7	11	2	4	12	15	91.7%	73.3%
	AW2	5	9	2	4	12	18	75.0%	50.0%
	AW3	6	10	-	-	10	14	100.0%	71.4%
	AW4	7	8	1	2	16	16	50.0%	50.0%
GS	GS1	6	6	1	2	17	21	35.3%	28.6%

- means unknown indeterminacy source

5.2 Research Questions

We aim to assess which combination of the two test case generation strategies and the four test case minimization strategies is *cost-effective*. In total, we have five combined test strategies. The results for the two test case generation strategies are reported in Table E-6. First, test cases are generated from a BSM using *ASiBP*. With this strategy, the numbers of generated test cases for the two case studies are small, which thus doesn't require test case minimization. The rest of the four strategies are based on test cases generated from a BSM using *ASlBP*, followed by test case minimization (Section 4.2.3) based on the uncertainty related strategies (Section 4.1.2): average normalized number of uncertainties covered (*Problem 1*), percentage of uncertainty space covered (*Problem 2*), average overall uncertainty measure (*Problem 3*), and percentage of unique uncertainties covered (*Problem 4)*. For simplicity, we refer to these strategies as *GMS1* (*ASiBP*), *Str2* (*ASlBP +Problem 1*), *Str3* (*ASlBP + Problem 2*), *Str4* (*ASlBP + Problem 3*) and *Str5* (*ASlBP + Problem 4*) in the rest of the paper. We selected eight commonly used multi-objective search algorithms from the Evolutionary Algorithm, Hybrid Algorithm, and Swarm Algorithm classifications of algorithms. Moreover, we used random search (RS) for the sanity check to determine if complex multi-objective search algorithms are needed, or simply RS suffices.

Table E-6. Results For the Test Case Generation Strategies

Case	UC	Strategy	#TC (nt)	%Transition	%UU
AW	AW1	ASP	20	91.3%	100%
		AMP	420	100%	100%
	AW2	ASP	8	88.8%	100%
		AMP	776	100%	100%
	AW3	ASP	5	85.7%	80%

		AMP	857	100%	100%
	AW4	ASP	5	93.7%	100%
		AMP	296	100%	100%
GS	GS1	ASP	5	71.4%	83.3%
		AMP	1799	100%	100%

Based on our overall objective, we would like to answer the following research questions.

RQ1: How does the selected multi-objective search algorithms (e.g., NSGA-II) compare to RS in terms of solving uncertainty-wise minimization problems (S2—S5)?

RQ2: Which algorithm is the best among selected ones to solve uncertainty-wise minimization problems (S2—S5) respectively?

RQ3: Which uncertainty-wise strategy (S1-S5) is effective to discover uncertainties in the real CPS?

5.3 Design of the Evaluation

The design of our evaluation is shown in Table E-7. The table presents, for each research question, which task we perform, which strategies are compared, which metrics (*Metrics* column) are used, which statistical methods (*Comparison Method* column) are applied, which algorithms are applied, and which case studies are used. Notice that, to decrease the possibility of obtaining results by chance we ran all the algorithms 100 times for each case study and each strategy [39]. We used the implementation of the eight selected multi-objective search algorithms provided by jMetal [40] and used the following default parameter settings: the *Population Size* of 100, the binary tournament for selecting parents, and the simulated binary criterion for recombination. A crossover rate of 90% was used, and mutation rate was polynomial with the rate of *1.0/n*, where *n* is the number of the bit representation of a solution.

Table E-7. Design of the Evaluation

RQ	Experiment Task	Strategy	Metric	Comparison Method	Algorithm		Case Study
1	Compare each algorithm with RS	Str2-Str5	HV (All), PTM (All), ANU (*Str2*), PUS (*Str3*), AUM (*Str4*), PUU (*Str5*), PTR (All)	Vargha and Delaney statistics ($\widehat{A_{12}}$), Kruskal–Wallis Test, Mann-	Evolutionary Algorithm	NSGA-II [41]	AW, GS
						NSGA-III [42]	
						MOCell [43, 44]	
2	Compare each pair of the multi-objective algorithms					SPEA2 [45]	
					Hybrid Algorithm	CellDE [46]	
						AbYSS [47]	
						GDE3 [48]	
					Swarm	SMPSO [49]	

				Whitney U Test (*p*-value)	Algorithm Random Search (only for RQ1)
3	Compare each pair of the strategies	Str1-Str5	*UUDP, NUO, Uk, UkDDP, EOT*	Simple Comparison	The best algorithm

For RQ1 and RQ2, we compared each pair of the algorithms using HyperVolume (HV) [50] and the individual objectives that are relevant for each strategy. For example, O2 is only valid for Str2. HV was selected based on the guidelines for choosing a quality indicator for search-based software engineering problems that require multi-objective optimization [51]. Based on the guidelines for reporting results for search-based software engineering problems [52], we chose Vargha and Delaney statistics ($\widehat{A_{12}}$) and the Mann Whitney U Test (*p*-value) to compare the eight selected multi-objective search algorithms with RS for Str2—Str5.

Results of test case generation for each case study with each test strategy are represented in Table E-6. For Str1, the numbers of test cases generated with *ASiBP* for all the case studies were small and didn't require minimization. For Str2 – Str5, we ran each problem 100 times, and thus we combined all the solutions from all the runs for comparison to answer RQ1 and RQ2. To compare the performance of the algorithms, we designed a mechanism to rank all the algorithms based on the $\widehat{A_{12}}$ values and *p*-values for each metric as shown in the rank algorithm (Fig. E-8). Furthermore, we calculate the confidence for nine algorithms as shown in Table E-8.

```
Rank Algorithm
input       algos[], len(algos)>=2
output      algos[], rank[]//rank[i] is the rank value of algos[i]
1   n ←len(algos)
2   for i ← 1 to n
3      for j ← i+1 to n //sort algos[]
4         if better[1](algos[i], algos[j])
5            switch(algos,i,j)
6   rank[1]=1;
7   for i ← 2 to n //set rank values for algos[]
8      if better[1](algos[i-1], algos[i])
9         rank[i]=rank[i-1]+1;
10     else
11        rank[i]=rank[i-1];
```

[1] Function *better(algo1, algo2)* compares *algo1* with *algo2*. It returns the best algorithm based on the following two conditions: 1) for HV, p-value<0.05 and A12>0.5; 2) p-value<0.05 and A12<0.5

Fig. E-8. Algorithm for Ranking

For RQ3, we picked the best algorithms (BA) for Str2—Str5 based on the results of RQ1 and RQ2, which were used to minimize test cases. The generated test cases for S1 and minimized test cases for Str2 – Str5 were executed on the current deployments of the GS

and AW case studies as shown in Fig. E-7. The execution results for Str1 – Str5 were evaluated based on various cost, effectiveness, and efficiency measures as shown in Table E-8.

Table E-8. Definitions of Metrics for Each Research Question

RQ	Metric		Definitions
RQ1 RQ2	$A = \{NSAG-II, NSGA-III, MOCell, SPEA2, CellDE, AbYSS, GDE3, SMPSO, RS\}$, $Str = \{Str2, Str3, Str4, Str5\}$, *Str2*={*PTM, ANU, PTR, HV*}, *Str3*={*PTM, PUS, PTR, HV*}, *Str4*={*PTM, AUM, PTR, HV*}, *Str5*={*PTM, PUU, PTR, HV*}. Note that 1) A_k represents the *k*th *Algorithm*, e.g. A_1 = *NSGA-II*; 2) $Str_{i,j}$ represents the *j*th objective of the *i*th strategy, e.g., Str_1 = *Str2*, $Str_{1,1}$ = *PTM.*		
	Rank of Algorithm for the objectives of the strategies		$Rank_{A_k}^{Str_{i,j}}$ is the rank value of the A_k algorithm, for the *j*th objective of Str_i strategy, which is calculated as *rank[k]* in Fig. E-8. .
	Confidence of Algorithm for the objectives of the strategies		Confidence of each objective of each strategy is to calculate the percentage of being better than the other algorithms, which is calculated as $Confidence_{A_k}^{S_{ij}} = (Rank_{A_k}^{S_{ij}} / \sum_{n=1}^{9} Rank_{A_n}^{S_{ij}}) \times 100\%$.
	Confidence of Algorithm for the strategies		Confidence of each strategy is to calculate the average confidence of each objective, which is calculated as $Confidence_{A_k}^{S_i} = (\sum_{n=1}^{4} Confidence_{A_k}^{S_{in}}/4) \times 100\%$.
RQ3	Effectiveness	*UUDP*	Unique uncertainty detection percentage is calculated as *UUDP* = *NUUO/NUU,* where *NUUO is* the number of unique uncertainties occurred during the test set execution.
		NUO NUO_{Ind} NUO_{ukInd}	The number of uncertainties occurred during the test set execution, which includes the occurrence of the uncertainties with the occurrence of their specified indeterminacy sources (NUO_{Ind}) or unknown indeterminacy sources (NUO_{ukInd}).
		Error	The number of errors found during the test execution.
		Uk	The number of unknown uncertainties occurred during the test set execution.
		UkDP	Unknown uncertainty detection percentage is calculated as *UkDP* = *Uk/NUU.*
	Cost	*ET*	The execution time of the test set.
	Efficiency	*NT*	The number of executed test cases.
		EoT EoT_{NUO} EoT_{Uk}	The efficiency in terms of time includes 1) EoT_{NUO} is the efficiency of uncertainty detection calculated as EoT_{NUO} = *NUO/ET*; 2) EoT_{Uk} is the efficiency of unknown uncertainty detection calculated as EoT_{Uk} = *Uk/ET.*

5.4 Results and Analyses

In this section, we present results and analyses for the three research questions.

5.4.1 Results for RQ1

Recall that RQ1 focuses on comparing the eight selected multi-objective search algorithms with RS based on the individual objectives, HV for (Str2—Str5) minimization

problems. Due to the large number of comparisons, the detailed results in terms of rank values, p-values and $\widehat{A_{12}}$ values are provided in the technical report corresponding to this paper [53] and submitted supplementary material. The summarized results in terms of confidence and risk (based on the rank of each algorithm) are presented in Table E-9 for each case study. For Str2—Str5, for each use case, we can see that RS has the lowest confidence to be the best algorithm (the *Conf.* column). These results suggest that our problems couldn't have been solved effectively with RS and thus the use of complex multi-objective search algorithms is justified.

5.4.2 Results for RQ2

For RQ2, the detailed results of the comparison of each pair of algorithms (C_2^9, i.e., 36 pair-wise comparisons) for each case study for Str2—Str5, in terms of rank values, p-values and $\widehat{A_{12}}$ values are provided in the technical report corresponding to this paper [53] and submitted supplementary material. The summarized results in terms of confidence of each algorithm, for each use case are presented in Table E-9. As shown in Table E-9, in terms of confidence for Str2—Str5, SPEA2 is consistently the best, or the second best (only for three instances). Based on the results, we recommend using SPEA2 with Str2—Str5 to find the most optimal minimized test cases.

Table E-9. Confidence For Each Algorithm For Each Strategy and Each Case Study

Str.	AW1	AW2	AW3	AW4	GS1	Algorithm	AW1	AW2	AW3	AW4	GS1	Str.
Str2	13%	12%	13%	9%	12%	NSGA-II	13%	15%	14%	11%	14%	Str4
	14%	14%	12%	12%	**15%**	NSGA-III	13%	13%	13%	13%	13%	
	8%	8%	8%	9%	9%	MoCell	9%	7%	7%	8%	8%	
	15%	**17%**	16%	**15%**	**15%**	**SPEA2**	**16%**	**17%**	17%	**16%**	**16%**	
	9%	13%	12%	10%	14%	AbYSS	10%	10%	12%	10%	13%	
	8%	5%	7%	8%	7%	CellDE	6%	5%	5%	7%	5%	
	14%	10%	10%	**15%**	10%	GDE3	13%	10%	10%	15%	10%	
	14%	15%	**17%**	14%	12%	SMPSO	15%	16%	**18%**	14%	15%	
	5%	5%	5%	7%	6%	RS	6%	5%	5%	7%	5%	
Str3	13%	13%	13%	12%	11%	NSGA-II	**13%**	13%	13%	11%	12%	Str5
	13%	13%	13%	12%	13%	NSGA-III	**13%**	13%	13%	11%	12%	
	8%	9%	9%	9%	9%	MoCell	8%	9%	9%	9%	9%	
	14%	**15%**	**15%**	**13%**	**15%**	**SPEA2**	**13%**	**15%**	**15%**	**13%**	**15%**	
	10%	12%	12%	11%	14%	AbYSS	10%	12%	12%	12%	13%	
	8%	7%	7%	10%	7%	CellDE	8%	7%	7%	10%	7%	
	12%	10%	10%	**13%**	10%	GDE3	12%	10%	10%	13%	10%	
	14%	13%	13%	12%	14%	SMPSO	**13%**	13%	13%	12%	14%	
	8%	7%	7%	9%	7%	RS	8%	7%	7%	9%	7%	

5.4.3 Results for RQ3

To answer RQ3, we chose SPEA2 to minimize test cases for Str2 – Str5 for the two case studies and executed the minimized test cases. The test execution results (together with the execution results for Str1) are provided in Table E-10. We compare Str1 – Str5 based on the cost, effectiveness, and efficiency measures (Table E-8). In terms of execution time (i.e., a cost measure, presented in column ET (s), Table E-10), we can observe that Str2 took the highest time to execute for all the use cases except for AW1, where Str4 took the highest time to execute test cases.

Table E-10. Results For RQ3

UC	Str.	NT	PTR	ET (s)	UUDP	NUO	NUO_{Ind}	NUO_{ukInd}	Uk	Err.	UkDP	EoT_{NUO} /min	EoT_{uk} /min
AW1	Str1	20	91.3%	216	45%	25	16	9	10	0	91%	0.116	2.78
	Str2	22	100%	291	45%	**36**	**23**	13	13	**1**	118%	0.124	2.68
	Str3	17	100%	244	45%	30	18	12	11	0	100%	0.123	2.70
	Str4	20	96%	**519**	36%	29	15	**14**	**16**	**1**	145%	0.056	1.85
	Str5	14	100%	170	45%	22	13	9	11	0	100%	**0.130**	3.88
AW2	Str1	8	88.8%	387	67%	11	8	3	0	0	0%	0.028	0
	Str2	106	100%	**2134**	78%	**314**	**205**	**109**	**0**	**4**	0%	0.147	0
	Str3	20	100%	866	78%	52	35	17	0	2	0%	0.060	0
	Str4	54	100%	1114	78%	148	97	51	0	3	0%	0.133	0
	Str5	30	100%	501	78%	91	58	33	0	2	0%	**0.182**	0
AW3	Str1	5	85.7%	3156	60%	8	-	-	0	0	0%	0.003	0
	Str2	138	100%	**99414**	100%	955	-	-	0	**1**	0%	**0.010**	0
	Str3	45	100%	29147	100%	271	-	-	0	0	0%	0.009	0
	Str4	92	100%	54990	100%	568	-	-	0	**1**	0%	**0.010**	0
	Str5	47	100%	30663	100%	305	-	-	0	0	0%	**0.010**	0
AW4	Str1	4	93.7%	8	75%	9	5	4	0	0	0%	1.089	0
	Str2	24	100%	**155**	75%	**296**	**163**	**133**	0	0	0%	1.909	0
	Str3	2	81%	11	63%	23	11	12	0	0	0%	**2.116**	0
	Str4	7	94%	38	75%	79	38	41	0	0	0%	2.105	0
	Str5	4	94%	20	75%	38	20	18	0	0	0%	1.913	0
GS1	Str1	5	71.4%	88	33%	2	1	1	0	0	0%	0.023	0
	Str2	393	95%	**29300**	83%	**1767**	**569**	**1198**	0	0	0%	0.060	0
	Str3	177	100%	12107	83%	717	211	506	0	0	0%	0.059	0
	Str4	203	100%	12717	67%	835	259	576	0	0	0%	**0.066**	0
	Str5	174	100%	11428	83%	715	243	472	0	0	0%	0.063	0

In Table E-10, the *nt* column shows the number of test cases for each test strategy (Str1 – Str5). Recall from Table E-8 that the *UUDP* column shows the percentage of times that the introduced indeterminacy sources led to observing corresponding uncertainties during test execution, whereas the *NUO* column represents the number of uncertainties that were observed as the result of test execution. As shown in Table E-10, consistently for all the five

use cases, test cases generated and minimized with Str2 always led to observe more uncertainties when comparing with the others (the *NUO* column). The NUO_{Ind} (Table E-10) column shows the number of uncertainties out of *NUO* that occurred because of known indeterminacy sources, whereas the NUO_{ukInd} column (definition in Table E-10) shows the number of uncertainties observed due to unknown indeterminacy sources. Once again Str2 is the best across the case studies in terms of NUO_{Ind}. In terms of NUO_{ukInd} (except for AW1 where Str4 is the best), Str2 is the best across the case studies. Even for AW1, Str4 observed only one more uncertainty than Str2.

The *Uk* (defined in Table E-8) column represents the number of unknown uncertainties observed due to unknown indeterminacy sources. For AW1, with Str4, 16 uncertainties in this category were observed, whereas the second highest was 13 with Str2. The *Error* column represents the number of error detected with each test strategy. For AW1 and AW2, both Str2 and Str4 observed one error each, whereas, for AW3, Str2 observed four errors, i.e., higher than the other strategies.

Therefore, we recommend Str2 as it performed better than the others in terms of the studied effectiveness measures except for *Uk* and NUO_{ukInd} for AW1, where Str4 was the second best.

We also compare the strategies based on the efficiency measures. The results are given in the last two columns of Table E-10. Note that the efficiency measures simply tell that how many uncertainties (measured with *Uk* and *NUO*) were observed per minute. For AW1, AW2, and AW3, for the EoT_{NUO}/min measure, Str5 is the best. For AW4, Str3 is the best with an efficiency value of 2.116 for EoT_{NUO}/min, whereas, for GS, Str4 is the best with an efficiency value of 0.066 *for* EoT_{NUO}/min. However, the differences of these two with the efficiency values of Str5 are not much. For example, for GS, Str5 has as efficiency value of 0.063, i.e., the difference of 0.003 with Str4. This means that Str5 is likely to observe 0.003 fewer uncertainties than Str4 per minutes. Such difference is negligible in practice. In terms of EoT_{Uk}/min for AW1, once again Str5 is the best strategy. Based on the above results, we suggest using Str2 when the test execution time is not a concern; otherwise, we recommend using Str5 since it is highly likely to be efficient.

5.5 Discussion

Based on the results and analysis of RQ1, we can conclude that our uncertainty-wise test minimization approaches are complex and thus RS was not sufficient to solve our problems. RS has the lowest confidence to be the best algorithm (i.e., 5.28% on average) as compared to the rest of the algorithms when studying the results of all the use cases together. When comparing the selected multi-objective search algorithms for the four uncertainty-wise test minimization problems (RQ2), we found that SPEA2 has the highest confidence to be the best algorithm (i.e., 12.12% on average) as compared to the rest of the algorithms including RS.

When comparing the five test strategies, we found Str2 (i.e., *ASlBP* with minimization focused on covering the number of uncertainties) with SPEA2 turned out to be the best. Str2 with SPEA2 observed on average 51%[24] more uncertainties than the rest of the strategies due to unknown indeterminacy sources when combining the results from all the use cases. Moreover, it managed to observe 13 unknown uncertainties due to unknown indeterminacy sources across all the use cases. In comparison, Str4 with SPEA2 managed to observe 16 unknown uncertainties due to unknown indeterminacy sources, i.e., three more than the Str2 and SPEA2 combination.

In terms of practical implications, we have four key findings. First, the results of observed known uncertainties due to known indeterminacy sources (the NUO_{Ind} column) simply confirm our belief about known uncertainties of a CPS. If the belief is not confirmed, it means that the belief of the test modeler is far from truth. Then we recommend a test modeler to update her/his belief on the test ready model based on the results of test execution. Second, the results of observed known uncertainties due to unknown indeterminacy sources (the NUO_{ukInd} column) tell us that the known uncertainties can happen due to the indeterminacy sources that we were not aware of. As a result, such unknown indeterminacy sources need to be investigated and discovered with the help of our industrial partners. Once discovered, the test ready models must be updated to reflect these indeterminacy sources. Third, the

[24] The value is calculated as $\frac{\sum_{i=1}^{4}\sum_{j=1,3,4,5}(NUO_{ukind}^{UC_iStr_2}-NUO_{ukind}^{UC_iStr_j})/(NUO_{ukind}^{UC_iStr_2}+NU)_{ukind}^{UC_iStr_j})}{4\times4}$, where *UC={AW1, AW2, AW4, GS1}, Str = {Str1, Str2, Str3, Str4, Str5}*. $NOU_{ukind}^{UC_1Str_1}$ is the number of uncertainties observed with Str1 for the AW1 use case.

discovery of unknown uncertainties due to unknown indeterminacy sources (the *Uk* column) need to be investigated once again together with our industrial partners and reflected in the test ready models as known uncertainties due to known indeterminacy sources (if investigated and found) for future testing. Fourth, the *Error* column tells the errors found during the test execution and must be fixed in the implementation of the CPSs. Note that we observed 15 times of occurrences of errors for the AW case study. Due to confidentiality issues, further details on the errors and uncertainties cannot be provided. Nonetheless, the results tell us that our proposed test strategies can help us confirming our belief about known uncertainties, discovering unknown uncertainties and unknown indeterminacy sources, and find errors.

5.6 Threats to Validity

External validity. A typical *external validity threat* with any empirical study is related to the generalization of results. Our experiment results are valid for two case studies (five use cases) from two CPS domains (Automation, Healthcare) and thus additional experiments with different case studies are required to further generalize the results..

Internal validity. . There are four main internal validity threats in our experiment. First, in terms of test case generation with *ASlBP*, we used the same criteria to generate test cases for all the use cases. This includes generating test cases that must achieve the 100% transition coverage and 100% unique uncertainty coverage. Second, as suggested in [52], all the SBSE problems face a common *internal validity* threat that is related to parameter settings used for the search algorithms. We used the default parameter settings for all the algorithms based on the existing guidelines [52, 54]. Third, we used the same criteria to introduce indeterminacy sources during the test execution for each use case. This means that we used the same values for *EnablePattern, FindPosition*, and *SelectSpecification* (Fig. E-5) when executing test cases generated from each test strategy across the use cases. Fourth, the fact that executing each test case more than once can lead to different execution results. Therefore, we executed a test case exactly once if it was included in the test case sets generated by multiple test strategies.

Conclusion validity. There are two main conclusion validity threats in our experiment. First, as discussed in [55], due to randomness in search algorithms, results may have been

produced by chance. We handled this threat as suggested in [55], that is to repeat the experiments 100 times. Based on the standard guidelines [52] to report search-based software engineering experiments, we chose the Kruskal–Wallis test to calculate p-value for multiple comparisons with 5% significance level, the Mann-Whitney U test to calculate p-value for pair comparison with 5% significance level, to determine practical and statistical significances of results. Second, our experiment results are based on one-time test execution due to limited resources available to execute test cases on the physical test infrastructures. Additional experiments are required in the future to execute test cases more than once to study whether executing one test case multiple times lead to observing different uncertainties.

Construct validity. As suggested in [39, 56], the same stopping criterion must be used for all the evaluated algorithms to avoid any potential bias in results. Following the guidelines, we used the same number of fitness evaluations (25000) and thus dealt with this type of validity threat.

6 Automation

The (open source) tool support[25] for UncerTest is shown in Fig. E-9, a user creates a BM, i.e., belief model (including BCDs and BSMs) in the IBM Rational Software Architect (RSA) using UncerTum implemented in the IBM RSA [9]. In addition to BCDs and BSMs, the BM also includes one or more object diagrams (corresponding to BCDs) that represent the test configuration of the CPS being tested.

The first toolset of UncerTest is referred to as *Abstract Test Case Generator*. *AG1* takes BSMs as input and convert them into graphs (SMGraph) in JGraph [57] based on a test case generation strategy (Section 4.1.2), which can be selected by a tester. *AG2* takes the graph representation of BSMs as input and converts them into deep paths using the JGrapht tool [57]. Notice that multiple regions are not handled by JGrapht, and thus we extended it for this purpose. *AG3* takes the generated deep paths as input and calculates *UM* for each path using the *Uncertainty Measurement Calculator* and produces abstract test cases and associated *UM* with each test case.

[25] The tool for UncerTest is open source, which is available at https://bitbucket.org/ManZH/uncertest-v1.

The second toolset is *Uncertainty-Wise Test Case Minimization*. Its *Solution Solver* uses jMetal's implementation of the multi-objective search algorithms and RS to minimize the number of abstract test cases based on the four test case minimization strategies (Section 4.2). A tester can select any algorithm and any of the four strategies to perform test case minimization. The output is a minimized set of test cases and values for the relevant objectives (Section 4.2). *Solution Processor* converts the output to an EMF model [58], which is the key input for the third toolset.

The third toolset is *Executable Test Case Generator*. *EG1* takes BCDs as input and converts them to *Java Entities*, which are further extended by a tester as *Entities Adapter* to provide actual implementation of operations, e.g., how to invoke REST APIs in GS. For each case study, a user has to manually implement *Entities Adapters* to bridge the gap between model elements and implementation of Test API. *EG2* takes the object diagram as input and outputs *Test Setup*, which is required for execution of test cases. Finally, *EG3* takes the EMF model file as the input and invokes EsOCL [34] to obtain concrete test data. EsOCL is a search-based OCL solver that takes input an OCL constraint and provides a set of data that satisfies the constraint. Using the output from EsOCL, EG3 produces executable test cases, where each executable test case imports Eclipse OCL [59] to check OCL constraints (state invariants) at runtime, which serve as test oracles.

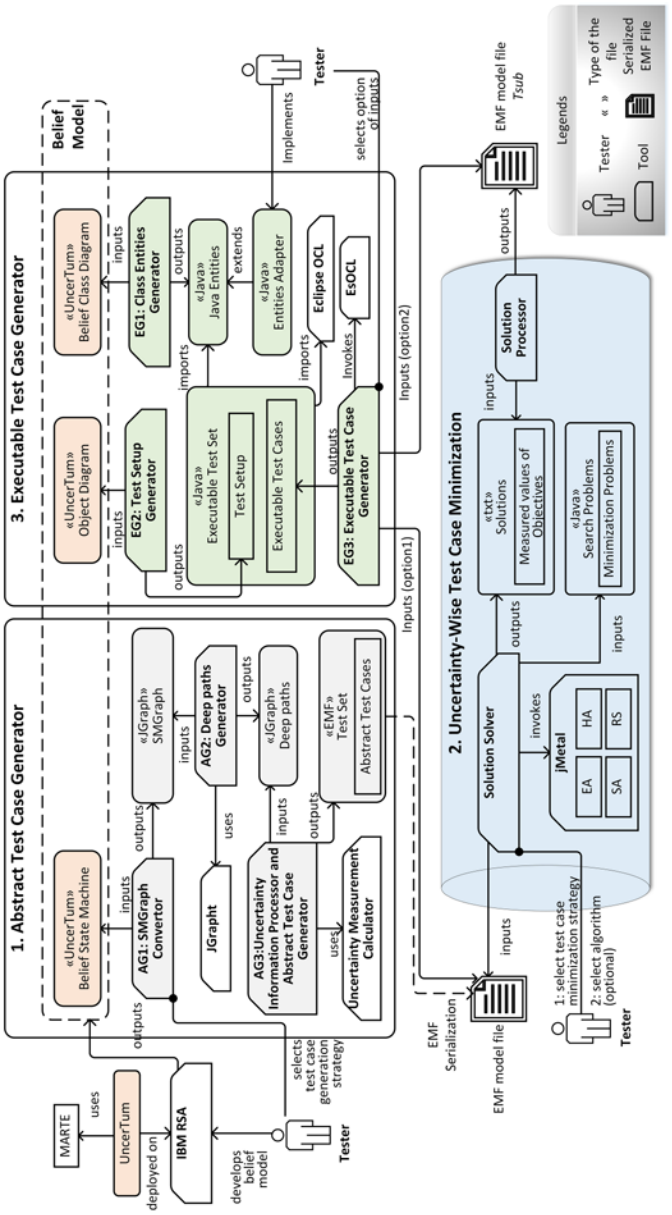


Fig. E-9. Overall Automation Solution of UncerTest

7 Related Work

Walkinshaw and Fraser [5] proposed a black-box testing framework to select test cases for execution to decrease uncertainty about the correctness of a software system. The proposed framework relies on Genetic Programming (GP) [60] to infer models of a system under test. It generates random inputs and assesses them on the inferred models to select ones that create most uncertainty, and eventually only execute the selected ones on the real

system under test. Uncertainty was measured in their context as the level of confidence in the corresponding output of input (i.e., test data). UncerTest shares a similar objective, that is, selecting test cases for execution by taking into account uncertainty. Differences between the two approaches can be summarized from the three aspects: 1) UncerTest focuses on testing CPS under uncertainty, but their proposed framework is for software; 2) UncerTest requires initial BMs with subjective uncertainty specified as the input, whereas in their approach models are inferred by GP, which requires the execution of the software under test; and 3) UncerTest elaborates uncertainty from the four aspects (i.e., number of uncertainties, number of unique uncertainties, uncertainty space, and uncertainty measure from the Uncertainty Theory), whereas their approach is based on an existing uncertainty sampling technique.

Another related work [6] focuses exclusively on time-related uncertainty. It relies on UML sequence diagrams together with the UML Profile for Schedulability, Performance, and Time (SPT) [61]. This work, however, only supports modeling uncertainty in time on messages of sequence diagrams. As discussed in Section 2.1, UncerTest is built on UncerTum [9], which is a comprehensive modeling framework for specifying various types of uncertainty (e.g., time, content and environment). The work presented in [6] focuses on stress testing of systems in the existence of time-related uncertainty on messages, which may complement the UncerTest framework, which can be investigated in the future.

David et al. [62] presented some test generation principles and algorithms (e.g., the online testing tool UPPAAL-TRON [63]) and discussed the feasibility of applying them for testing timed systems under uncertainty, at a high level of abstraction. In their context, uncertainty is mainly caused by the inherent concurrent and indeterminate nature of timed systems. UncerTest, however, addresses uncertainty with a much broader scope and has an end-to-end MBT solution.

In [64], the authors presented a solution to transform UML use case diagrams and state diagrams into usage graphs appended with probability information about the expected use of the software. Such probability information can be obtained in several ways by relying on, e.g., domain expertise or usage profiles of software. 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 terms of testing, the authors proposed to use an existing work [65] to generate test cases. In the context of UncerTest, we focus on test generation based on the uncertainty theory [12].

To model uncertainty (inherent in real-world applications) with UML class diagrams, an extension was proposed in [66-68], 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 [69] 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 [70] to transform fuzzy UML data models into web ontologies to support automated reasoning on fuzzy properties in the context of web services. These works focus on the analyses at the design time, whereas our work focuses on testing. Regarding modeling, our UncerTum focuses on uncertainty in a comprehensive and precise manner by considering various types of measures such as probability, vagueness, and fuzziness. The methodologies proposed in [66-68] for specifying fuzzy UML data can easily integrate with our model libraries when needed and potentially used to support MBT of CPSs under uncertainty. However, this requires further investigation.

In [71], a language-independent solution was proposed 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 regarding expressing the uncertainty of modelers. However, in the context their work, the focus is on uncertainty in partial models for supporting model refinement and evolution. In contrast, we focus on modeling uncertainty (lack of confidence) in test ready models that are in turn used for test case generation and minimization relying on the uncertainty theory.

8 Conclusion

Nowadays, Cyber-Physical Systems (CPSs) are everywhere in our daily life. It is forecasted that applications of CPSs will span over many different domains shortly, including autonomous vehicles, robotics, healthcare, industrial automation, among others.

One critical dimension of the complexity of developing and testing such systems is due to the inherent uncertainty of their operational environment and uncertain behaviors of themselves. To tackle this challenge, in this paper, we proposed a model-based and search-based test case generation and minimization framework (named as UncerTest) for testing CPSs under uncertainty. UncerTest takes advantages of the uncertainty theory and search-based optimization techniques, based on which, it also proposes an innovative set of uncertainty-related test case minimization strategies. We evaluated UncerTest with two industrial CPSs case studies and eight commonly used multi-objective search algorithms. The best test strategy managed to discover on average 51% more uncertainties due to unknown indeterminacy sources as compared to the rest of the test strategies across the case studies. The same test strategy managed to discover 118% more unknown uncertainties as compared to the already known ones.

Acknowledgment

This research was supported by the EU Horizon 2020 funded project U-Test (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. The corresponding author of the paper is Tao Yue. We sincerely thank our industrial partners (ULMA Handlilng Systems and Nordic Medtest), especially Oscar Okariz and Malin Hedman, for their support on providing the case studies and conducting the experiment.

References

[1] D. B. Rawat, J. J. Rodrigues, and I. Stojmenovic, Cyber-physical systems: from theory to practice, CRC Press, 2015.

[2] P. Derler, E. A. Lee, and A. S. Vincentelli, Modeling Cyber-Physical Systems, Proceedings of the IEEE, vol. 100, no. 1 (2012) 13-28, 10.1109/JPROC.2011.2160929.

[3] M. Woehrle, K. Lampka, and L. Thiele, Conformance testing for cyber-physical systems, ACM Transactions on Embedded Computing Systems (TECS) vol. 11, no. 4 (2013) 1-23, 10.1145/2362336.2362351.

[4] H. Abbas, B. Hoxha, G. Fainekos, J. V. Deshmukh, J. Kapinski, and K. Ueda, Conformance testing as falsification for cyber-physical systems, arXiv preprint arXiv:1401.5200 (2014).

[5] N. Walkinshaw, and G. Fraser, Uncertainty-Driven Black-Box Test Data Generation, in: the 10th IEEE International Conference on Software Testing, Verification and Validation (ICST 2017), Tokyo, Japan. pp. 253-263, 2016.

[6] V. Garousi, 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. pp. 92-101, 2008.

[7] G. Bammer, and M. Smithson, Uncertainty and risk: multidisciplinary perspectives, Routledge, 2012.

[8] D. V. Lindley, Understanding uncertainty (revised edition), John Wiley & Sons, 2014.

[9] M. Zhang, S. Ali, T. Yue, and R. Norgre, An Integrated Modeling Framework to Facilitate Model-Based Testing of Cyber-Physical Systems under Uncertainty, Technical report 2016-02, Simula Research Laboratory, 2016; https://www.simula.no/publications/integrated-modeling-framework-facilitate-model-based-testing-cyber-physical-systems.

[10] M. Zhang, S. Ali, T. Yue, R. Norgren, and O. Okariz, Uncertainty-Wise Cyber-Physical System test modeling, Software & Systems Modeling (2017), 2017/07/25, 10.1007/s10270-017-0609-6.

[11] P. Ammann, and J. Offutt, Introduction to software testing, Cambridge University Press, 2016.

[12] B. Liu, Uncertainty theory, Springer, 2015.

[13] J. Brownlee, Clever algorithms: nature-inspired programming recipes, First Edition ed., LuLu, 2012.

[14] "Future Position X," accessed 2017; http://www.fpx.se/.

[15] "ULMA Handling System," accessed 2017; http://www.ulmahandling.com/en/.

[16] M. Zhang, B. Selic, S. Ali, T. Yue, O. Okariz, and R. Norgren, 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, 2016.

[17] OMG, UML Profile For MARTE: Modeling And Analysis Of Real-Time Embeded Systems™, 2011, http://www.omg.org/spec/MARTE/.

[18] B. Liu, Why is there a need for uncertainty theory, Journal of Uncertain Systems, vol. 6, no. 1 (2012) 3-10.

[19] P. C. Jorgensen, Software testing: a craftsman's approach, CRC press, 2016.

[20] Z. Xuemei, T. Xiaolin, and P. Hoang, Considering fault removal efficiency in software reliability assessment, IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans, vol. 33, no. 1 (2003) 114-120, 10.1109/TSMCA.2003.812597.

[21] Y. Zhu, Uncertain optimal control with application to a portfolio selection model, Cybernetics and Systems, vol. 41, no. 7 (2010) 535-547, 2010/09/24, 10.1080/01969722.2010.511552.

[22] L. Yang, K. Li, and Z. Gao, Train Timetable Problem on a Single-Line Railway With Fuzzy Passenger Demand, IEEE Transactions on Fuzzy Systems, vol. 17, no. 3 (2009) 617-629, 10.1109/TFUZZ.2008.924198.

[23] J. Peng, Risk metrics of loss function for uncertain system, Fuzzy Optimization and Decision Making, vol. 12, no. 1 (2013) 53-64, 2013//, 10.1007/s10700-012-9146-5.

[24] S. Han, Z. Peng, and S. Wang, The maximum flow problem of uncertain network, Information Sciences, vol. 265 (2014) 167-175, 5/1/, http://dx.doi.org/10.1016/j.ins.2013.11.029.

[25] W. Rudin, Real and complex analysis, Tata McGraw-Hill Education, 1987.

[26] OMG, Unified Modeling Language (UML), June 2015, http://www.omg.org/spec/UML/.

[27] OMG, Unified Modeling Language™ (UML), 2015, http://www.omg.org/spec/UML/.

[28] D. E. Knuth, "The art of computer programming, 3rd edn. seminumerical algorithms, vol. 2," Addison-Wesley, Reading, 1997.

[29] J. Offutt, and A. Abdurazik, Generating tests from UML specifications, in: International Conference on the Unified Modeling Language. pp. 416-429, 1999.

[30] J. Offutt, S. Liu, A. Abdurazik, and P. Ammann, Generating test data from state-based specifications, Software testing, verification and reliability, vol. 13, no. 1 (2003) 25-53.

[31] P. Samuel, R. Mall, and A. K. Bothra, Automatic test case generation using unified modeling language (UML) state diagrams, IET software, vol. 2, no. 2 (2008) 79-93.

[32] L. C. Briand, Y. Labiche, and Y. Wang, Using simulation to empirically investigate test coverage criteria based on statechart, in: Proceedings of 26th International Conference on Software Engineering (ICSE 2004), Edinburgh, UK. pp. 86-95, 2004.

[33] OMG, "Object Constraint Language™ (OCL™)," 2014, http://www.omg.org/spec/OCL/.

[34] S. Ali, M. Z. Iqbal, A. Arcuri, and L. C. Briand, Generating test data from OCL constraints with search techniques, IEEE Transactions on Software Engineering, vol. 39, no. 10 (2013) 1376-1402.

[35] "Quuppa - Do more with Location," accessed 2017; http://quuppa.com/.

[36] "Nordic Med Test," accessed 2017; http://www.nordicmedtest.se/.

[37] "IK4-IKERLAN," accessed 2017; http://www.ikerlan.es/eu/.

[38] "Use Cases - Industrial Case Studies," accessed 2017; http://www.u-test.eu/use-cases/.

[39] S. Ali, L. C. Briand, H. Hemmati, and R. K. Panesar-Walawege, A systematic review of the application and empirical investigation of search-based test case generation, IEEE Transactions on Software Engineering, vol. 36, no. 6 (2010) 742-762.

[40] "jMetal," accessed 2016; http://jmetal.sourceforge.net/.

[41] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, A fast and elitist multiobjective genetic algorithm: NSGA-II, IEEE transactions on evolutionary computation, vol. 6, no. 2 (2002) 182-197.

[42] K. Deb, and H. Jain, An Evolutionary Many-Objective Optimization Algorithm Using Reference-Point-Based Nondominated Sorting Approach, Part I: Solving Problems With Box Constraints, IEEE Transactions on Evolutionary Computation, vol. 18, no. 4 (2014) 577-601, 10.1109/TEVC.2013.2281535.

[43] A. J. Nebro, J. J. Durillo, F. Luna, B. Dorronsoro, and E. Alba, Mocell: A cellular genetic algorithm for multiobjective optimization, International Journal of Intelligent Systems, vol. 24, no. 7 (2009) 726-746.

[44] A. J. Nebro, J. J. Durillo, F. Luna, B. Dorronsoro, and E. Alba, Design issues in a multiobjective cellular genetic algorithm, in: S. Obayashi, K. Deb, C. Poloni, T. Hiroyasu and T. Murata, eds. International Conference on Evolutionary Multi-Criterion Optimization. pp. 126-140, 2007.

[45] E. Zitzler, M. Laumanns, and L. Thiele, SPEA2: Improving the strength Pareto evolutionary algorithm, in: Evolutionary Methods for Design, Optimization and Control with Applications to Industrial Problems (EUROGEN 2001), Athens. Greece, *International Center for Numerical Methods in Engineering*, 2001.

[46] J. J. Durillo, A. J. Nebro, F. Luna, and E. Alba, Solving Three-Objective Optimization Problems Using a New Hybrid Cellular Genetic Algorithm, in: R. Günter, J. Thomas, L. Simon, P. Carlo and B. Nicola, eds. the 10th international conference on Parallel Problem Solving from Nature: PPSN X. pp. 661-670, 2008.

[47] A. J. Nebro, F. Luna, E. Alba, B. Dorronsoro, J. J. Durillo, and A. Beham, AbYSS: Adapting scatter search to multiobjective optimization, IEEE Transactions on Evolutionary Computation, vol. 12, no. 4 (2008) 439-457.

[48] S. Kukkonen, and J. Lampinen, GDE3: The third evolution step of generalized differential evolution, in. pp. 443-450,

[49] A. J. Nebro, J. J. Durillo, J. Garcia-Nieto, C. A. C. Coello, F. Luna, and E. Alba, SMPSO: A new pso-based metaheuristic for multi-objective optimization, in: 2009 IEEE Symposium on Computational Intelligence in Multi-Criteria Decision-Making (MCDM), Nashville, TN, USA. pp. 66-73, 2009.

[50] E. Zitzler, and L. Thiele, Multiobjective evolutionary algorithms: a comparative case study and the strength Pareto approach, IEEE transactions on Evolutionary Computation, vol. 3, no. 4 (1999) 257-271.

[51] S. Wang, S. Ali, T. Yue, Y. Li, and M. Liaaen, A practical guide to select quality indicators for assessing pareto-based search algorithms in search-based software engineering, in: Proceedings of the 38th International Conference on Software Engineering (ICSE 2016), New York, NY, USA. pp. 631-642, 2016.

[52] A. Arcuri, and L. Briand, A practical guide for using statistical tests to assess randomized algorithms in software engineering, in: Proceedings of the 33rd International Conference on Software Engineering (ICSE 2011), Waikiki, Honolulu, HI, USA. pp. 1-10, 2011.

[53] M. Zhang, S. Ali, T. Yue, and M. Hedman, Uncertainty-wise Test Case Generation and Minimization for Cyber-Physical Systems: A Multi-Objective Search-based Approach, Technical report 2016-13, Simula Research Laboratory, 2016; https://www.simula.no/publications/uncertainty-based-test-case-generation-and-minimization-cyber-physical-systems-multi.

[54] D. J. Sheskin, Handbook of parametric and nonparametric statistical procedures, CRC Press, 2003.

[55] S. Wang, S. Ali, and A. Gotlieb, Minimizing test suites in software product lines using weight-based genetic algorithms, in: Proceedings of the 15th annual conference on Genetic and evolutionary computation. pp. 1493-1500, 2013.

[56] M. de Oliveira Barros, and A. C. Dias-Neto, Threats to Validity in Search-based Software Engineering Empirical Studies, Technical Report 0006/2011, Universidade Federal Do Estado Do Rio de Janeiro, 2011; http://seer.unirio.br/index.php/monografiasppgi/article/viewFile/1479/1307.

[57] "JGrapht," accessed 2016; http://jgrapht.org/.

[58] "Eclipse Modeling Framework (EMF)," accessed 2016; https://eclipse.org/modeling/emf/.

[59] "Eclipse OCL," accessed 2016; http://www.eclipse.org/modeling/mdt/?project=ocl#ocl.

[60] R. Poli, W. B. Langdon, N. F. McPhee, and J. R. Koza, A field guide to genetic programming, Lulu. com, 2008.

[61] OMG, UML Profile For Schedulability, Performance, and Time™, 2005, http://www.omg.org/spec/SPTP/.

[62] A. David, K. G. Larsen, S. Li, M. Mikucionis, and B. Nielsen, Testing real-time systems under uncertainty, in: International Symposium on Formal Methods for Components and Objects. pp. 352-371, 2010.

[63] A. Hessel, K. G. Larsen, M. Mikucionis, B. Nielsen, P. Pettersson, and A. Skou, Testing real-time systems using UPPAAL, *Formal methods and testing*, pp. 77-117: Springer, 2008.

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

[65] J. M. Selvidge, Statistical Usage Testing: Expanding the Ability of Testing, in Software Testing, Analysis & Review, 1999.

[66] Z. Ma, Fuzzy information modeling with the UML, Idea (2005).

[67] Z. M. Ma, F. Zhang, and L. Yan, Fuzzy information modeling in UML class diagram and relational database models, Applied Soft Computing, vol. 11, no. 6 (2011) 4236-4245.

[68] L. Yan, and Z. M. Ma, Extending nested relational model for fuzzy information modeling, in: 2009 WASE International Conference on Information Engineering. pp. 587-590, 2009.

[69] Z. M. Ma, F. Zhang, L. Yan, and J. Cheng, Representing and reasoning on fuzzy UML models: A description logic approach, Expert Systems with Applications, vol. 38, no. 3 (2011) 2536-2549.

[70] F. Zhang, and Z. M. Ma, Construction of fuzzy ontologies from fuzzy UML models, International Journal of Computational Intelligence Systems, vol. 6, no. 3 (2013) 442-472.

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

[72] M. Famelis, R. Salay, and M. Chechik, Partial models: Towards modeling and reasoning with uncertainty, in: Proceedings of the 34th International Conference on Software Engineering (ICSE 2012), Zurich, Switzerland. pp. 573-583, 2012.

[73] M. Famelis, and S. Santosa, MAV-Vis: a notation for model uncertainty, in: Proceedings of the 5th International Workshop on Modeling in Software Engineering (MiSE 2013), San Francisco, CA, USA. pp. 7-12, 2013.

Appendixes

Appendix A. Definitions of U-Model Concepts

To understand uncertainty, in our previous work [15], we defined the conceptual model, *U-Model*, to specify, classify and identify uncertainty and its associated concepts. U-RUCM presented in this paper is an implementation of *U-Model* to enable the specification of uncertainty in use case models. *U-Model* was published in [15] and we have added definitions of its concepts in this appendix to make the paper self-contained, which are organized into three parts: belief model, uncertainty model and measure model.

A.1 Belief Model

The Belief Model in Fig. Appx-1 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 [15]. In addition, the definitions of the concepts in Belief Model are represented in Table. Appx-1.

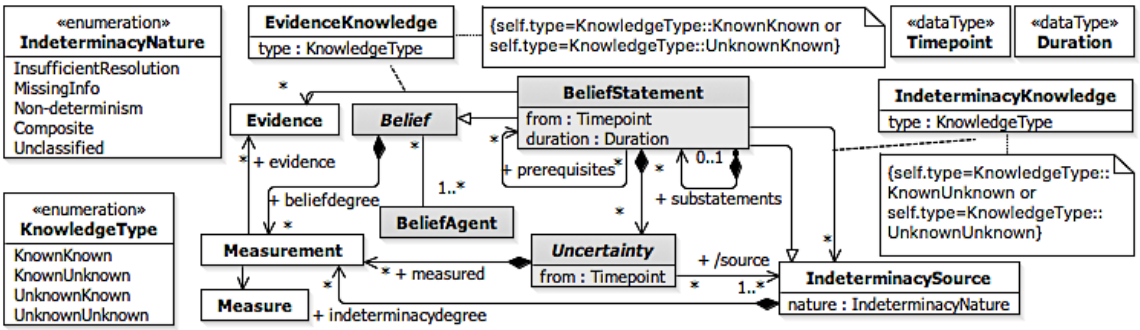


Fig. Appx-1. Core Belief Model of *U-Model*

Table. Appx-1. Definitions of the Concepts in the Belief Model

Concept	Definition
Belief (abstract)	A belief is an *implicit* subjective explanation or description of some phenomena or notions[26] 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**:

[26] The term "phenomena" here is intended to cover aspects of objective reality, whereas "notion" covers abstract concepts, such those encountered in mathematics or philosophy.

Concept	Definition
	• 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[27] 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

[27] We exclude here from this definition "virtual" belief agents, such as those that might occur in virtual reality systems and computer games.

Concept	Definition
	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[28]. **Enumeration literals:** • InsufficientResolution – The information available about the phenomenon in question is not sufficiently precise. • 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

[28] Indeterminacy represents a situation whereby the full knowledge necessary to determine the required factual state of some phenomena or notions is unavailable.

Concept	Definition
	• 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. • 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*.

A.2 Uncertainty Model

The Uncertainty Model in Fig. Appx-2 inspired by the literature of uncertainty expands on Uncertainty from several different viewpoints and introduces related abstractions [15], i.e. risk, pattern, and the definitions of the concepts in Uncertainty Model are represented in Table. Appx-2.

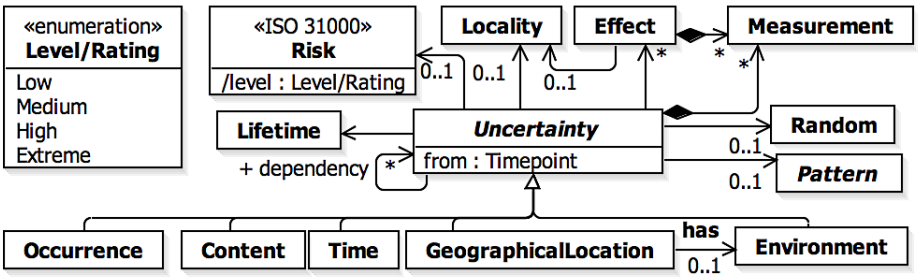


Fig. Appx-2. Core Uncertainty Model of *U-Model*

Table. Appx-2. Definitions of the 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 [40] or any other matrices

Concept	Definition
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*
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*.

The Pattern Model in Fig. Appx-3 shows the conceptual model for the occurrence pattern of Uncertainty [15], and the definitions of the concepts in Pattern Model are represented in Table. Appx-3.

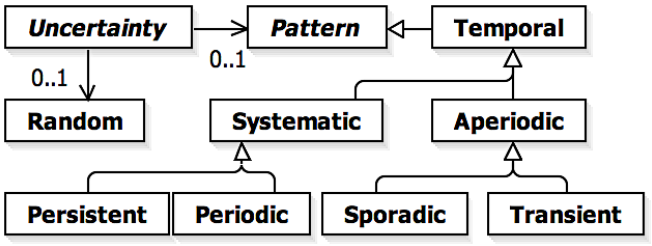


Fig. Appx-3. The Pattern Model

Table. Appx-3. Definitions of the 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 **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.

A.3 Measure Model

The Measure Model in Fig. Appx-4 describes the commonly known measures related to Uncertainty [15], and the definitions of the concepts in Measure Model are represented in Table. Appx-4.

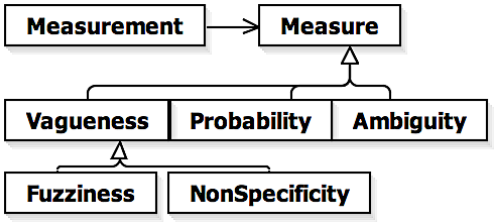


Fig. Appx-4. Core Measure Model of *U-Model*

Table. Appx-4. Definitions of the 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 [41]. **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.

Appendix B. An Example of Questionnaire of the AW Case Study

As discussed in Section 6.2 of Paper B, we conducted a questionnaire-based survey. The summary of the questions is provided in Table. Appx-5 where we also indicate the example questions that are relevant to each type of the question. In Section B.1, we provide a sanitized uncertainty requirement of the AW case study, for which a list of questions was derived (Q1-Q10). In Section B.2, after refining the RUCM specification (Fig. Appx-5), two questions (Q11-Q12) were derived.

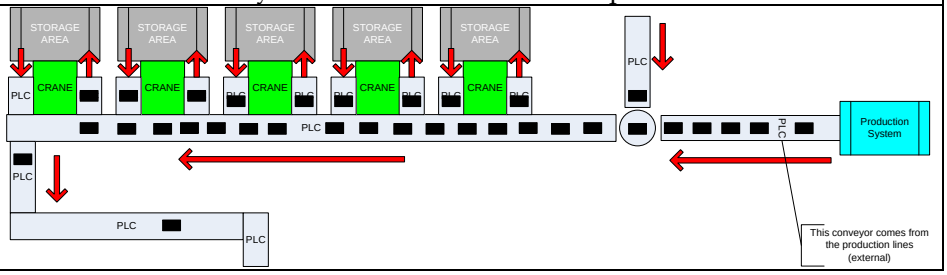
Table. Appx-5. Design of the Questionnaire and Example Questions

#	Explanation	Index of Question
1	Inquiry the boundary of the system to define actors in the use case model.	Q1, Q2, Q3
2	Check the completeness of the flow of events of each use case specification.	Q4
3	Inquiry the existence of sources related to an actor.	Q5, Q6, Q7
4	Inquiry the existence of potential uncertainties related to system properties or behaviors.	Q7
5	Inquiry existence of the potential uncertainties regarding time, nature and human being.	Q8, Q9, Q10
6	Inquiry if a potential uncertainty is valid by checking if it is derived based on system properties or behaviors.	Q11-Q12.1)
7	Inquiry the completeness of the types of uncertainties defined in U-Model.	Q11-Q12.2)
8	Inquiry the type of a specified uncertainty.	Q11-Q12.2)
9	Inquiry the measure and measurement of a specified uncertainty.	Q11-Q12.3).a)
10	Inquiry the risk of a specified uncertainty.	Q11-Q12.3).b)
11	Inquiry the evidence to support the specified measurement and risk of a specified uncertainty.	Q11-Q12.3).a-b).i

B.1 Examples of Uncertainty Requirements Specified in RUCM and Corresponding Questions

In Table. Appx-6, we provide an example of uncertainty requirements developed by our industrial partner. Note that the RUCM editor was not used because the partner wanted to add additional fields (e.g., *Means for validation/verification*, *Models/Metrics*, *Change History*), which were not part of the RUCM template. Therefore, Word was used as the tool to specify all the use case specifications.

Table. Appx-6. An Example of uncertainty requirements specified in RUCM

Scale Up for a Larger Number of Orders to Handle	
Use Case ID	UC2_INTE_1.1
Use Case Name	Scale Up for a Larger Number of Orders to Handle
Means for validation /verification	A communication infrastructure among WMS, Production System Simulator and PLC Simulator is built: • The input buffer status is full or not modifying ad-hoc its status. • The Production System Simulator introduces a pallet.
Models/Metrics	

Precondition	The warehouse processes a limited (bounds set by design) number of pallets.	
Primary Actor	-	
Secondary Actors	-	
Dependencies	-	
Generalization	-	
Basic Flow	**Steps**	
	1	The Production System introduces a pallet into the warehouse by the production system.
	2	WMS **VALIDATES THAT** the input buffer has free space to store pallets.
	3	WMS sends the order to enter the new pallet into the input buffer.
	Post-condition: The pallet is located at the input buffer.	
Specific Uncertainty Alternative Flow (buffer hasn't got free space)	**RFS BF 2**	
	1	WMS **VALIDATES THAT** the input buffer has not got any free space to store pallets.
	2	The Pallet waits indefinitely for free space in the buffer.
	3	**ABORT**
	Post-condition: Material flow stops are propagated upstream towards the production system.	
Additional Information	-	
Responsibility	Person A	
Change History	2015-05-20 Person B – First Version 2015-06-01 Person A – Reviewed 2015-07-28 Person A – Refined Version	

Q1. What do you believe about **Production System** mentioned in Table. Appx-6?
Actor (Third party) ☐ or Part of Handling system ☐

Q2. What do you believe about **Pallet** mentioned in Table. Appx-6?
Actor (Third party) ☐ or Part of Handling system ☐

Q3. What do you believe about **WMS** mentioned in Table. Appx-6?
Actor (Third party) ☐ or Part of Handling system ☐

Q4. Do you believe the reason/source of **Specific Uncertainty Alternative Flow** is unknown?
Yes, I have no idea at all ☐
Yes, I have some uncertain idea, but not complete. ☐ Please specify what you know.

No, I know it. ☐ Please specify what you know.

Q5. Do you believe the size of **pallet** can introduce ***uncertainty***?
Yes ☐ No ☐
If no, does any other property of **pallet** can cause uncertainty? Please specify if possible.

Q6. Do you believe that **Production System** introduces the **pallet** always one by one?
Yes ☐ No ☐

Q7. Please refer to step 3 in **Error! Reference source not found.**, "WMS sends the order to enter the new pallet into the input buffer".

1) Is it possible that "WMS does not send the order to enter the new pallet into the input buffer at all"?
If yes, the reason is related to
WMS ☐ Pallet ☐ Input buffer ☐ none of them, please specify _____
2) Is it possible that "WMS sends the order to enter the new pallet into the input improperly"?
If yes, the reason is related to
WMS ☐ Pallet ☐ Input buffer ☐ none of them, please specify _____
3) Is there any other possibility?
If yes, please specify_____

Q8. Do you believe any ***Time constraints*** may cause uncertainty in this case?
If yes, please specify _____
If yes, do you believe it is necessary to consider in this use case? Yes ☐ No ☐

Q9. Do you believe any ***Nature phenomena*** may cause ***uncertainty*** in this case?
If yes, please specify _____
If yes, do you believe it is necessary to consider in this use case? Yes ☐ No ☐

Q10. Do you believe any ***human behavior*** may cause ***uncertainty*** in this use case?
If yes, please specify _____
If yes, do you believe it is necessary to consider in this use case? Yes ☐ No ☐

B.2 Example of Refined Uncertainty Requirements in RUCM and Corresponding Questions

Belief Specification

Belief Spec. Name	Scale up for a large number of Orders to Handle
Brief Description	None
Precondition	None
Primary Actor	Production System
Secondary Actors	Pallet, Input buffer

Basic Flow (Untitled) ▼	Steps	
	1	The Production System asks to introduce a pallet into warehouse to WMS.
	2	WMS VALIDATES THAT the input buffer has free space to store the pallets.
	3	WMS sends the response to Production System.
	4	The Production System introduce a pallect into warehouse.
	5	WMS sends the order to enter the new pallet into the input buffer.
	Postcondition	The pallet is located at the input buffer.

Fig. Appx-5. Uncertainty Requirement specified in RUCM (a revised version of Table. Appx-6)

Please refer to Fig. Appx-5,

Q11. Do you believe that uncertainties exist in **Step 1**?
1) Yes ☐ No ☐
If yes, how many ***uncertainties*** do exist?_____
Please specify these ***uncertainties***,_____,
2) For each uncertainty, please specify its type?
Occurrence ☐ ***Content*** ☐ ***Time*** ☐ ***Environment*** ☐ ***Geographical Location*** ☐ ***Others*** ☐
3) Do you believe this ***uncertainty*** can be measured?
Yes ☐ No ☐ If no, please specify the reason._____
If yes, please answer the following question.
a. What do you believe the likelihood of this ***uncertainty*** is?
Please specify the probability _____if possible, otherwise please select one of the following options:

Rare	Unlikely	Possible	Likely	Almost Certain

☐	☐	☐	☐	☐

i. Do you have any evidence to support?

Yes ☐ No ☐ if yes, please specify _____.

b. What level of risk is associated with this type of ***uncertainty***?

Please specify the exact percentage _____(0-100, 100 is the highest level of risk) if possible; otherwise please select one of the options:

Very Low	Low	Medium	High	Extreme
☐	☐	☐	☐	☐

i. Do you have any evidence to support?

Yes ☐ No ☐ if yes, please specify _____.

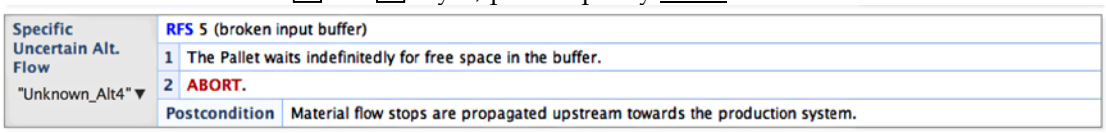


Fig. Appx-6. Uncertainty RFS 2

Q12. Please refer to Fig. Appx-6, did you specify this ***uncertainty*** as above?

1) Yes ☐ No ☐

If no, do you believe it is real? Yes ☐ No ☐. If no, please specify the reason_____

If it is real, is it necessary to consider? Yes ☐ No☐. If no, please specify the reason_____

If it is real and it is necessary to consider, please answer the following

2) For each uncertainty, please specify its type?

Occurrence ☐ ***Content*** ☐ ***Time*** ☐ ***Environment*** ☐ ***Geographical Location*** ☐ ***Others*** ☐

3) Do you believe this ***uncertainty*** can be measured?

Yes ☐ No ☐ If no, please specify the reason_____

If yes, please answer the following question.

a. What do you believe the likelihood of this ***uncertainty*** is?

Please specify the probability _____if possible, otherwise please select one of the following options:

Rare	Unlikely	Possible	Likely	Almost Certain
☐	☐	☐	☐	☐

i. Do you have any evidence to support?

Yes ☐ No ☐ if yes, please specify _____.

b. What level of risk is associated with this type of ***uncertainty***?

Please specify the exact percentage _____(0-100, 100 is the highest level of risk) if possible; otherwise please select one of the options:

Very Low	Low	Medium	High	Extreme
☐	☐	☐	☐	☐

ii. Do you have any evidence to support?

Yes ☐ No ☐ if yes, please specify _____.

Appendix C. An Example of BUCS Specified with the U-RUCM Editor

Belief Use Case Specification

Use Case Name	Scale Up for a Larger Number of Orders to Handle
Brief Description	None
Primary Actor	Production System
Secondary Actors	Input buffer,Pallet, Operator
Dependency	**INCLUDE USE CASE** WMS Verifies the Input Pallet
Generalization	None
Belief Agent(s)	CompanyA
Timepoint and Duration	March–2016, After
Belief Degree	UModel.Measure.Vagueness::Likely
Indeterminacy Source(s)	**REF** Broken Input Buffer, **REF** Connection Loss, **REF** Improper Implementation for Adaption of Third part Application
Evidence	**REF** Execution Log
Belief Precondition	None

Belief Basic Flow "Normal"▼	**Steps**
	1 Production System executes the input order.
	2 **DO**
	3 Production System inquires of introduction of a pallet into warehouse.
	4 WMS **VALIDATES THAT** the input buffer has free space to store pallets.
	5 WMS sends the introduction response to Production System.
	6 Production System introduces a pallet.
	7 **INCLUDE USE CASE** WMS Verifies the Input Pallet
	8 WMS sends the order to MFC for entering the new pallet into the input buffer.
	9 **INCLUDE USE CASE** MFC Handles Input Order
	10 **UNTIL** No more introduction inquire.
	11 WMS displays the order is finished.
	Postcondition The order is finished.

Belief Specific Alternative Flow (URFS) "BAlt1"▼	**URFS** Normal 1
	A1 Production System executes the input order improperly.
	1 WMS watis for the input pallet.
	2 **ABORT**.
	Postcondition None

Belief Specific Alternative Flow (RFS) "BAlt2"▼	**RFS** Normal 3
	1 WMS waits for the free space.
	2 WMS **VALIDATES THAT** the input buffer has free space to store pallets.
	3 **RESUME STEP** Normal 5
	Postcondition The input buffer has free space.

Belief Specific Alternative Flow (URFS) "BAlt3"▼	**URFS** BAlt2 1
	A1 WMS displays the time out of waiting for the free space.
	1 Operator checks the input buffer.
	2 **ABORT**.
	Postcondition The order is not finished.

Belief Specific Alternative Flow (URFS) "BAlt4"▼	**URFS** Normal 6
	A1 Production System displays the time out of introducing the pallet.
	1 Operator checks the problem manually.
	2 **ABORT**.
	Postcondition The order is not finished.

Belief Specific Alternative Flow (URFS) "BAlt5"▼	**URFS** Normal 8
	A1 WMS sends the pallet to the rejection station.
	1 **RESUME STEP** Normal 2
	Postcondition The pallet is rejected.

Fig. Appx-7. Belief Use Case Specification *Scale up for a large number of orders*

Fig. Appx-7 presents a sanitized belief use case specification of the AW case study specified with the *U-RUCM* Editor.

Appendix D. tolveR-E

Table. Appx-7 presents definitions of heuristics of tolveR-E. Note that **ocl.evaluate** is a function that is used to evaluate the constraint specified in OCL. The result of the evaluation is either *true* or *false*. The **catch TriggerException** represents the exception in case none of the specified triggers occur. For example, $S1 \xrightarrow{Tran1} S1$, the event of the trigger of Tran1 is kind of TimeEvent and the effect of Tran1 is *"throw new TriggerException(S1.name)"*.

Table. Appx-7. Definitions of heuristics of tolveR-E

#	Definition in OCL	Suggested Action
R1	**context** State not **ocl.evaluate**(**self**.stateInvariant)	One of R1.1, R1.2, or R1.3 will be selected.
R1.1	State.allInstance->excludes(**self**)->select(s:State | **ocl.evaluate**(s.stateInvariant))->size()=0	Modify this State or Add an unknown State with applied «BeliefElement»
R1.2	State.allInstance->excludes(**self**)->select(s:State | **ocl.evaluate**(s.stateInvariant))->size()=1	Add a new Transition with the same Trigger with applied «BeliefElement»
R1.3	State.allInstance->excludes(**self**)->select(s:State | **ocl.evaluate**(s.stateInvariant))->size()>1	Check the redundant problem, and add the transitions with the same Trigger with applied «BeliefElement» to these states if they are correct
R2	**context** State **catch TriggerException** and **ocl.evaluate**(**self**.stateInvariant)	One of R2.1, R2.2, or R2.3 will be selected.
R2.1	**context** State self.outgoings->exists(t:Transition | t.triggers->exists(t:Trigger | t.event.**ockIsKindOf**(CallEvent)))	Check invocation of operation
R2.2	**context** State self.outgoings->exists(t:Transition | t.triggers->exists(t:Trigger | t.event.**ockIsKindOf**(SignalEvent))	Check composite structure diagram and state machine of driver
R2.3	**context** State self.outgoings->exists(t:Transition | t.triggers->exists(t:Trigger | t.event.**ockIsKindOf**(TimeEvent)))	Check the TimeExpression
R2.4	**context** State self.outgoings->exists(t:Transition | t.triggers->exists(t:Trigger | t.event.**ockIsKindOf**(ChangeEvent)))	Check the ChangeExpression
R3	**context** State **catch TriggerException** and not **ocl.evaluate**(**self**.stateInvariant)	One of R3.1, R3.2, or R3.3 will be selected.

R3.1	Refer to R1.1	Add an unknown transition to an unknown state with applied «BeliefElement»
R3.2	Refer to R1.2	Add an unknown transition to a known state with applied «BeliefElement»
R3.3	Refer to R1.3	Check the redundant problem, and add the unknown transitions to these states if they are correct.
R4	**context** Transition not **self**.guards->forAll(c:Constraint | **ocl.evaluate**(c))	One of R4.1 or R4.2 will be selected.
R4.1	**context** Transition **self**.triggers->exists(t:Trigger | t.event.**ockIsKindOf**(CallEvent))	Modify the guard of call event / Add new transition with applied «BeliefElement»
R4.2	**context** Transition **self**.triggers->exists(t:Trigger | t.event.**ockIsKindOf**(SignalEvent))	Modify the guard of signal event/ Add new transition with applied «BeliefElement»/ Check the signal from DM

Appendix E. tolveR-D

Table. Appx-8 presents definitions of heuristics of tolveR-D.

The value ranges to make constraint *true* is represented as below,

$$C(x_0, \dots x_n) = C_0(x_0) \cap \dots \cap C_n(x_n)$$

The possible situations whereby the invariant needs to be modified are described as follows (C^{org} represents the original constraint, and C^{dai} represents the invariant from daikon). Note that any of them should apply «BeliefElement» by default.

Table. Appx-8. Definitions of heuristics of tolveR-D

#	Description	
D1	$C^{org} \supset C^{dai}$, we suggest	
	1	The variables in both constraints are the same. For example, S1 $\xrightarrow{Tran1}$ S2, the state invariant of S2 is {x > 1}, then the invariant from Daikon is {x > 2}, so 1) Split S2 into two states with the same trigger, S1 $\xrightarrow{Tran1}$ S2.1 {x > 2} and S1 $\xrightarrow{Tran1}$ S2.2 {x > 1 and x ≤ 2}; 2) Modify the state invariant of S2 to {x > 2}; 3) No change
	2	The number of variables is more than the original constraints. For example, S1 $\xrightarrow{Tran1}$ S2, the state invariant of S2 is {x > 1}, then the invariant from Daikon is {x > 1 and y = 0}, so 1) Split S2 to two states with the same trigger, S1 $\xrightarrow{Tran1}$ S2.1 {x > 1 and y = 0} and S1 $\xrightarrow{Tran1}$ S2.2 {x > 1 and y ≠ 0}, 2) Modify the state invariant of S2 to {x > 1 and y = 0}; 3) No change
D2	$C^{org} \subset C^{dai}$, we suggest	

	1	The variables are the same. For example, $S1 \xrightarrow{Tran1} S2$, the state invariant of S2 is $\{x > 1\}$, then the invariant from Daikon is $\{x > 0\}$, so 1) Merge relevant states which may reach (e.g. $S2 \xrightarrow{Tran2} S3$ $\{x > 0$ and $x < 1\}$) to a composite state $S1 \xrightarrow{Tran1} \{S2 \xrightarrow{Tran2} S3\}$, 2) Modify the state invariant of S2 to $\{x > 0\}$; 3) No change
	2	The number of variables is less than in the original constraints. For example, $S1 \xrightarrow{Tran1} S2$, the state invariant of S2 is $\{x > 1$ and $y = 0\}$, then the invariant from Daikon is $\{x > 1\}$, so 1) Merge relevant states which may reach ($S2 \xrightarrow{Tran2} S3$ $\{x > 1$ and $z > 2\}$, $S2 \xrightarrow{Tran3} S3$ $\{x > 1$ and $t = 0\}$) to a composite state $S1 \xrightarrow{Tran1} \{S2 \xrightarrow{Tran2} S3, S2 \xrightarrow{Tran3} S3\}$, 2) Modify the state invariant of S2 to $\{x > 1\}$; 3) No change
D3	$C^{org} \cap C^{dai} \neq \emptyset$ and $C^{org} \not\subset C^{dai}$ and $C^{org} \not\supset C^{dai}$, we suggest	
	1	The variables are the same. For example, $S1 \xrightarrow{Tran1} S2$, the state invariant of S2 is $\{x > 1\}$, then the invariant from Daikon is $\{x < 2\}$, so 1) Make intersections of C^{org} and C^{dai}, then $S1 \xrightarrow{Tran1} S2\{x > 1$ and $x < 2\}$, 2) Use constraint from Daikon $S1 \xrightarrow{Tran1} S2\{x < 2\}$; 3) No change
	2	The variables are different. For example, $S1 \xrightarrow{Tran1} S2$, the state invariant of S2 is $\{x > 1$ and $y = 0\}$, then the invariant from Daikon is $\{y = 0$ and $z > 2\}$, so 1) Make intersections of C^{org} and C^{dai}, then $S1 \xrightarrow{Tran1} S2\{x > 1$ and $y = 0$ and $z > 2\}$, 2) Use the constraint from Daikon $S1 \xrightarrow{Tran1} S2\{y = 0$ and $z > 2\}$; 3) No change
D4	$C^{org} \cap C^{dai} = \emptyset$, we suggest	
	1	The variables are the same. For example, $S1 \xrightarrow{Tran1} S2$, the state invariant of S2 is $\{x > 1\}$, then the invariant from Daikon is $\{x < 0\}$, so 1) the first solution is to make unions of C^{org} and C^{dai}, then $S1 \xrightarrow{Tran1} S2$ $\neg\{x \leq 1$ and $x \geq 0\}$, 2) use constraint from Daikon $S1 \xrightarrow{Tran1} S2\{x < 0\}$; 3) unchanged
	2	The variables are different. For example, $S1 \xrightarrow{Tran1} S2$, the state invariant of S2 is $\{x > 1\}$, then the invariant from Daikon is $\{y = 0\}$, so 1) Make unions of C^{org} and C^{dai}, then $S1 \xrightarrow{Tran1} S2$ $\neg\{x \leq 1$ and $y \neq 0\}$, 2) Use constraint from Daikon $S1 \xrightarrow{Tran1} S2\{y = 0\}$; 3) No change