SoftwareX 26 (2024) 101676

Contents lists available at ScienceDirect

SoftwareX

journal homepage: www.elsevier.com/locate/softx

Original software publication

# UTypes: A library for uncertain datatypes in Python

Carlos Javier Fernández-Candel, Paula Muñoz *, Javier Troya, Antonio Vallecillo

*ITIS Software, Universidad de Malaga, Spain*

ARTICLE INFO

*Keywords:*
Uncertainty
Datatypes
Belief
Subjective logic
Libraries

ABSTRACT

Existing Python uncertainty packages support the expression and propagation of uncertainty in numeric types, such as `float` or `int`. However, they do not cover the rest of the built-in types which can also be affected by uncertainty when representing physical systems. The Uncertain Datatypes (`UTypes`) library provides extensions of Python built-in datatypes `bool`, `int`, `float`, `enum` and `str` to seamlessly incorporate the data uncertainty coming from physical measurements or user estimations into Python programs, along with the set of operations defined for the values of these types. The library implements in a natural and efficient manner linear error propagation theory in Python and performs uncertainty calculations analytically.

## Code metadata

| | |
|---|---|
| Current code version | v1.1.5 |
| Permanent link to code/repository used for this code version | https://github.com/ElsevierSoftwareX/SOFTX-D-23-00654 |
| Permanent link to Reproducible Capsule | - |
| Legal Code License | MIT License |
| Code versioning system used | git |
| Software code languages, tools, and services used | Python |
| Compilation requirements, operating environments & dependencies | Python 3.11, numpy |
| If available Link to developer documentation/manual | https://github.com/atenearesearchgroup/uncertainty-datatypes-python/blob/master/README.md |
| Support email for questions | atenea@itis.uma.es |

## Software metadata

| | |
|---|---|
| Current software version | v1.1.5 |
| Permanent link to executables of this version | https://pypi.org/project/uncertainty-datatypes/ |
| Permanent link to Reproducible Capsule | - |
| Legal Software License | MIT License |
| Computing platforms/Operating Systems | Unix-like, Microsoft Windows. |
| Installation requirements & dependencies | Python 3.11, numpy |
| If available, link to user manual - if formally published include a reference to the publication in the reference list | https://github.com/atenearesearchgroup/uncertainty-datatypes-python/blob/master/docs/UserGuide.md |
| Support email for questions | atenea@itis.uma.es |

## 1. Motivation and significance

Uncertainty is an inherent property of any system operating in a real environment or interacting with physical elements [1]. These systems have to cope with issues such as inaccurate measurements; lack of knowledge about the system or its environment; incorrect,

* Corresponding author.
*E-mail addresses:* cjferna@uma.es (Carlos Javier Fernández-Candel), paulam@uma.es (Paula Muñoz), jtroya@uma.es (Javier Troya), av@uma.es (Antonio Vallecillo).

https://doi.org/10.1016/j.softx.2024.101676
Received 29 September 2023; Received in revised form 1 February 2024; Accepted 27 February 2024
Available online 5 March 2024

incomplete or vague information; unreliable data sources or numerical approximations [2].

The need to deal explicitly with uncertainty has become evident in many fields, such as self-adaptive systems [3–6], web of things applications [7], cyber–physical Systems [8], simulation [9,10], knowledge representation [11], event processing systems [12], software modeling [13–18] or model transformations [19,20], among others. These works deal with uncertainty in several forms, using different notations and mechanisms. However, although several modeling and programming languages allow the representation of uncertainty for describing system attributes [21], these aspects are not normally incorporated into their type systems. Thus, operating with uncertain values and propagating uncertainty are normally cumbersome processes, difficult to implement by programmers and software engineers [14].

To address this problem, specialized libraries provide automated support for uncertainty propagation. For example, the Python `uncertainties` package [22] supports uncertainty propagation, variable correlation, and derivatives. `Soerp` [23] is an uncertainty package that supports second-order analysis of error propagation and uncertainty quantification. The number of uncertainty propagation packages is large, each one focusing on specific aspects [24].

The problem with these implementations is threefold. First, these packages only allow associating uncertainty with real numbers, neglecting the uncertainty that can affect other essential built-in datatypes such as booleans, enumerations or strings [14]. Second, their comparison operators are not expressive enough: the comparison of uncertain real numbers returns a Boolean value, when it should be a probability. For example, if $x = 2.0 \pm 0.3$ and $y = 2.5 \pm 0.25$, the result of comparing $x < y$ should be 0.893, and not a simple true or false value [14]. Finally, most of them trade precision for performance, which sometimes hinders their use when employed in computationally intensive applications such as iterative methods [10,25], even when such high accuracy is not needed.

This paper presents the Uncertain Types (`UTypes`) Python package that successfully addresses these limitations.

## 2. Software description

The `UTypes` library is implemented as a Python package, `utypes.py`, which provides a set of datatypes (`ubool`, `sbool`, `uint`, `ufloat`, `uenum` and `ustr`) that extend the Python built-in datatypes (`bool`, `int`, `float`, `enum` and `str`) with uncertainty. The library implements linear error propagation theory in Python. The uncertain datatypes provided by the library is well suited for applications that require the basic mechanisms for the propagation of uncertainty, efficient computation, and a closed algebra of datatypes. The extension mechanisms also ensure that the extended operations work as expected when values of original types are given as input parameters.

Users can install the corresponding `PyPi` package [26] using the `pip` command "`pip install uncertainty-datatypes`" and then import the module by using "`from uncertainty.utypes import *`".

A distinctive feature of the `UTypes` library is that it also incorporates Subjective logic [27] (type `sbool`). Subjective logic is a logic that explicitly takes epistemic uncertainty and trust into account. It is an extension of probabilistic logic (type `ubool`), which in turn is an extension of Boolean logic (type `bool`). That is, `UTypes` implements the type hierarchy `bool <: ubool <: sbool` [15]. This enables the seamless combination of different types of uncertainties under the same library, and in particular the representation and combination of first- and second-order uncertainties, including probabilities, belief and trust.

Correlations between expressions with uncertainty are not taken into account by default in the `UTypes` library. This saves having to keep track of all correlations between quantities (variables and functions) at all times, improving computational performance. This implies that, by default, we assume that variables are independent. Otherwise, the `UTypes` library also supports the specification of correlations between the variables involved in the mathematical operations of types `uint` and `ufloat`. This allows dealing with dependent variables if necessary.

The derivatives of mathematical expressions are not automatically handled by the library either. Again, this saves keeping track of the value of derivatives, something that also impacts performance. Other unsupported features include automatic handling of arrays of uncertain numbers and higher-order analysis of error propagation.

### 2.1. Software architecture

The `UTypes` library is implemented in the form of a Python package that provides one module, `utypes.py`, which includes the five submodules with the implementation of the supported uncertain datatypes: `unumbers.py`, `ubool.py`, `sbool.py`, `ustr.py` and `uenum.py`. This section briefly describes the basic types provided by each module. For a detailed description of these types, their values, and operations, please refer to the `UTypes` User Guide [28].

*Uncertain numerical values.* Module `unumbers.py` defines `ufloat` and `uint` classes that represent uncertain reals and integers, respectively. Values of these two types are represented by pairs `(x,u)` where `x` is the numerical (nominal) value and `u` its associated uncertainty. Thus, value `ufloat(3.5,0.1)` represents the uncertain real number $3.5 \pm 0.1$, and `uint(30,1)` represents the uncertain integer $30 \pm 1$. This representation of uncertainty for numerical values follows the *ISO Guide to Measurement Uncertainty* [2], where values are represented by the mean and standard deviation of the assumed probability density function describing how measurements of the ground truth value are distributed.

*Uncertain boolean values.* Module `ubool.py` defines the `ubool` class that extends type `bool` by using probabilities instead of the traditional truth values and replacing truth tables with probability formulas. Thus, an `ubool` value expresses a probability representing the degree of belief (i.e., the *confidence*) that a given statement is true. For example, value `ubool(0.7)` means that there is a 70% chance that a fact is true or that an event will occur. Python `bool` values `True` and `False` correspond to `ubool(1.0)` and `ubool(0.0)`, respectively. Moreover, `ubool` values can be used instead of `bool` values. In this case, a user-defined certainty threshold determines when a given probability can be considered to be `True` or `False`. The default value of the threshold is 0.9, but can be changed using `ubool.setCertainty(c:float)` and queried using method `ubool.getCertainty()`.

*Subjective logic.* The `sbool.py` module defines the `sbool` class, which provides an extension of `ubool` to represent *binomial opinions* in Subjective Logic [27]. They allow expressing degrees of belief with epistemic uncertainty and also trust. A *binomial opinion* $w_X^A = (b, d, u, a)$ about a given fact $X$ by a belief agent $A$ is represented as a quadruple "`sbool(b,d,u,a)`" where: $b$ is the degree of belief that $X$ is True; $d$ is the degree of belief that $X$ is False; $u$ is the amount of uncommitted belief (i.e., epistemic uncertainty); and $a$ is the prior probability in the absence of belief or disbelief. These values are all real numbers in the range [0, 1], and satisfy that $b + d + u = 1$. The *projection* of a binomial opinion is defined as $P = b + au$.

*Uncertain strings.* Module `ustr.py` defines the `ustr` class that is used to represent strings with uncertainty. More precisely, type `ustr` extends Python type `str`, adding to their values a degree of confidence in the contents of the string. This is useful, for example, when rendering strings obtained by inaccurate OCR devices or texts translated from other languages if there are doubts about specific characters or words. Values of type `ustr` are pairs `(s,c)`, where `s` is the nominal string and `c` the associated confidence (a real number between 0 and 1). To calculate the confidence of a string `s`, the *Levenshtein distance* is normally used. For example, `ustr('hell0 world!',0.92)` means that we do not trust at most one of the 12 characters of the string.

*Uncertain enumerations.* Module `uenum.py` defines the `uenum` class, which is the embedding supertype for Python type `enum` that adds uncertainty to each of its literals. A value of an uncertain enumeration type `enum` is not a single literal, but a set of pairs $\{(l_1, c_1), \ldots, (l_n, c_n)\}$, where $\{l_1, \ldots, l_n\}$ are the enumeration literals and $\{c_1, \ldots, c_n\}$ are numbers in the range [0, 1] that represent the probabilities that the variable takes each literal as its value. They should fulfill that $c_1 + \cdots + c_n = 1$. Pairs whose confidence $c_i$ is 0 can be omitted. The literals of an `uenum` value can be accessed using the property `literals`. A copy of the dictionary can be obtained with the property `elements`. For example, if `x = uenum(''Red'':0.7'',Blue'':0.1)` then `x.literals=[''Red'','',Blue'']` and `x.elements={''Red'': 0.7'',Blue'':0.1}`.

### 2.2. Software functionalities

The `UTypes` library provides all the basic operations for the newly defined uncertain types, as well as type projections and embeddings to and from the built-in types.

Whenever possible, we have defined three concrete syntaxes for all operations: *infix* operators (e.g., `x+y`), instance *methods* (`x.add(y)`), and *functions* (`add(x,y)`). The infix operators respect the Python syntax, so the values and variables of the new types can be used seamlessly in usual Python expressions.

This has worked well except for the Python infix logical operators (`and`, `or`, `xor`, `not`, etc.) because Python does not allow to overload them [29] and have a different meaning when used with objects. Therefore, new symbols have been defined ("`&`" for `and`, "`|`" for `or`, "`^`" for `xor`, "`~`" for `not`, etc.) that should always be used in logical operations that involve `ubool` values. To avoid possible confusion, these infix operators also accept a textual notation (e.g., `x |AND| y`) that is preferred. Of course, all `ubool` operators work as expected when used with `bool` values.

Python mathematical operations for numerical values (`abs`, `sqrt`, `sin`, `floor`, etc.) are supported for uncertain values, too. Similarly, all operations on `str` values are supported on `ustr` values and variables.

The infix comparison operators (`<`,`<=`,`==`,`!=`,`>=`,`>`) are also supported in all uncertainty types. They return `ubool` values when their arguments involve uncertain types, otherwise they return a `bool` value.

Finally, the library supports all the operations defined in Subjective logic for dealing with subjective opinions, including the logical operations (which extend the `ubool` operations), trust functions (such as discount and referral), and all fusion operators used to combining opinions in order to reach consensus [27].

See the `UTypes` User Guide [28] for a complete description of all supported operations and their syntaxes.

## 3. Illustrative examples

The following examples[1] illustrate the use of the uncertain types provided by the `UTypes` library.

*Basic operations.* Listing 1 shows some examples of basic operations with uncertain types. Results are shown as comments. The first example combines boolean and uncertain boolean in a logical expression. The second one uses numerical variables. The last two show comparisons between uncertain variables, one using the comparison operator '`<`' and one using the `max()` operation.

```
a = ubool(0.3); b = ubool(0.8); c = ubool(0.5); d = False
w = (~a & b) |IMPLIES| (c | d) ; print (w)
# ubool(0.720)

x = ufloat(2.0,0.3); y = ufloat(2.5,0.25); z = 1.5
print ((x / y)**2 + z)
# ufloat(2.140,0.329)

print (x < y)
# ubool(0.639)

print(max(sin(ufloat(53.34,0.8)),uint(23,1),ufloat(24.4,0.8),23.45))
# ufloat(24.0, 0.800)
```

**Listing 1:** Some basic operations with uncertain values.

*A smart home.* Consider a smart home with sensor-equipped rooms measuring temperature and humidity. The sensors have a known accuracy of $\pm 0.5$ degrees for temperature and $\pm 1.0\%$ for humidity. A room control system decides whether to activate the air conditioning (AC) based on sensor readings. If the temperature exceeds 25 degrees or humidity surpasses 80%, the AC is automatically activated; otherwise, it remains off. Listing 2 shows the code that implements such a decision.

```
def ac_on() -> ubool:
    return (temp >= 25.0) | (humidity >= 80.0)
```

**Listing 2:** Condition to turn the AC system on.

Using this operation, if `temp` is $25.5 \pm 0.5$ and `humidity` is $79.0 \pm 1$, the result of the `ac_on()` operation is `uboolean(0.867)`, i.e., the AC system should be switched on with a probability of 0.867. Assuming that our threshold is 0.85, we decide to switch it on.

*Adding trust to the AC system using subjective logic.* Now, consider a house occupant named Ada who does not trust the accuracy of the kitchen's temperature readings because she knows that the sensor is placed near the oven, which causes occasional inaccuracies. Ada's skepticism introduces subjective uncertainty into the system's decision. Bob and Cam are two other house occupants whose trust in the sensors differs from that of Ada. Bob is quite certain that the readings are incorrect because he had to turn off the AC yesterday while cooking. In turn, Cam trusts the sensor readings. Their individual trusts can be expressed using binomial opinions: `ada=sbool(0.3,0.2,0.5,0.867)`, `bob=sbool(0.0,0.9,0.1,0.867)`, and `cam=sbool(0.9,0.0,0.1, 0.867)`. In order to make a decision, we can use the Weighted Belief Fusion (WBF) operator that combines them to reach a consensual decision `sbool.weightedFusion([ada,bob,cam])`. The resulting opinion is `sbool(0.442,0.437,0.121,0.867)`, whose projection is `ubool(0.547)` with a degree of uncertainty of 0.121. This means that we better switch the AC system off, with sufficient confidence to do it—note how we are able to quantify the uncertainty of the decision, too.

*Resource utilization.* We also studied the performance of our library, in terms of execution time and memory utilization. We conducted a series of measurements for the different operations using the `UTypes` library and compared the resulting times with those obtained with the `uncertainties` package. First, we repeatedly executed the basic operations of type `ufloat`: `+`, `-`, `*`, `/`, and `**`, accumulating the results on a variable. Each execution performed 8 operations between 10 and 50 thousand times. Each test was repeated 30 times, calculating the geometric mean to reduce the effects of outliers [30]. Peak memory consumption was measured in each test. Columns 3–7 of Table 1 show the resulting times (in seconds), while column 8 (S.up) shows the average speedup obtained by our proposal. The last two columns show the peak memory consumption (in Mb) and the ratio between them. Roughly, our library achieves a speedup between 2.55× and 3.80× higher than the `uncertainties` package, which requires between 8 and 14 times more memory than ours. This increase in memory consumption is due to the logic of the `uncertainties` package, which

[1] The code of these examples can be found at https://github.com/atenearesearchgroup/uncertainty-datatypes-python/tree/master/examples.

**Table 1**
Performance comparison figures.

| op | Library | 10K | 20K | 30K | 40K | 50K | S.up | Mem | Ratio |
|---|---|---|---|---|---|---|---|---|---|
| + | UTypes | 0.089 | 0.126 | 0.162 | 0.197 | 0.232 | 3.68 | 17.20 | 8.32 |
| | Uncertaint. | 0.227 | 0.525 | 0.605 | 0.704 | 0.854 | | 143.03 | |
| – | UTypes | 0.091 | 0.133 | 0.171 | 0.209 | 0.246 | 3.08 | 17.23 | 8.30 |
| | Uncertaint. | 0.224 | 0.362 | 0.487 | 0.643 | 0.759 | | 143.08 | |
| * | UTypes | 0.109 | 0.161 | 0.208 | 0.254 | 0.299 | 2.55 | 17.21 | 8.89 |
| | Uncertaint. | 0.251 | 0.379 | 0.507 | 0.635 | 0.762 | | 152.94 | |
| / | UTypes | 0.147 | 0.211 | 0.271 | 0.330 | 0.388 | 3.20 | 17.29 | 10.61 |
| | Uncertaint. | 0.311 | 0.481 | 0.888 | 1.129 | 1.240 | | 183.37 | |
| ** | UTypes | 0.154 | 0.215 | 0.277 | 0.337 | 0.399 | 3.80 | 17.35 | 13.84 |
| | Uncertaint. | 0.547 | 0.804 | 1.048 | 1.285 | 1.516 | | 240.03 | |
| Incub. | UTypes | 0.284 | 0.400 | 0.586 | 0.711 | 0.826 | 2.55 | 17.33 | 14.69 |
| | Uncertaint. | 0.940 | 1.169 | 1.325 | 1.525 | 1.742 | | 254.54 | |

stores the correlation between the uncertain variables. We also compared our proposal with a real implementation of a digital twin of a real plant incubator [31], for which simulations of different lengths (i.e., number of iterations) were run. Each iteration performed a combination of additions (2), subtractions (3), multiplications (7), divisions (2) and comparisons (2), obtaining similar results (cf. last two columns of Table 1). Tests were conducted on a computer with an Intel Core i7-1255U@1.70 GHz processor, 32 GB of RAM, and Windows 10 Pro.

## 4. Impact

The `Utypes` uncertainty library is the Python version of a Java uncertainty library[2] that we developed 6 years ago. Before we ported it, the original Java library had already been successfully used in several research projects and tools.

- Together with the TU Wien, the library was originally used to provide support for *quantities* in UML and other software modeling languages [32,33].
- Recently, we used it in a joint project with the Mälardalen University (Sweden) in two industrial settings to improve the consistency checking processes of their software artifacts, using epistemic uncertainty to prioritize conflicts and uncover previously hidden potential inconsistencies [34].
- The uncertain types are also used to represent and operate with Belief uncertainty in Software models. As part of a joint initiative with the Open University of Catalonia and CEA-LIST (France), a UML profile was developed to support Subjective logic opinions and their operations. We demonstrated it with several applications from diverse domains, including digital humanities, digital stores, software engineering and smart houses [13].
- The library was used to extend a Complex Event Process (CEP) engine with uncertainty information about the events and their properties [12]. The uncertain CEP engine was later used in one of our projects to implement a trust-based and uncertainty-aware communication framework [35].
- Together with the group from the University of Bremen, the developers of the tool USE (UML Specification Environment) [36], we extended the UML and OCL type systems with uncertainty, and implemented them in the USE tool using the library [15,37].
- The library has been used to implement uncertainty-aware simulations of PID controllers, together with several European research groups, with very successful results [10,25].
- The library is also being used in the extension of Knowledge graphs called Subjective Knowledge Graphs [11]. They are now used to classify police reports on property crimes [38].
- Finally, the library has been used in different applications that we have developed in domains such as the Web of Things [7], Software Modeling [14,15,39] and Model transformations [20].

The library has also been used is several research projects funded in public calls.

- In the research project "Digital Avatars[3]: A Framework for Collaborative Social Computing Applications" (2018–22), the library was used to implement all the trust and reputation mechanisms required in the communication between peer-to-peer applications running in smartphones [35,40].
- In the research project "IPSCA[4]: Including people in smart city applications" (2022–26), uncertain types are used to deal with the agents' beliefs about the environment they perceive and the models they use [13].

Last year we decided to reimplement the library in Python 3 because most of the applications that required handling uncertain data types were written in Python. In particular, PID controller simulation applications and CEP engines are mostly developed in Python nowadays. The existing uncertainty packages in Python were rather slow for these applications, and we were asked to port our Java library to Python. Moreover, the Java implementation was rather verbose due to its prefix syntax based only on methods chaining. The results of the Python implementation in terms of usability and performance are satisfactory and, in addition, the usability of our library has significantly improved because the numerical operators in Python can be overloaded. This feature allowed a very natural extension of the Python built-in datatypes.

## 5. Conclusions

In this paper we have presented the `UTypes` Python 3 library, which extends the Python built-in datatypes with uncertainty, implementing linear error propagation. Both measurement and belief uncertainty are supported by the extended datatypes `ubool`, `sbool`, `uint`, `ufloat`, `uenum` and `ustr`. The library is well suited for applications that require basic mechanisms to propagate uncertainty efficiently, and a closed algebra of datatypes.

The library has the potential to impact in various domains where uncertainty plays a key role, in particular CPS applications, Digital Twins, Industry 4.0, the Web of Things, Smart cities, and in general in all applications that operate in physical environments or interact with human beings.

As the library gains recognition and grows its user base, we expect new needs to emerge. Right now we are working on the Type-A implementation of numerical uncertainties [2], where uncertain values are represented by $n$ independent observations $X = x_1, \ldots, x_n$ instead of using just the mean and standard deviation. We are also investigating efficient higher-order error propagation mechanisms for uncertainty analysis. Finally, anyone willing to contribute to this library is welcome to do so, either through the GitHub repository [26] or directly contacting the authors.

## CRediT authorship contribution statement

**Carlos Javier Fernández-Candel:** Investigation, Software, Validation, Writing – original draft, Writing – review & editing. **Paula Muñoz:** Investigation, Software, Writing – original draft, Writing – review & editing, Validation. **Javier Troya:** Funding acquisition, Investigation, Methodology, Software, Supervision, Validation, Writing – original draft, Writing – review & editing. **Antonio Vallecillo:** Funding acquisition, Investigation, Methodology, Software, Validation, Writing – original draft, Writing – review & editing, Supervision.

[2] Java library GitHub: https://github.com/atenearesearchgroup/uncertainty.

[3] COSCA project website: https://atenea.lcc.uma.es/projects/COSCA.html.

[4] IPSCA project website: https://atenea.lcc.uma.es/projects/IPSCA.html.

## Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

## Data availability

No data was used for the research described in the article.

## Acknowledgments

This work was partially supported by the Spanish Government (FEDER/Ministerio de Ciencia e Innovación – Agencia Estatal de Investigación) under projects SoCUS [TED2021-130523B-I00] and IPSCA [PID2021-125527NB-I00], and by the Junta de Andalucia, Spain, under contract QUAL21 010UMA.

## References

[1] Garlan D. Software engineering in an uncertain world. In: Proc. of the FoSER workshop at FSE/SDP 2010. ACM; 2010, p. 125–8. http://dx.doi.org/10.1145/1882362.1882389.

[2] JCGM 100:2008: Evaluation of measurement data – Guide to the expression of uncertainty in measurement (GUM). Joint Committee for Guides in Metrology. 2008, URL https://www.bipm.org/documents/20126/2071204/JCGM_100_2008_E.pdf.

[3] Esfahani N, Malek S. Uncertainty in self-adaptive software systems. In: Software engineering for self-adaptive systems II. LNCS, vol. 7475, Springer; 2010, p. 214–38. http://dx.doi.org/10.1007/978-3-642-35813-5_9.

[4] Cámara J, Troya J, Vallecillo A, Bencomo N, Calinescu R, Cheng BHC, et al. The uncertainty interaction problem in self-adaptive systems. Softw Syst Model 2022;21(4):1277–94. http://dx.doi.org/10.1007/s10270-022-01037-6.

[5] Moreno GA, Cámara J, Garlan D, Schmerl B. Proactive self-adaptation under uncertainty: A probabilistic model checking approach. In: Proceedings of the 2015 10th joint meeting on foundations of software engineering. ESEC/FSE 2015, New York, NY, USA: ACM; 2015, p. 1–12.

[6] Hezavehi SM, Weyns D, Avgeriou P, Calinescu R, Mirandola R, Perez-Palacin D. Uncertainty in self-adaptive systems: A research community perspective. ACM Trans Auton Adapt Syst 2021;15(4).

[7] Moreno N, Bertoa MF, Barquero G, Burgueño L, Troya J, García-López A, et al. Managing uncertain complex events in web of things applications. In: Proc. of ICWE'18. LNCS, vol. 10845, Springer; 2018, p. 349–57. http://dx.doi.org/10.1007/978-3-319-91662-0_28.

[8] Zhang M, Selic B, Ali S, Yue T, Okariz O, Norgren R. Understanding uncertainty in cyber-physical systems: A conceptual model. In: Proc. of ECMFA'16. LNCS, vol. 9764, Springer; 2016, p. 247–64. http://dx.doi.org/10.1007/978-3-319-42061-5_16.

[9] Vicino D, Wainer GA, Dalle O. Uncertainty on discrete-event system simulation. ACM Trans Model Comput Simul 2022;32(1):2:1–27. http://dx.doi.org/10.1145/3466169.

[10] Jézéquel J-M, Vallecillo A. Uncertainty-aware simulation of adaptive systems. ACM Trans Model Comput Simul 2023;33(3):8:1–8:19. http://dx.doi.org/10.1145/3589517.

[11] Navarrete FJ, Vallecillo A. Introducing subjective knowledge graphs. In: Proc. of EDOC'21. IEEE; 2021, p. 61–70. http://dx.doi.org/10.1109/EDOC52215.2021.00017.

[12] Moreno N, Bertoa MF, Burgueño L, Vallecillo A. Managing measurement and occurrence uncertainty in complex event processing systems. IEEE Access 2019;7:88026–48. http://dx.doi.org/10.1109/ACCESS.2019.2923953.

[13] Burgueño L, Muñoz P, Clarisó R, Cabot J, Gérard S, Vallecillo A. Dealing with belief uncertainty in domain models. ACM Trans Softw Eng Methodol 2023;32(2):31:1–34. http://dx.doi.org/10.1145/3542947.

[14] Bertoa MF, Burgueño L, Moreno N, Vallecillo A. Incorporating measurement uncertainty into OCL/UML primitive datatypes. Softw Syst Model 2020;19(5):1163–89. http://dx.doi.org/10.1007/s10270-019-00741-0.

[15] Muñoz P, Burgueño L, Ortiz V, Vallecillo A. Extending OCL with subjective logic. J Object Technol 2020;19(3):3:1–15. http://dx.doi.org/10.5381/jot.2020.19.3.a1.

[16] Famelis M, Rubin J, Czarnecki K, Salay R, Chechik M. Software product lines with design choices: Reasoning about variability and design uncertainty. In: Proc. of MODELS'17. IEEE Computer Society; 2017, p. 93–100. http://dx.doi.org/10.1109/MODELS.2017.3.

[17] Famelis M, Salay R, Chechik M. Partial models: Towards modeling and reasoning with uncertainty. In: Proc. of ICSE'12. IEEE Computer Society; 2012, p. 573–83. http://dx.doi.org/10.1109/ICSE.2012.6227159.

[18] Salay R, Chechik M, Horkoff J, Sandro AD. Managing requirements uncertainty with partial models. Requir Eng 2013;18(2):107–28. http://dx.doi.org/10.1007/s00766-013-0170-y.

[19] Eramo R, Pierantonio A, Rosa G. Managing uncertainty in bidirectional model transformations. In: Proc. of SLE'15. ACM; 2015, p. 49–58. http://dx.doi.org/10.1145/2814251.2814259.

[20] Burgueño L, Bertoa MF, Moreno N, Vallecillo A. Expressing confidence in models and in model transformation elements. In: Proc. of MODELS'18. ACM; 2018, p. 57–66. http://dx.doi.org/10.1145/3239372.3239394.

[21] Troya J, Moreno N, Bertoa MF, Vallecillo A. Uncertainty representation in software models: A survey. Softw Syst Model 2021;20(4):1183–213. http://dx.doi.org/10.1007/s10270-020-00842-1.

[22] Uncertainties package website. 2024, uncertainties-package, URL https://pythonhosted.org/uncertainties. [Accessed: January 2024].

[23] soerp package – PyPi. 2024, soerp, URL https://pypi.python.org/pypi/soerp. [Accessed: January 2024].

[24] Wikipedia. List of uncertainty propagation software. 2024, URL https://web.archive.org/web/20230326201711/https://en.wikipedia.org/wiki/List_of_uncertainty_propagation_software. [Accessed: January 2024].

[25] Deantoni J, Muñoz P, Gomes C, Verbrugge C, Mittal R, Heinrich R, et al. Quantifying and combining uncertainty for improving the behavior of digital twin systems. 2024. arXiv:2402.10535.

[26] Uncertainty-datatypes package - PyPi. 2024, utypes-pypi, URL https://pypi.org/project/uncertainty-datatypes. [Accessed: January 2024].

[27] Jøsang A. Subjective logic – a formalism for reasoning under uncertainty. In: Artificial intelligence: foundations, theory, and algorithms, Springer; 2016, http://dx.doi.org/10.1007/978-3-319-42337-1.

[28] UTypes – User guide. 2024, UTypes-userguide, URL https://github.com/atenearesearchgroup/uncertainty-datatypes-python/blob/master/docs/UserGuide.md/. [Accessed: January 2024].

[29] Python Enhancement Proposal PEP 335 — Overloadable Boolean Operators. 2024, pep-0335, URL https://peps.python.org/pep-0335. [Accessed: January 2024].

[30] Hoefler T, Belli R. Scientific benchmarking of parallel computing systems: Twelve ways to tell the masses when reporting performance results. In: Proc. of SC'15. ACM; 2015, p. 73:1–73:12. http://dx.doi.org/10.1145/2807591.2807644.

[31] Feng H, Gomes C, Thule C, Lausdahl K, Sandberg M, Larsen PG. The incubator case study for digital twin engineering. 2021, arXiv:2102.10390.

[32] Burgueño L, Mayerhofer T, Wimmer M, Vallecillo A. Using physical quantities in robot software models. In: Proc. of ROSE@MODELS'18. ACM; 2018, p. 23–8. http://dx.doi.org/10.1145/3196558.3196562.

[33] Burgueño L, Mayerhofer T, Wimmer M, Vallecillo A. Specifying quantities in software models. Inf Softw Technol 2019;113:82–97. http://dx.doi.org/10.1016/j.infsof.2019.05.006.

[34] Jongeling R, Vallecillo A. Uncertainty-aware consistency checking in industrial settings. In: Proc. of MODELS'23. IEEE; 2023, p. 73–83. http://dx.doi.org/10.1109/MODELS58315.2023.00026.

[35] Muñoz P, Pérez-Vereda A, Moreno N, Troya J, Vallecillo A. Incorporating trust into collaborative social computing applications. In: Proc. of EDOC'21. IEEE; 2021, p. 21–30. http://dx.doi.org/10.1109/EDOC52215.2021.00020.

[36] Gogolla M, Büttner F, Richters M. USE: A UML-Based specification environment for validating UML and OCL. Sci Comp Prog 2007;69:27–34.

[37] Ortiz V, Burgueño L, Vallecillo A, Gogolla M. Native support for UML and OCL primitive datatypes enriched with uncertainty in USE. In: Proc. of oCL@MODELS'19. CEUR proceedings, vol. 2513, 2019, p. 59–66.

[38] Navarrete F, Garrido A, Bobed C, Vallecillo A. Ontology-driven automated reasoning about property crimes. 2023, [Submitted].

[39] Gogolla M, Vallecillo A. (An Example for) formally modeling robot behavior with UML and OCL. In: Proc. of MORSE@STAF'17. LNCS, vol. 10748, Springer; 2017, p. 232–46. http://dx.doi.org/10.1007/978-3-319-74730-9_22.

[40] Moreno N, Pérez-Vereda A, Vallecillo A. Managing reputation in collaborative social computing applications. J Object Technol 2022;21(3):3:1–13. http://dx.doi.org/10.5381/jot.2022.21.3.a1.