J. Cámara, S. Hahner, D. Perez-Palacin, A. Vallecillo, M. Acosta, N. Bencomo, R. Calinescu, and S. Gerasimou, "Uncertainty Flow Diagrams: Towards a Systematic Representation of Uncertainty Propagation and Interaction in Adaptive Systems," in 19th International Conference on Software Engineering for Adaptive and Self-Managing Systems (SEAMS '24), IEEE/ACM, 2024, doi: 10.1145/3643915.3644084.

Uncertainty Flow Diagrams: Towards a Systematic Representation of Uncertainty Propagation and Interaction in Adaptive Systems

Javier Cámara
jcamara@uma.es
ITIS Software, Universidad de Málaga
Spain

Sebastian Hahner
sebastian.hahner@kit.edu
Karlsruhe Institute of Technology
Germany

Diego Perez-Palacin
diego.perez@lnu.se
Linnaeus University
Sweden

Antonio Vallecillo
av@uma.es
ITIS Software, Universidad de Málaga
Spain

Maribel Acosta
maribel.acosta@tum.de
Technical University of Munich
Germany

Nelly Bencomo
nelly.bencomo@durham.ac.uk
Durham University
United Kingdom

Radu Calinescu
radu.calinescu@york.ac.uk
University of York
United Kingdom

Simos Gerasimou
simos.gerasimou@york.ac.uk
University of York
United Kingdom

ABSTRACT

Sources of uncertainty in adaptive systems are rarely independent, and their interaction can affect the attainment of system goals in unpredictable ways. Despite ample work on “taming” uncertainty, the research community has devoted little attention to the systematic representation, analysis, and mitigation of uncertainty propagation and interaction (UPI) in adaptive systems. To address this oversight, we introduce *Uncertainty Flow Diagrams*, a notation that captures key UPI aspects. We demonstrate the use and benefits of our novel notation on Znn.com, an adaptive news site infrastructure.

KEYWORDS

Uncertainty propagation, Uncertainty interaction, Modeling notations, Flow Diagrams

ACM Reference Format:
Javier Cámara, Sebastian Hahner, Diego Perez-Palacin, Antonio Vallecillo, Maribel Acosta, Nelly Bencomo, Radu Calinescu, and Simos Gerasimou. 2024. *Uncertainty Flow Diagrams*: Towards a Systematic Representation of Uncertainty Propagation and Interaction in Adaptive Systems. In *19th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS '24), April 15–16, 2024, Lisbon, AA, Portugal.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3643915.3644084

1 INTRODUCTION

Sources of uncertainty in adaptive systems are rarely independent, and their propagation and interactions can affect the attainment of system goals in subtle and often unpredictable ways [10]. As such, the management of *uncertainty propagation and interactions* (UPI) must be considered as a first-class systems development problem, e.g., by representing UPI explicitly in the models underpinning the operation of control loops, and by considering them during the analysis and planning activities of these loops.

The few existing approaches tackling uncertainty propagation focus on *homogeneous* uncertainties, i.e., uncertainties that are similar in nature, admit the same representations and are amenable to similar reasoning mechanisms. Examples of such approaches range from methods for the propagation of uncertainty in measurement [22] and for the propagation of belief uncertainty based on probability theory [11], to those that use *possibilities* (in Fuzzy set theory [35, 46]), *plausibilities* or belief functions (in Dempster-Shafer's theory [40]) or subjective logic [23]. In software, the propagation of design uncertainty has also been treated using design space variability exploration techniques [6, 8, 16, 25, 41, 44].

In contrast, the propagation of *heterogeneous* uncertainties has received less attention, particularly in the area of adaptive systems. The reason for this is twofold. First, even managing individual (and homogeneous) uncertainties is very challenging, so the research community has allocated most effort to addressing this “simpler” problem so far. Second, this otherwise commendable effort has not produced systematic approaches for the rigorous, unified treatment (e.g., representation, analysis, mitigation) of common uncertainties and their interactions in the area of self-adaptation (e.g., measurement uncertainty versus uncertainty induced by model abstraction).

In this paper, we argue that our research area is mature enough for such systematic approaches to be developed [45], that they are essential for managing UPI impacts on key properties of adaptive systems, and that they should be included in new engineering processes that yield more robust and resilient, adaptive systems [10].

To devise such systematic approaches, we need to address challenges that relate to the representation of UPI, as well as their analysis and quantification. So far, we only have notations to represent different types of uncertainty in isolation [42], but not their interaction [9, 10]. Thus, representing the different types of uncertainty interactions that affect relevant system properties remains a major challenge that entails not only categorizing the different

SEAMS '24, April 15–16, 2024, Lisbon, AA, Portugal

ACM ISBN 979-8-4007-0585-4/24/04…$15.00
https://doi.org/10.1145/3643915.3644084

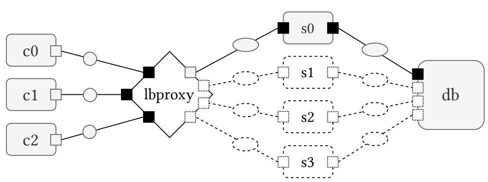

Figure 1: Znn.com system architecture [15]

classes of interactions that can be found in the context of an adaptive system, but also devising appropriate notations and patterns to represent them and enable their automated analysis and mitigation. A key requirement for these notations is that they should be able to capture how uncertainty propagates both *horizontally* (i.e., within the same level of abstraction) and *vertically* (i.e., across different levels of abstraction, for instance going from the managed to the managing subsystem of an adaptive system, and *vice versa*). To address this and other challenges in representing and reasoning about UPI there is a need for notations and analysis techniques able to handle heterogeneous sources of uncertainty, enabling engineers to trace them back to the properties on which they have an impact.

While different notations have been employed to trace uncertainty through software system models in specific contexts such as confidentiality in Data Flow Diagrams (DFD) [18, 19] and UML activity diagrams [14, 17], these are not enough to address the challenges posed by UPI. We posit that leveraging key elements of DFD and architectural descriptions can enhance our understanding of how uncertainty propagates horizontally and vertically in an adaptive system, and how (homogeneous or heterogeneous) uncertainty interactions influence the satisfaction of system goals. To that end, we define *Uncertainty Flow Diagrams (UFD)*, a notation that captures the impact of UPI on system properties. UFD specification can be informed by available system artifacts (e.g., models, analyses) that use architecture-centric descriptions to reason about adaptations, and are proposed as a stepping stone to enable the integration of various analysis techniques (e.g., based on assume-guarantee verification [26, 34]) to overcome UPI analysis challenges.

2 MOTIVATING SCENARIO

We illustrate our approach in the context of Znn.com [7], an adaptive news website infrastructure that features a three-tier architecture comprising a set of servers that provide contents from backend databases to clients via front-end presentation logic (Figure 1). A load balancer distributes requests across the pool of servers, the size of which can be adjusted according to service demand.

Znn.com follows MAPE-K [24] and its adaptive layer is implemented with Rainbow [15]. Extra-functional goals include cost minimization, performance, and security. For clarity, we will focus only on performance (i.e., maintaining a low response time). In Rainbow, goals are captured as utility functions whose accrued value has to be maximized during system execution.

When Znn.com receives a spike in workload, this is reflected in the experienced response time (r) measured through a probe embedded at the system level. Of course, the sensed value ($\hat{r}$) is not the same as the ground truth r because the sensor has a limited accuracy and yields values within a range $[\hat{r}_{min}, \hat{r}_{max}]$ that contains r. Once the measured value $\hat{r}$ is obtained, the monitoring stage in MAPE-K incorporates it into the architectural model of the system, which is updated at run time in the knowledge base. This process

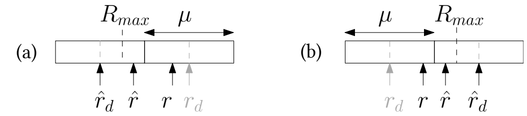

Figure 2: Model-Sensing interaction: (a) preventing required adaptation (b) causing spurious adaptation.

entails discretizing the value of the response time to accommodate the granularity in the model. We designate the discretized value of the measured response time by $\hat{r}_d$, which is defined as:

$$\hat{r}_d \equiv \arg \min_{x \in [\mathbb{R}]_\mu} (|\hat{r} - x|), \tag{1}$$

where $[\mathbb{R}]_\mu = \{x \in \mathbb{R} \mid x = i\mu,\ i \in \mathbb{Z},\ \alpha \leq x \leq \beta\}$ is the set of values that the discretized response time variable $\hat{r}_d$ can take, μ is a parameter that controls the granularity of the discretization (smaller μ means higher model fidelity), and $[\alpha, \beta]$ is the range of the variable. The discretized value of the measured response time $\hat{r}_d$ is then retrieved by the analysis stage in MAPE-K and compared against the maximum acceptable threshold for response time, R_{max}, which is stored as another property in the architectural model of the system in the knowledge base. Hence, if $\hat{r}_d > R_{max}$, the analysis stage triggers the planning stage, which will select an adequate adaptation strategy to fix the problem (e.g., activating additional servers, or reducing the quality of the contents served to clients).

The uncertainty induced by the measurement and discretization processes of the response time property can interact in more than one way with other uncertainties. In the situation illustrated in Figure 2(a), both r and the observed value $\hat{r}$ are above threshold R_{max}. Rectangles represent discretization buckets. When a real value is observed within a bucket, it is snapped to its center value (dashed lines). The discretization process snaps the value of $\hat{r}$ to $\hat{r}_d$, preventing the triggering of adaptation in a situation in which it would have been required. Note that without the error induced by the probe, the discretization process on its own would not have been enough to prevent the triggering of the adaptation, given that r would have been snapped to r_d, which is still above R_{max}.

Figure 2b illustrates the case in which r and the observed value $\hat{r}$ are below R_{max}. Here, a spurious adaptation is triggered because $\hat{r}_d$ is above R_{max}. Once again, it is only the combined effect of uncertainties in discretization and observation that cause this situation.

Making informed design decisions in the presence of such interactions is complex. Hence, we propose moving towards systematic approaches to handle UPI to obtain more resilient adaptive systems.

3 METHODOLOGICAL CONTEXT

Our approach (Figure 3) is intended to be used at design time by engineers building an adaptive system, who may use as input an already available set of (software) design and implementation artifacts (e.g., probes and effectors, analyzers, planners, specifications of adaptation tactics/strategies). The approach provides as output a report on the impact of uncertainty interactions on key system properties. This report can then be used to make informed decisions about design and implementation changes that may be required to mitigate the effect of UPI upon the satisfaction of the aforementioned properties. The process can be used iteratively to refine the design and implementation of the system incrementally.

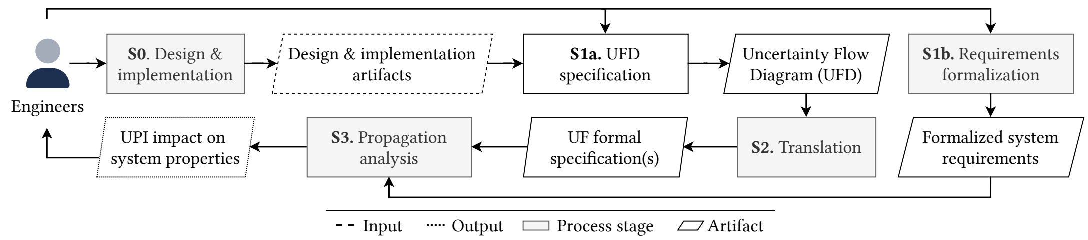

Figure 3: Overview of the envisaged solution showing the steps of design, specification, translation, and analysis.

The approach consists of the following steps: **S0.** *Design and implementation* precedes the rest of the process if a set of design/implementation artifacts is unavailable *a priori*; **S1a.** *UFD specification*, which involves the construction of UFD by engineers, informed by existing artifacts and stakeholder knowledge (e.g., members of the engineering team, domain experts). We describe their syntax in Section 4; **S1b.** *Formalization of system requirements*, that may include constraints (e.g., structural, behavioral, quality), system goals, and other relevant system properties; **S2.** *Translation* of UFD into formal specifications used as input to the tools to be employed in propagation the following step; and **S3.** *Propagation analysis* conducted with formal analysis tools that provide results informing about how uncertainties interact, affecting the set of system properties formalized in **S1b**.

4 UNCERTAINTY FLOW DIAGRAMS

A key component of our proposal is a notation to represent explicitly both (i) the uncertainty associated with each piece of information handled by the system, and (ii) how information and its associated uncertainty flow along computations. The main advantage of this approach is that it enables localizing and encapsulating the interactions between uncertainties within the relevant individual computations (`Actions`), providing a structured way of dealing with them. This approach is similar to the encapsulation of behavior proposed by the Object-Oriented paradigm. By isolating the individual computations into `Actions`, and by explicitly declaring the uncertainty that can affect their input and output parameters (represented by `Pins`), the behavior of each `Action` can be specialized to deal with the uncertainty interactions that may happen during the computation that the `Action` performs. Normally, each `Action` will encapsulate a small piece of behavior that involves variables with uncertainty, e.g., the computation of the value of a variable or the comparison between two or more uncertain variables.

The notation allows representing the control flow between `Actions`. `ControlFlows` are in charge of connecting `Pins` (possibly through `ControlNodes` that implement decisions, forks, merges and joins of the control flow), passing the information and its associated uncertainty from an output `Pin` to its connected input `Pin`. These are useful to check that the types of the connected `Pins` match, i.e., the output `Pin` is a subtype of the input `Pin`. They do not make calculations on the uncertainty of the information they handle; they only transmit the information and its associated uncertainty.

Figure 4 shows the UFD metamodel. Its core is a simplified version of Data Flow Diagrams (DFDs) [12] or UML 2.5 Activities [31], extended with information about uncertainty (based on the OMG's PSUM specification [33]). The top-level element is the `Activity`, i.e., a graph whose nodes and edges are `ActivityNodes` and `ControlFlows`, respectively. The graph represents how the information flows through the computations performed by a program. `Actions` represent behavior, are represented by squares with rounded corners, and have `Pins` (small white squares) that represent the types of input/output parameters. Each `Pin` has a `Type`. `Actions` may have an associated `Behavior`, such as the invocation of a method to transform the action's inputs into outputs.

To enable hierarchical modeling and vertical uncertainty propagation, each `Action` can also be refined by one or multiple `Activities` that represent its inner workings. At the highest abstraction level, the whole system can be represented by a single node with input and output pins, i.e., a black box. Either a `Behavior` or a set of refining internal `Activities` can be specified for an `Action`. It is also possible to specify multiple internal activities that represent alternatives. This enables the expression of structural uncertainty as variations, which is common to represent design uncertainty [42].

A `ControlFlow` is represented by a directed arrow that connects the outgoing `Pin` of an `Activity` with the incoming `Pin` of another `Activity`. `ControlFlows` may include `Guards` that have to be satisfied for the information to flow between the connected `Pins` (or `ControlNodes`). When more than one `Guard` is specified, they all need to be satisfied for the `ControlFlow` to take place (AND-semantics).

`ControlNodes` are used to define more elaborated flows between the `Pins`, including `Merges`, `Decisions`, `Forks` or `Joins` (with their usual UML, SPEM or BPMN semantics). `Initial` and `Final` nodes are also `ControlNodes` that represent the starting and final nodes of an `Activity`. `ExceptionNodes` allow dealing with exceptions, defining exceptional exits of actions, due to invalid situations that cannot be naturally handled by the action.

UFDs also enable the explicit representation of uncertainty, with the goal of dealing with the interactions between uncertainties that happen when making computations. UFDs allow the specification of individual uncertainties associated with `Pins` or with `Actions`. Each uncertainty can be of a different type (i.e., heterogeneous). This includes the common uncertainty types `Measurement Uncertainty`, `Discretization Uncertainty`, `Occurence Uncertainty`, `Design Uncertainty`, and also `Belief Uncertainty` [42].

Each uncertainty can have an associated `Measure`, which also has a `Type`. For example, one way to assess a `Measurement Uncertainty` is in terms of the accuracy of the measurement, which is normally expressed by means of a real number that represents the possible variation of the nominal value of the parameter, i.e., its estimated

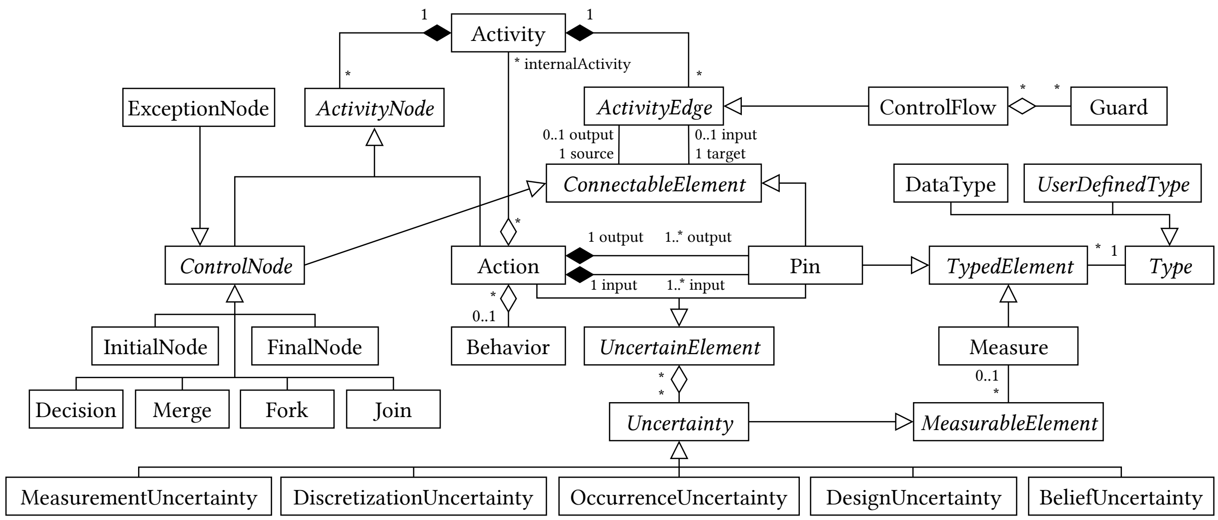

Figure 4: Uncertainty Flow Diagram (UFD) metamodel showing the relation of activities, flows, and different uncertainty types.

standard deviation. Likewise, `Belief Uncertainty` is normally expressed by a real number between 0 and 1 representing the likelihood (expressed as a probability) that the stated fact is true. Alternatively, uncertainty can be expressed by defining multiple `Internal Activities` as variation of the behavior of a single `Action` which is usually used to express `Design Uncertainty`. This enables also the expression of additional uncertainty types that are not included in our proposed metamodel. Any uncertainty type can be expressed if it can be denoted either quantitatively using an associated `Measure` or structurally using alternative internal `Activities`.

Different concrete syntaxes can be defined for this metamodel, such as Data Flow Diagrams (DFDs) [12], UML activity diagrams [31], SPEM [28], BMPN [30] models, or Petri Nets [21], with some small additions to represent the associated uncertainty. For example, the notation defined in UML for Activity Diagrams can be used to represent all the elements in the UFD metamodel except the uncertainty information, which can be represented using a UML profile. Figures that depict UFD in the next section employ this notation. The key issue is that the semantics of the UFD models can be seamlessly *embedded* into the semantics of other control-flow or process models. Note that some of these existing proposals provide ways to represent different types of uncertainty, but *none of them can represent multiple different types and their interactions*, while ours can.

5 DEMONSTRATION

In this section, we first demonstrate how to employ UFD with the problem described in Section 5.1, and then illustrate in Section 5.2 one of the possible embeddings of the scenario description in UML.

5.1 Representing UPI in ZNN.com

To demonstrate our approach, we tackle the problem that takes place when the uncertainty due to sensor readings is combined with the discretization in the variables to accommodate model granularity (cf. Section 2). A critical adaptation is activating a new server, which is triggered when the response time r goes above threshold R_{max}. This comparison can be defined by the operation (specified in OCL): `compare(r:Real,t:Real):Boolean = r > t`.

Using our graphical notation, this can be seen as an action whose inputs are two real numbers and returns a Boolean (Figure 5a). Then, if $r = 3.55$ and $R_{max} = 3.33$, then `compare(r,RMax) = true`.

The first type of uncertainty prevents knowing the exact value of r, due to the precision of the sensing devices and approximation errors caused by floating point calculations. This forces us to work with an approximated value $\hat{r}$. There is always a difference κ between $\hat{r}$ and r and, therefore, this uncertainty may produce some wrong decisions when $r > R_{max}$ but $\hat{r} < R_{max}$. We then need to incorporate this information into our computations. The Real values will be enriched with the accuracy of the sensors, and the Boolean values will be enriched with the degree of likelihood, i.e., they become probabilities (Figure 5b).

One way to deal with this type of uncertainty is by using an extended type system that enables the use of random variables to represent primitive data types [42]. Thus, if the accuracy of the sensor is u, then the comparison function can be written as follows:

```
uCompare(rh:UReal, t:UReal):UBoolean = rh > t
```

The result of this comparison function is of type `UBoolean` (a probability): the likelihood of the variable affected with uncertainty to be greater than the threshold t. For example, if $r = 3.55$, $R_{max} = 3.33$, $\hat{r} = 3.45$, the precision of the sensor is $u = 0.1$, and the threshold is a "crisp" value, with no uncertainty, then we obtain:

```
compare(r, RMax) = true
compare(r_hat, RMax) = true
uCompare(UReal(r,0.1), RMax) = UBoolean(true,0.986)
uCompare(UReal(r_hat,0.1), RMax) = UBoolean(true,0.885)
```

These are sensible results and provide more information than their "crisp" versions. In any case, this is part of the current state of the art because propagating one type of uncertainty is well described in the existing literature.

A second source of uncertainty happens when the value of the obtained response time $\hat{r}$ is discretized, e.g., rounded to the nearest integer. The final value, $\hat{r}_d$ is then compared to the threshold. There might be a problem when the discretized value is below the threshold, but the real value is not. This only happens when the value of the threshold is in the lower part of the discretization bucket.

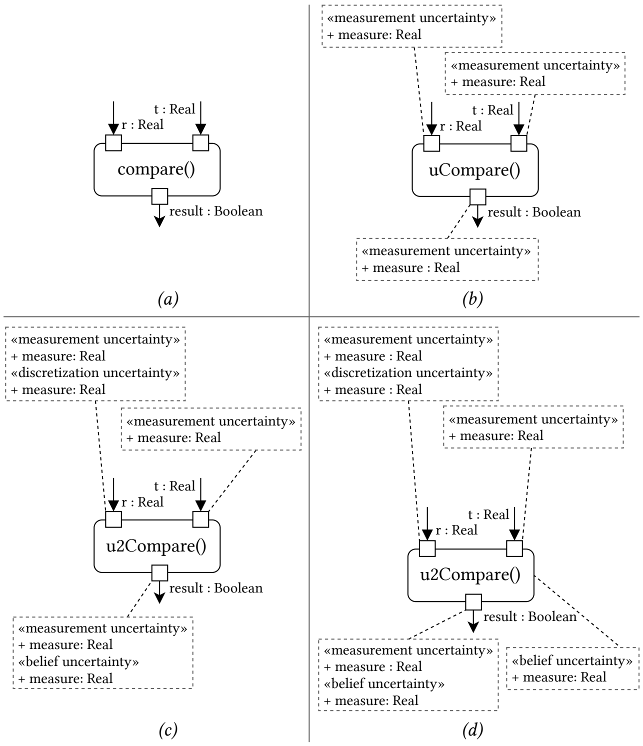

Figure 5: `Compare()` operation handling different uncertainty

For example, if $r = 3.4$ and $R_{max} = 3.3$. In this case, $r > R_{max}$ but $r_d < R_{max}$. This is easy to solve, by simply discretizing the threshold before comparing it with the discretized value.

However, as mentioned in Section 2, the problem occurs when the two uncertainties are combined, which may lead to wrong decisions. For instance, although the ground truth value r, its discretization r_d and the sensed value $\hat{r}$ lie above the threshold, the sensed and discretized value $\hat{r}_d$ does not.

To address this issue, we can define another operation, which is able to deal with both types of uncertainties (Figure 5c). Note how the discretization uncertainty is added to the affected parameter, and the result now comes not only with measurement uncertainty, but also with belief uncertainty [4, 42]. The reason is that the original Boolean result not only becomes a probability expressing the likelihood of the truth of the comparison, but it should also provide some degree of confidence in this likelihood. And just as measurement uncertainty is accompanied by the corresponding accuracy, the discretization uncertainty needs to be accompanied by the error made in the discretization ($de = |x - x_d|$).

The behavior of the new comparison operation, which considers both types of uncertainties, can be specified in OCL as follows:

```
u2Compare(rd:Integer,de:UReal,t:UReal):
      Tuple(res:UBoolean,confidence:UBoolean) =
  Tuple{res:uCompare(rd,t),confidence:(de>0.5)}
```

This operation provides not only the result of the comparison (in terms of the likelihood of the value being above the threshold) but also the confidence that we can have in such a comparison. For example, if again $r = 3.55$, $R_{max} = 3.33$, $\hat{r} = 3.45$, $u = 0.1$, then $\hat{r}_d = 3.0$ and hence we obtain:

```
compare(r_hat_d, RMax) = false
uCompare(r_hat_d, RMax) = UBoolean(true,0.0)
u2Compare(r_hat_d,(UReal(r_hat,0.1)-r_hat_d).abs(),RMax) =
  Tuple{res=UBoolean(true,0.0),
        confidence=UBoolean(true,0.309)}
```

The previous comparisons (in gray) fail when the two uncertainties are combined. The new operation, `u2Compare()` can combine both uncertainties and also returns false. However, it yields a confidence of only 0.309. The engineer can now use this information to disregard this value because its associated confidence is too low.

Furthermore, a reliable result for the comparison can be easily computed using the `equiv` operator to combine the likelihood of the comparison and its confidence: `result.res = result.confidence`. With this, we can define the operation in OCL that is able to determine whether $\hat{r}_d$ is in fact greater than the threshold R_{max}:

```
rightCompare(rd:Integer,de:UReal,t:Real):UBoolean =
  let x:Tuple(res:UBoolean,confidence:UBoolean) =
        u2Compare(rd, de, t) in x.res = x.confidence
```

The resulting value is now:

```
rightCompare(r_hat_d,(UReal(r_hat,0.1)-r_hat_d).abs(),RMax)
    = UBoolean(true,0.692)
```

Note how this value is correct and fixes the problem that happened when the two uncertainties were combined, and their interaction caused the comparison to produce a wrong result. Now we are able to deal correctly with both types of uncertainties. Finally, the activity itself can also be affected by uncertainty. For example, we may not be entirely sure that the algorithm used in the activity to accomplish its calculations is fully reliable. This is expressed by attaching some uncertainty to the activity node (Figure 5d).

In this way, engineers have a means to explicitly represent the different types of uncertainties that affect their calculations and how they propagate. More importantly, their interactions are now encapsulated within specific actions and can be treated in a particular way within these specific actions depending on their nature and particularities. The objective is to be able to more accurately quantify the uncertainty of the system in order to make decisions with higher confidence and less error.

Our notation also supports design uncertainty (e.g., system elements that may be realized by alternative, functionally equivalent components at run time). Figure 6 shows an example in which the sensor for the response time property in Znn can be realized by two alternatives, one of which includes a preprocessing step (e.g., a sliding window average to reduce oscillations in measurements). These alternatives would affect the measurement uncertainty in different ways, resulting in different uncertainty interactions.

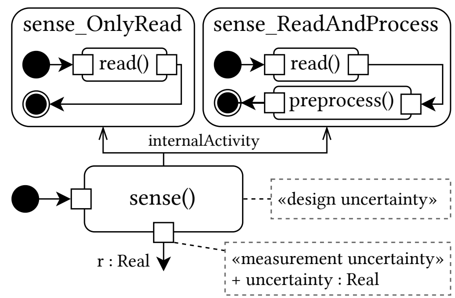

Figure 6: Structural uncertainty encoded as internal activities

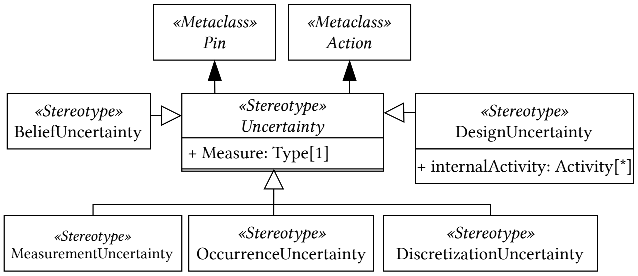

Figure 7: UFD profile, an UML profile for UFDs

5.2 Embedding Znn.com's UFD in UML

The representation of the motivating scenario UFD using UML Activity Diagrams requires an extension of UML that allows representing the uncertainty information; i.e., all the specializations of the *Uncertainty* class in Figure 4. We have implemented such an extension in the form of a UML profile, called UFD profile, which is depicted in Figure 7. This profile creates a stereotype for each class of uncertainty in the metamodel, provides the *Measure* attribute to each stereotype, provides the `Design Uncertainty` with means to refer to multiple internal activities, and allows applying the uncertainty stereotypes to *Pin* and *Action* UML metaclasses, which correspond to the classes that aggregate *Uncertainty* in Figure 4.

Figure 8 depicts the workflow for the adaptation decision in our scenario. It uses a UML Activity Diagram with the UFD profile in Figure 7. The diagram generalizes the previous Figures 5 and 6 since it represents how the uncertainties propagate through different actions. It comprises three actions: an action that reads the values from sensors ($\hat{r}$ value in Section 2) that is subject to design uncertainty and propagates measurement uncertainty, an action that stores the sensed values in the system model and generates the uncertainty due to the model granularity, and an action that compares the uncertain data in the model with the adaptation threshold value. The attribute values of «*DesignUncertainty*»are shown as a comment to the stereotyped Action. The attributes of the rest of the stereotypes are similar to the values in Figure 5 and are omitted.

6 RELATED WORK

Within the systems safety analysis domain, error propagation analysis has been traditionally employed during the early stages of systems engineering to understand how errors can propagate across the system by leveraging system architectural representations [1]. Despite their usefulness, these notations and analysis techniques are not enough to address the challenges posed by UPI.

Data Flow Diagrams (DFD) originate from structural analysis of software systems [12] and are used to analyze a variety of quality properties [27, 37, 39, 43]. Often, a precondition for such analyses is the extension of the DFD syntax, e.g., to express assets within the system [43], or the behavior of nodes [38], including specialized query and constraint languages [20, 27, 43]. Recently, DFD have been used to analyze uncertainty, e.g., in combination with fuzzy inference [5], or by using tracing [18] and propagation techniques [19]. However, despite the fact that these techniques can manage uncertainty propagation, they are often focused on a specific aspect of the system, such as confidentiality [19], and are not equipped to capture or analyze the interaction of uncertainties.

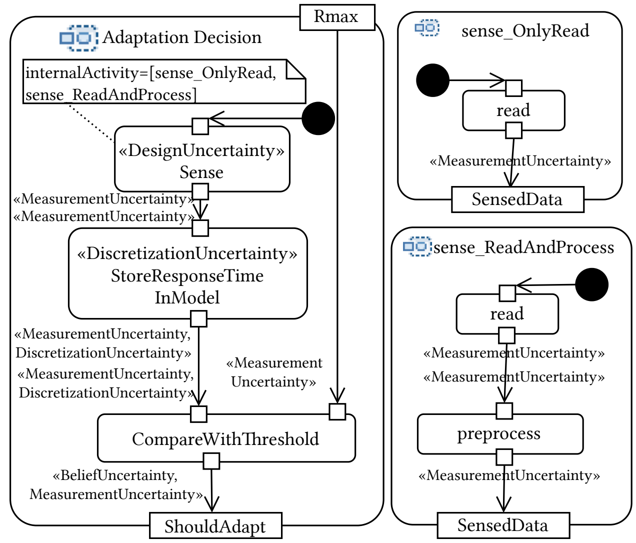

Figure 8: Example of profiled UML Activity Diagram.

Similarly, there are numerous notations for incorporating uncertainties of different types into software models [42]. Even SysML [32] and the UML MARTE Profile [29] provide some stereotypes and properties to represent some types of uncertainty, especially measurement uncertainty. However, existing notations allow modeling mostly homogeneous uncertainties, i.e., of the same type. We have seen how this is not sufficient to deal with uncertainty interactions. The new OMG initiative named PSUM (Precise Semantics for Uncertainty Modeling) [33] provides a metamodel for representing different types of uncertainty, but it says nothing about how to deal with their interactions. As far as we know, ours is the first notation aimed at capturing the propagation and interaction of different types of uncertainty through multiple levels of abstraction.

7 CONCLUSION

We have presented *Uncertainty Flow Diagrams (UFD)*, a notation to capture UPI, intended as part of the foundations required to build more resilient adaptive systems capable of analyzing and mitigating the combined effects of multiple sources of uncertainty. We posit that this vision will be enabled by leveraging model transformation techniques to automatically translate between UFD and various formalisms that can enable the analysis of UPI and will vary, depending on the types of uncertainties involved. For instance, Bayesian Networks [3] and Markov Decision Processes [13, 36] are promising candidates to analyze belief or conditional dependencies between system components, whereas Stochastic Petri Nets [2] are useful to analyse uncertainty in concurrent probabilistic real-time systems.

Next steps will move towards providing tool support, including facilities for building automated translation mechanisms of UFD specifications into other formalisms. Moreover, we will use such tool support to instantiate diverse UPI analysis mechanisms in different domains to provide a comprehensive evaluation of our approach and assess its generality.

ACKNOWLEDGMENTS

Work partially funded by the Spanish Government (AEI) under projects TED2021-130523B-I00 and PID2021-125527NB-I00, and supported by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) - SFB 1608 - 501798263. EPSRC (UK) Project Twenty20Insight(Grant No. EP/T017627/1) and also by funding from the topic Engineering Secure Systems of the Helmholtz Association (HGF) and by KASTEL Security Research Labs.

REFERENCES

[1] W. Abdelmoez, D.M. Nassar, M. Shereshevsky, N. Gradetsky, R. Gunnalan, H.H. Ammar, B. Yu, and A. Mili. Error propagation in software architectures. In *10th International Symposium on Software Metrics. Proceedings.*, page 384–393, 2004.

[2] Falko Bause and Pieter S Kritzinger. *Stochastic petri nets*, volume 1. Vieweg Wiesbaden, 2002.

[3] Nelly Bencomo and Amel Belaggoun. A world full of surprises: bayesian theory of surprise to quantify degrees of uncertainty. In *Companion Proceedings of the 36th International Conference on Software Engineering*, ICSE Companion 2014, page 460–463, New York, NY, USA, 2014. Association for Computing Machinery.

[4] Manuel F. Bertoa, Loli Burgueño, Nathalie Moreno, and Antonio Vallecillo. Incorporating measurement uncertainty into ocl/uml primitive datatypes. *Software and Systems Modeling*, 19(5):1163–1189, September 2020.

[5] Nicolas Boltz, Sebastian Hahner, Maximilian Walter, Stephan Seifermann, Robert Heinrich, Tomas Bures, and Petr Hnetynka. Handling environmental uncertainty in design time access control analysis. In *2022 48th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*. IEEE, 2022.

[6] Radu Calinescu, Milan Češka, Simos Gerasimou, Marta Kwiatkowska, and Nicola Paoletti. Efficient synthesis of robust models for stochastic systems. *Journal of Systems and Software*, 143:140–158, 2018.

[7] Shang-Wen Cheng, David Garlan, and Bradley R. Schmerl. Evaluating the effectiveness of the rainbow self-adaptive system. In *2009 ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems, SEAMS 2009, Vancouver, BC, Canada, May 18-19, 2009*, pages 132–141. IEEE Computer Society, 2009.

[8] Javier Cámara. Haiq: Synthesis of software design spaces with structural and probabilistic guarantees. In *Proceedings of the 8th International Conference on Formal Methods in Software Engineering*, FormaliSE '20, page 22–33, New York, NY, USA, 2020. Association for Computing Machinery.

[9] Javier Cámara, Radu Calinescu, Betty H. C. Cheng, David Garlan, Bradley Schmerl, Javier Troya, and Antonio Vallecillo. Addressing the uncertainty interaction problem in software-intensive systems: challenges and desiderata. In *Proceedings of the 25th International Conference on Model Driven Engineering Languages and Systems*, page 24–30, Montreal Quebec Canada, October 2022. ACM.

[10] Javier Cámara, Javier Troya, Antonio Vallecillo, Nelly Bencomo, Radu Calinescu, Betty H. C. Cheng, David Garlan, and Bradley Schmerl. The uncertainty interaction problem in self-adaptive systems. *Software and Systems Modeling*, 21(4):1277–1294, August 2022.

[11] Bruno de Finetti. *Theory of Probability: A critical introductory treatment.* John Wiley & Sons, 2017.

[12] Tom DeMarco. *Structure analysis and system specification*, pages 255–288. Springer, 1979.

[13] Luis Garcia-Paucar, Huma Samin, and Nelly Bencomo. Decision making for self-adaptation based on partially observable satisfaction of non-functional requirements. In *ACM Transactions on Autonomous and Adaptive Systems*, 2024.

[14] Luis Enrique García-Fernández and Mercedes Garijo. Modeling strategic decisions using activity diagrams to consider the contribution of dynamic planning in the profitability of projects under uncertainty. *IEEE Transactions on Engineering Management*, 57(3):463–476, August 2010.

[15] David Garlan, S-W Cheng, A-C Huang, Bradley Schmerl, and Peter Steenkiste. Rainbow: Architecture-based self-adaptation with reusable infrastructure. *Computer*, 37(10):46–54, 2004.

[16] Simos Gerasimou, Radu Calinescu, and Giordano Tamburrelli. Synthesis of probabilistic models for quality-of-service software engineering. *Automated Software Engineering*, 25:785–831, 2018.

[17] Carlo Ghezzi, Leandro Sales Pinto, Paola Spoletini, and Giordano Tamburrelli. Managing non-functional uncertainty via model-driven adaptivity. In *2013 35th International Conference on Software Engineering (ICSE)*, page 33–42, May 2013.

[18] Sebastian Hahner, Tizian Bitschi, Maximilian Walter, Tomáš Bureš, Petr Hnětynka, and Robert Heinrich. Model-based confidentiality analysis under uncertainty. In *2023 IEEE 20th International Conference on Software Architecture Companion (ICSA-C)*, page 256–263. IEEE, 2023.

[19] Sebastian Hahner, Robert Heinrich, and Ralf Reussner. Architecture-based uncertainty impact analysis to ensure confidentiality. In *2023 IEEE/ACM 18th Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*, page 126–132, May 2023.

[20] Sebastian Hahner, Stephan Seifermann, Robert Heinrich, Maximilian Walter, Tomáš Bureš, and Petr Hnětynka. Modeling data flow constraints for design-time confidentiality analyses. In *2021 IEEE 18th International Conference on Software Architecture Companion (ICSA-C)*, page 15–21. IEEE, March 2021.

[21] ISO/IEC 15909-1:2019. *Systems and software engineering – High-level Petri nets – Part 1: Concepts, definitions and graphical notation.* ISO/IEC, 2019.

[22] JCGM 100:2008. *Evaluation of measurement data—Guide to the expression of uncertainty in measurement (GUM).* ISO Joint Com. for Guides in Metrology, 2008. http://www.bipm.org/utils/common/documents/jcgm/JCGM_100_2008_E.pdf.

[23] Audun Jøsang. *Subjective Logic – A Formalism for Reasoning Under Uncertainty.* Artificial Intelligence: Foundations, Theory, and Algorithms. Springer, 2016.

[24] Jeffrey O Kephart and David M Chess. The vision of autonomic computing. *Computer*, 36(1):41–50, 2003.

[25] Anne Koziolek, Heiko Koziolek, and Ralf Reussner. Peropteryx: automated application of tactics in multi-objective software architecture optimization. In *Joint ACM SIGSOFT conference – QoSA and ACM SIGSOFT symposium – ISARCS on Quality of software architectures – QoSA and architecting critical systems – ISARCS*, QoSA-ISARCS '11, page 33–42. ACM, 2011.

[26] Marta Kwiatkowska, Gethin Norman, David Parker, and Hongyang Qu. Assume-guarantee verification for probabilistic systems. In *Tools and Algorithms for the Construction and Analysis of Systems*, Lecture Notes in Computer Science, page 23–37, Berlin, Heidelberg, 2010. Springer.

[27] Oege de Moor, Mathieu Verbaere, Elnar Hajiyev, Pavel Avgustinov, Torbjorn Ekman, Neil Ongkingco, Damien Sereni, and Julian Tibble. Keynote address: .ql for source code analysis. In *Seventh IEEE International Working Conference on Source Code Analysis and Manipulation (SCAM 2007)*, page 3–16, Sep 2007.

[28] Object Management Group. *Software & Systems Process Engineering Metamodel (SPEM) Specification. Version 2.0*, April 2008. OMG document formal/08-04-01.

[29] Object Management Group. *UML Profile for MARTE: Modeling and Analysis of Real-Time Embedded Systems. Version 1.1*, June 2011. OMG Document formal/2011-06-02.

[30] Object Management Group. *Business Process Model and Notation (BPMN) Specification. Version 2.0*, January 2014. https://www.bpmn.org/.

[31] Object Management Group. *Unified Modeling Language (UML) Specification. Version 2.5*, March 2015. OMG document formal/2015-03-01.

[32] Object Management Group. *OMG Systems Modeling Language (SysML), version 2.0*, June 2023. OMG Document ptc/23-06-02.

[33] Object Management Group. *Precise Semantics for Uncertainty Modeling (PSUM), Version 1.0 Beta 1*, March 2023. https://www.omg.org/spec/PSUM/1.0/Beta1/PDF.

[34] Esteban Pavese, Víctor Braberman, and Sebastian Uchitel. Probabilistic environments in the quantitative analysis of (non-probabilistic) behaviour models. In *Proceedings of the 7th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering*, ESEC/FSE '09, page 335–344, New York, NY, USA, August 2009. ACM.

[35] Stuart J. Russell and Peter Norvig. *Artificial Intelligence, A Modern Approach.* Prentice Hall, 3 edition, 2010.

[36] Huma Samin, Nelly Bencomo, and Peter Sawyer. Decision-making under uncertainty: be aware of your priorities. *Software and Systems Modeling*, 21(6):2213–2242, December 2022.

[37] Simon Schneider and Riccardo Scandariato. Automatic extraction of security-rich dataflow diagrams for microservice applications written in java. *Journal of Systems and Software*, 202:111722, Aug 2023.

[38] Stephan Seifermann, Robert Heinrich, Dominik Werle, and Ralf Reussner. A unified model to detect information flow and access control violations in software architectures. In *Proceedings of the 18th International Conference on Security and Cryptography*, page 26–37. Science and Technology Publications, 2021.

[39] Stephan Seifermann, Robert Heinrich, Dominik Werle, and Ralf Reussner. Detecting violations of access control and information flow policies in data flow diagrams. *Journal of Systems and Software*, 184, Feb 2022.

[40] Glenn Shafer. *A Mathematical Theory of Evidence.* Princeton U. Press, 1976.

[41] Marco Sinnema and Sybren Deelstra. Classifying variability modeling techniques. *Information and Software Technology*, 49(7):717–739, July 2007.

[42] Javier Troya, Nathalie Moreno, Manuel F. Bertoa, and Antonio Vallecillo. Uncertainty representation in software models: a survey. *Software and Systems Modeling*, 20(4):1183–1213, August 2021.

[43] Katja Tuma, Riccardo Scandariato, and Musard Balliu. Flaws in flows: Unveiling design flaws via information flow analysis. In *2019 IEEE International Conference on Software Architecture (ICSA)*, page 191–200. IEEE, Mar 2019.

[44] Ken Vanherpen, Joachim Denil, Paul De Meulenaere, and Hans Vangheluwe. Design-space exploration in MDE: an initial pattern catalogue. In *First Int. Workshop on Combining Modelling with Search- and Example-Based Approaches (CMSEBA)*, volume 1340 of *CEUR Workshop Proceedings*, pages 42–51, 2014.

[45] Danny Weyns, Radu Calinescu, Raffaela Mirandola, Kenji Tei, Maribel Acosta, Amel Bennaceur, Nicolas Boltz, Tomas Bures, Javier Camara, Ada Diaconescu, Gregor Engels, Simos Gerasimou, Ilias Gerostathopoulos, Sinem Getir Yaman, Vincenzo Grassi, Sebastian Hahner, Emmanuel Letier, Marin Litoiu, Lina Marsso, Angelika Musil, Juergen Musil, Genaina Nunes Rodrigues, Diego Perez-Palacin, Federico Quin, Patrizia Scandurra, Antonio Vallecillo, and Andrea Zisman. Towards a research agenda for understanding and managing uncertainty in self-adaptive systems. *ACM SIGSOFT Software Engineering Notes*, 48(4):20–36, 2023.

[46] Hans-Jürgen Zimmermann. *Fuzzy Set Theory – and Its Applications.* Springer Science+Business Media, 2001.