Uncertainty-aware Simulation of Adaptive Systems

JEAN-MARC JÉZÉQUEL, University of Rennes, CNRS, Inria, IRISA, France
ANTONIO VALLECILLO, ITIS Software, Universidad de Málaga, Spain

Adaptive systems manage and regulate the behavior of devices or other systems using control loops to automatically adjust the value of some measured variables to equal the value of a desired set-point. These systems normally interact with physical parts or operate in physical environments, where uncertainty is unavoidable. Traditional approaches to manage that uncertainty use either robust control algorithms that consider bounded variations of the uncertain variables and worst-case scenarios, or adaptive control methods that estimate the parameters and change the control laws accordingly. In this paper we propose to include the sources of uncertainty in the system models as first-class entities using random variables, in order to simulate adaptive and control systems more faithfully, including not only the use of random variables to represent and operate with uncertain values, but also to represent decisions based on their comparisons. Two exemplar systems are used to illustrate and validate our proposal.

CCS Concepts: • **Software and its engineering → Software design engineering**; **Model-driven software engineering**; • **Computing methodologies → Uncertainty quantification**.

Additional Key Words and Phrases: Model-based Software Engineering, Control systems, Self-adaptive systems, Uncertainty.

1 INTRODUCTION

An adaptive system is a system that changes its behavior in response to its environment or to changes in its interacting parts. In general, these systems are rather complex to design, prove correct and optimize, and therefore simulations are used to analyze not only their behavior but also their properties of interest. In this context, models are used to represent the relevant characteristics of the system under study, whereas the simulations represent the evolution of the model over time [43].

Simulations are commonly used in domains where physical artifacts are costly to build and deploy, such as manufacturing [26] or robotics [32]. A typical example is an automated assembly line, in which conveyor belts and gantries are used to transport semi-assembled parts from one workstation to another, and the parts are added in sequence until the final assembly is produced. These systems are thoroughly simulated before they are deployed to ensure correct behavior once they are built. However, when deployed, they most often require some fine-tuning. The problem is that their physical parts and elements are never perfect: they contain looseness and small inaccuracies that need to be adjusted for. These inaccuracies are not usually captured by the models, often resulting in parts falling off the trays or clamps not gripping the items when initially deployed, for example.

Reality is indeed different from the model and its simulation because, e.g., the values obtained from the sensors are actually imprecise, the physical contour of the parts is not exactly the same in all cases, or the moving times are not always precise. Since this uncertainty is usually not explicitly considered — despite being an essential

Authors' addresses: Jean-Marc Jézéquel, jezequel@irisa.fr, University of Rennes, CNRS, Inria, IRISA, France; Antonio Vallecillo, av@uma.es, ITIS Software, Universidad de Málaga, Spain.

1049-3301/2023/3-ART $15.00
https://doi.org/10.1145/3589517

aspect of any physical system [12, 19] — the decisions made by the control system to order movements or to make adaptations are based on imprecise information that can even lead to catastrophic failures.

The usual solution to deal with uncertainty in these situations, including not only manufacturing but also in all types of control systems [10], uses robust control, a conservative strategy that relies on estimating static upper bounds on the variations of the variables, and assumes worst-case scenarios. This approach is easy to implement, but it may be too conservative in many situations and therefore sub-optimal — e.g., wasting too many resources or arriving at non-optimal approximations. Adaptive control systems aim at addressing this problem by using dynamic variables (instead of upper bound constants) to control the system's behavior. Although this strategy results in simulations that are more faithful to reality, they are much more difficult to develop and prove correct because all uncertainty operations, as well as the propagation of uncertainty, need to be manually and explicitly programmed by the software engineer. At the simulation level, this is commonly achieved using some of the existing uncertainty propagation packages, such as [2, 22, 23] (see [41] for a comprehensive list), but they are still quite complex to use. More importantly, these packages enable the propagation of uncertainty through arithmetic operations on uncertain numbers, but their comparison is not usually implemented. Indeed, these packages only offer crude support for comparing uncertain values, and we shall see that this is an essential operation for operating with uncertainty in a more precise manner.

In this paper we propose to use random variables as first-class entities in programs and models to represent and operate with the system uncertain values, including their comparison. In this way, the controller is going to be able to manage the uncertainties and then the simulations are going to be much more faithful than they currently are.

Our concrete claims are that (1) we need to include the sources of uncertainty as first-class entities in the system models, and (2) they should be better handled by the type system, and not by the programmers. This will enable the natural representation and management of uncertain numbers in models, and the automatic propagation of uncertainty through operations, which are cumbersome and error-prone tasks when performed manually by the programmers. In addition, it will allow the explicit representation of the uncertainty that occurs when two uncertain numbers are compared.

The organization of the paper is as follows. After this introduction, Section 2 briefly describes the context and background of our work. Then, Section 3 presents our proposal, starting with a running example that serves to illustrate our approach. We then describe how to represent and manage some of the uncertainties that affect that system. Another example is used to illustrate further uncertainties in a more complex setting. After that, Section 4 discusses some of the advantages and possible limitations of our proposal. Finally, Section 5 relates our proposal to similar works, and Section 6 concludes with an outlook on future work.

Open research: All the software, artifacts and results described in the paper are publicly available from https://github.com/atenearesearchgroup/uncertainty-aware-adaptive-systems.

2 CONTEXT

2.1 Introduction to the Chasing Robot

In this Section we present background concepts using a toy example of the simulation of a Chasing Robot model where a robot must follow a moving target, staying as close as possible to the target but without ever going under a specified safety distance (*e.g.,* 1 m). To keep it simple, we only consider a target moving at a constant speed (*e.g.,* 2 m/s) in straight line.

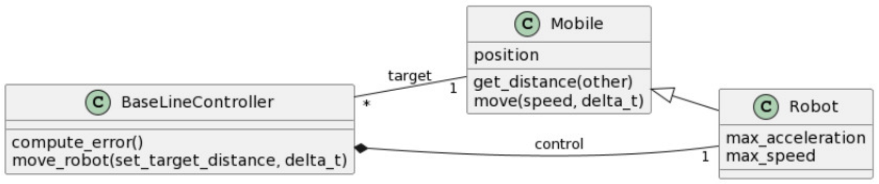

Fig. 1. Architecture of the Chasing Robot Example.

The chasing robot is controlled by a straightforward PID controller,[1] as illustrated in Figure 1 and detailed in the Python code below:

```python
class BaseLine:
  def __init__(self, robot: Robot, kp: float, ki: float, kd: float):
    self.robot = robot
    self.speed = 0.0
    self.max_acceleration = 5
    self.Kp = kp
    self.Ki = ki
    self.Kd = kd
    self.target = None
    self.error = [0.0, 0.0, 0.0]

  def get_error(self, target_distance: float):
    distance = self.robot.get_distance(self.target)
    return distance - target_distance

  def compute_target_speed(self, target_distance: float, dt: float) -> float:
    error = self.get_error(target_distance)
    self.error[2] = self.error[1]
    self.error[1] = self.error[0]
    self.error[0] = error
    derivative = (error - self.error[1]) / dt
    integral = (error+self.error[1]+self.error[2]) * dt
    return self.Kp * error + self.Ki * integral + self.Kd * derivative
```

Simulating this model consists in having a loop that:

(1) moves the target by calling its `move()` method with a given delta time (`dt`)
(2) asks the controller to move the robot it controls with the same delta time. This is implemented by method `move_robot()` below.

[1]A proportional–integral–derivative (PID) controller is a control loop mechanism that continuously calculates an error value as the difference between a desired setpoint and a measured process variable and applies a correction based on proportional, integral, and derivative terms (denoted P, I, and D respectively) [1].

```
def move_robot(self, target: Mobile, target_distance: float, dt: float):
  speed = self.compute_target_speed(target_distance, dt)
  speed = max(min(self.robot.max_speed, speed), 0)
  delta_speed = speed-self.speed
  if delta_speed != 0:
    sign = delta_speed/abs(delta_speed)
    if abs(delta_speed) < self.max_acceleration*dt:
      self.speed = speed
    else:
      self.speed += sign * self.max_acceleration*dt
    self.robot.move(self.speed, dt)
```

As expected, after an initial acceleration phase, the chasing robot speed converges towards following its target at the required distance. To assess the controller performance, we consider two indicators: (1) the minimum distance ever reached between the chasing robot and its target, and (2) the average distance over the entire course. The goal is, of course, to obtain the smallest possible average distance without ever going under the safety distance.

Our `BaseLine` Controller simulation[2] performs relatively well with respect to these indicators: starting 10 m behind the target, it ends up after 30s with an average distance of 2.0 m and a minimum one of 1.07 m. However, when tried in a real situation (*i.e.,* not a simulated one), the chasing robot would violate the safety distance, and even in some cases crash into the target. The reason is that reality is much more uncertain than our simple model.

2.2 Uncertainty sources

References [38, 44] summarize the main sources of uncertainty found in cyber physical systems. In this paper, we only focus on measurement uncertainty [3, 18]. In the chasing robot example, we can identify two sources of measurement uncertainty: (1) when calculating the distance to the target we rely on, *e.g.,* ultrasound sensors that yield imprecise data, and (2) when setting the speed of the robot, its actual speed might be a bit different due to inertia and friction.

Once again, for the sake of simplicity in this section we will only consider (1), *i.e.,* the distance uncertainty. In our simulation, we can introduce a distance sensor uncertainty by adding a random value X to the computation of the distance by a Mobile. X is chosen within a normal distribution, with an average value of 0 (no skew) and a standard deviation depending on the actual distance (due to the speed of sound) of $\sigma * (1 + 0.25 * real\_distance)$.

Now, if we run again our simulation several times with increasing values of σ, we can indeed see that as soon as $\sigma \geq 0.0325$ m the chasing robot can violate the safety distance.

2.3 Robust control

In control theory, robust control is an approach to make a controller work in the presence of uncertainty, assuming that certain variables will be unknown but bounded [1]. These robust methods aim to achieve robust performance and/or stability in the presence of bounded modeling errors. We can thus implement a variant of our `BaseLine` Controller that we call `RobustController`, that basically takes a fixed safety margin of $10 \times \sigma$ when computing the error to be minimized in the PID control:

```
class RobustController (BaseLine):
  def __init__(self, robot: Robot, kp: float, ki: float, kd: float):
    super().__init__(robot, kp, ki, kd)

  def get_error(self, target_distance: float):
    distance = self.robot.get_distance(self.target)
    return distance -target_distance -Mobile.sensor_accuracy*10 # margin with initial distance
```

That works well: for *e.g.,* $\sigma = 0.0325$ m the chasing robot always stays above the safety distance (at least 2.05 m). However its overall performance is not so good, with an average distance of only 2.55 m, which is too

[2]Each simulation is repeated 30 times to deal with pseudo-random number generation issues.

conservative. The goal of this paper is to investigate whether we can do any better by considering the distance returned by the sensors as an explicit random variable, *i.e.,* treat this kind of random variables as first class entities in our adaptive control programs.

3 UNCERTAINTY-AWARE CONTROL SYSTEMS

In this section we describe our proposal, and illustrate it on the chasing robot example discussed above. Our approach consists in three steps:

- Identify the possible sources of uncertainty
- Explicitly represent them so that they can be managed
- Incorporate them into the control loop, to improve the decision process of the control system in such a way that it can manage the identified uncertainties.

3.1 The Chasing robot example revisited

In our simple chasing robot example, we only consider the uncertainty due to the estimation of the distance to the target. Since we know that the distance sensor returns a value within a normal distribution with standard deviation of σ, we are going to explicitly model that return value with a random variable X having this σ standard deviation. Then, in the control loop of the robot, we can use X to make decisions (this approach is sometimes called adaptive control).

For most systems, including mission critical ones, reliability is not absolute but estimated in terms of the probability of not failing the mission. Depending on the stakes, this probability can range from 0.999 to 0.9999999999 or more, and is usually made readable by "counting the 9's". For instance, a probability of 0.999 is called 3 nines.

In the case of our chasing robot, this makes it possible to let the user choose the level of risk she wants to take, and use it as a parameter for controlling the robot. For example, for a 3 nines probability of keeping the safety distance, probability theory tells us that we need to take a margin of 3.29σ, while for 5 nines we need 4.41σ. We can easily model that in Python using the class `ufloat` from the `uncertainties` package that provide basic support for random variables:

```python
class ProbabilisticController (BaseLine):
  def __init__(self, robot: Robot, kp: float, ki: float, kd: float, risk: float):
    super().__init__(robot, kp, ki, kd)
    self.confidence = self.confidence_interval[risk]

  confidence_interval = {.90:1.64, .99:2.67, .999:3.29, .9999:3.89, .99999:4.41}

  def get_error(self, target_distance: float) -> ufloat:
    distance = self.robot.get_distance(self.target)
    return ufloat(distance, abs((1+0.25*distance)*Mobile.sensor_accuracy))


class PXNinesController (ProbabilisticController):
  def __init__(self, robot: Robot, kp: float, ki: float, kd: float, risk: float):
    super().__init__(robot, kp, ki, kd, risk)

  def get_error(self, target_distance: float) -> float:
    distance = super().get_error(target_distance)
    return distance.nominal_value - distance.std_dev*self.confidence -target_distance # p>risk
```

Figure 2 summarizes the performance of various versions of our chasing robot controller when σ increases. It is obtained by running each chasing robot for 30 seconds and plotting both its real minimum and average distance to the target. The experiment is repeated 30 times to account for random perturbations.

When the precision of the distance sensor degrades, the `BaseLine` controller fails, going below the safety distance (1 m) as explained above. In contrast, the `RobustController` is too cautious and we can see how its

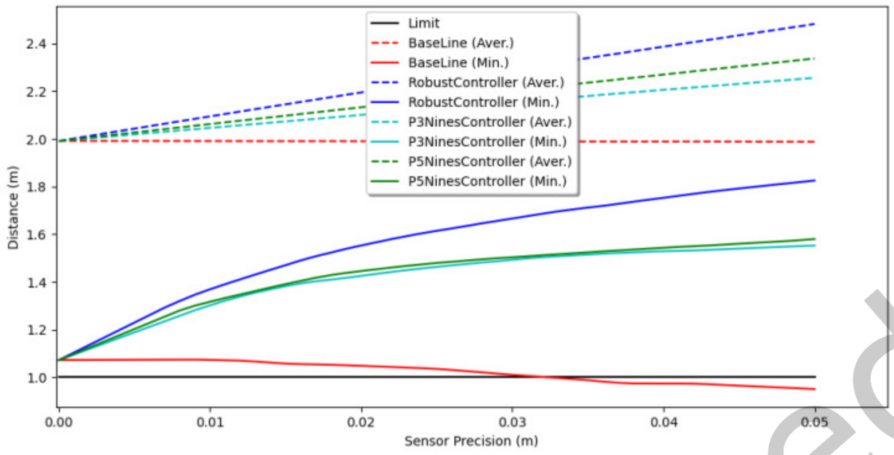

Fig. 2. Performance of various control algorithms for the chasing robot example.

performance degrades as the precision of the sensors decreases: it stays too far behind the target. However, our two versions of an uncertainty aware controller (`P3Nines` and `P5Nines`) are making a good trade off between quality of service (average distance) and safety (not going under 1 m).

Since this example is very simple, the basic support for random variables currently found in, *e.g.,* Python, is good enough to handle it. However, as soon as computations need to be done manually on random variables, things get much more complicated to handle at the programmatic level without strong support for treating random variables as first class entities, including support for comparison operators among them. Let's demonstrate that in the next section with another example, the ZNN system, which is a bit more complex than the chasing robot system.

3.2 The Znn.com system

Znn.com is a news service that is commonly used as an example of a self-adaptive system [35]. Its architecture is shown in Figure 3. The system comprises several *Servers*, some of which can be inactive, and a load balancer (*Dispatcher*) in charge of receiving *requests* from *Clients* and selecting the active server that will process them. The system monitors the dispatcher and the servers to make decisions in order to optimize its behavior.

One typical example is the invariant stating that the current system response time *R* for any request should always stay below some threshold *RMax*. When a request is received, the Dispatcher tries to find an active server able to process the request and respond to the originating client within the required time limit. If none is found, the dispatcher activates one of the inactive servers and sends the pending request to it. If all servers are active and none of them can ensure processing the request within the required timeframe, the request is denied and returned to the client. Server activation takes some time, the so-called starting latency. If a server is inactive for more than a certain time, it shuts itself down to save energy and waits for the dispatcher to activate it when required.

Our exemplar system comprises one dispatcher and four servers, all initially inactive. Four clients generate requests at a given pace. For simplicity, we assume that the processing time of all requests is the same, namely 20 time units.

To simulate this ZNN system we decided to use UML executable models, in order to show how our approach can be used with different paradigms, *i.e.,* it can work with both modeling and programming languages. In particular,

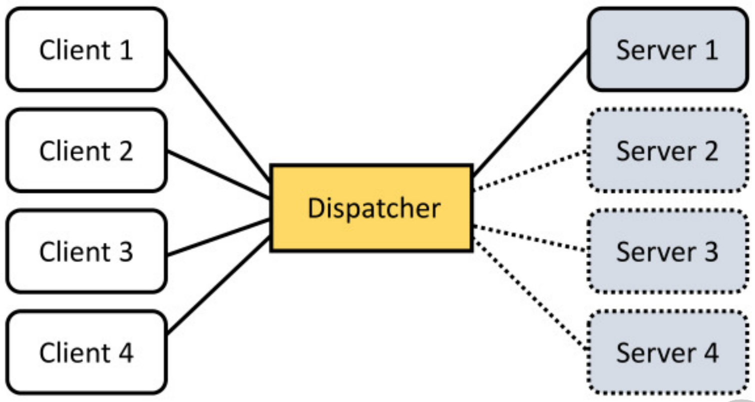

Fig. 3. The ZNN.com system architecture.

we have used UML and OCL [28] to specify the ZNN system, and the UML-based Specification Environment (USE) [14] to execute the UML system specifications. The use of such high level specifications provides interesting benefits, such as that we can abstract away from any concrete implementation, focusing on high-level models that allow run-time verification of system properties, and thus they require a very lightweight development process. Furthermore, these UML models could be formally analyzed using high-level validation tools [15], or transformed into concrete implementations if needed.

3.2.1 Baseline behavior. We simulated the system using different workloads, depending on the pace at which the clients issue their requests. In the *low workload* (LW) scenario, each client issues a request every 30 time units. Under *medium workload* (MW), requests are issued every 20 time units (*i.e.,* same as the processing time of requests). This simulates a stable system that works at optimal performance. In the *heavy workload* (HW) scenario, clients issue their requests every 18 time units, to ensure that servers cannot process all incoming requests. Assuming that the response time limit is 62 time units (*i.e., RMax* = 62) and that all clients issue their requests at the same time, the simulations show that no requests are overdue in any scenario, and that no requests are denied either under low and medium workloads. However, around 7.53% of the requests are denied under heavy workload, as theoretically expected: the throughput of the four clients is, respectively, 2.6666, 4.0 and 4.4444 requests per time unit in each scenario, while the combined processing capability of the servers is always 4.1333. This means that 7.5267% of the requests (=4.4444/4.1333 - 1.0) should be denied in the heavy workload scenario, as the simulations corroborate.

The three key decisions that the components of the system should make are: (1) whether a server can accept a request because it is able to respond to it in time; (2) whether a request is ready to be responded because its has been processed; and (3) whether a request is overdue.

These decisions can be implemented by the following query operations of a server (they are specified in OCL):

```
fits(r:Request):Boolean =
    r.finishTime - r.arrivalTime + self.swapTime < self.config.RMax

hasFinished(r:Request,now:Real):Boolean =
    r.finishTime <= now

isOverdue(r:Request):Boolean =
    (self.actualFinishTime - self.arrivalTime) > self.config.RMax
```

Table 1. Baseline system: % of denied and overdue requests under medium (20) and heavy (18) workloads.

Proc.Time Uncert.	Denied-20	Overdue-20	Denied-18	Overdue-18
0.0	0.00%	0.00%	7.81%	2.52%
0.1	0.00%	0.00%	7.81%	2.53%
0.2	0.00%	0.00%	7.27%	2.66%
0.3	0.00%	0.00%	7.54%	2.76%
0.4	0.00%	0.00%	7.27%	3.01%
0.5	0.00%	0.03%	7.81%	2.27%
0.6	0.00%	0.03%	7.81%	3.32%
0.7	0.00%	0.03%	7.81%	3.33%
0.8	0.00%	0.05%	7.54%	3.48%
0.9	0.00%	0.13%	7.27%	3.76%
1.0	0.00%	0.16%	7.54%	4.52%

In the last operation, `isOverdue()`, the value of attributes `arrivalTime` and `actualFinishTime` are set by the server when the request is accepted and when it is removed from its queue of pending requests, respectively.

3.2.2 A more realistic behavior. As mentioned in the introduction, reality is different from simulations, especially in the case of physical systems that are subject to different types of uncertainties. In this paper we assume that data collected from the environment can never be directly observed without noise, and also that an accurate model of the environment cannot be obtained [7, 27]. These are inherent characteristics of any physical system, and therefore cannot be neglected.

Let us consider here two sources of uncertainty that may affect our system:

- The actual time taken by a server to process a request is not a fixed value, but a random variable. This may cause the actual processing time to be greater than the expected one. Thus, a server may accept a request because, according to its calculations, it is able to respond to it within the required time, but then the actual processing time is longer than expected and the response is delayed.
- Clocks have some imprecision. Even if we assume that the deviations are only of micro-time units (*i.e.,* 10^{-6}), this can cause some comparisons between time variables to fail. We will see how this can cause some requests to be delayed, because when the system checks if a request has finished, the comparison fails and the request has to wait for the next clock cycle to be answered, hence causing unnecessary delays.

These two aspects correspond to measurement uncertainties [18], which affect the values of the variables managed by the simulation. To evaluate the effect of such uncertainties, we developed a simulation system (hereinafter, the `Baseline` system) where the values of the variables could have small variations, due to the lack of precision of the sensors (*e.g.,* the clock readings) or indeterminacy of the environment (*e.g.,* variations in the processing times of the requests due to other concurrent executing tasks running in the server).

To illustrate the effects of such uncertainties, Table 1 shows the percentage of denied and overdue requests under medium and heavy workloads, for different levels of processing time imprecision. The first column displays the value of the processing time uncertainty, which ranges between 0 and 5% of the processing time of each request, i.e., between 0 and 1 time units. This is used by the system to assign each request a deviation from its expected processing time, which simulates a more realistic situation where the actual processing times of requests are not perfect values but random variables. Of course, the heavier the workload the worst the results.

Figure 4 shows these results in a graphical way. Note how the percentages of denied requests maintains around the “expected” theoretical value of 7.53% value. Sometimes it is even lower because the requests are accepted based on their estimated processing times of 20 time units, and not on their actual processing times. In some

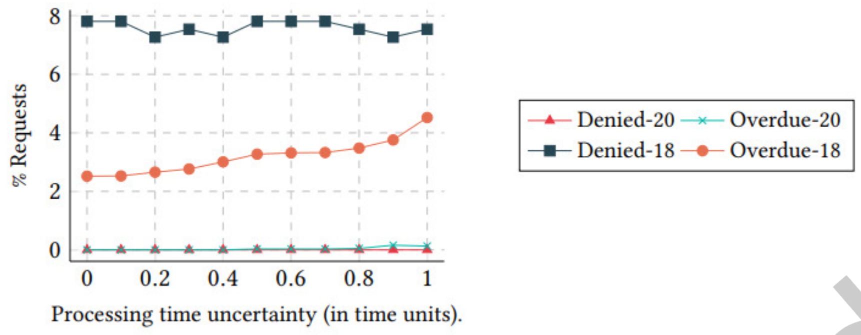

Fig. 4. Baseline system: % of denied and overdue requests under medium (20) and heavy (18) workloads.

cases, these decisions led to overdue responses. This is similar to the situation described in the introduction, where the behavior of the machines and parts on the assembly line did not correspond to what was expected from the simulations, and some parts fell off the trays, or the gantry grippers failed to grasp the parts.

3.2.3 Taming uncertainty: robust approach. As previously mentioned, robust control methods aim to achieve robust performance and/or stability in the presence of bounded uncertainty [1]. These methods normally use interval arithmetic [17, 40]. Thus, instead of representing a value as a single number x, interval arithmetic represents each value as a range of possibilities defined by the interval $[x_m, x_M]$ that contains x. The most common use is in software, to keep track of rounding errors in calculations and of uncertainties in the knowledge of the exact values of physical and technical parameters, so that reliable results can be guaranteed.

In the ZNN example, we implemented a robust solution using an interval of $\pm 5\sigma$ to ensure more than six nines precision in the processing time (what we have called 1.0 confidence in Table 2). This can be implemented by simply changing method `fits()` to include such a safety interval (note that in our case we use $\sigma = 1$, and therefore $5\sigma = 5$):

```
fits(r:Request):Boolean =
  r.est_finishTime - r.arrivalTime + self.swapTime + 5.0  -- safety interval added
  < self.config.RMax
```

However, it still does not work! We still get 0.08% overdue requests (see first row of Table 2, under column CI:PTU). Analyzing the causes, we realized that this is because we need to consider the second source of uncertainty, *i.e.*, the imprecision of the clock. As mentioned above, a slight variation of the clock readings may cause that we miss one time step. For example, the actual finishing time of a request is 30.0, but the clock time is 29.9999999. Then, the comparison `r.finishTime <= now` returns false and the request has to wait for the next time step.

To tackle this issue using a robust control approach, we substitute the uncertain variable (in this case, the clock readings) by an interval, and use the interval in the comparison. With this, query `hasFinished()` is implemented as follows:

```
hasFinished(r:Request,now:Real):Boolean =
  (now - r.finishTime).abs() <= 1.0)
```

This change has the desired effect, and no overdue requests are produced. However, the approach taken by robust control systems is too coarse-grained and conservative, normally wasting too many resources or producing

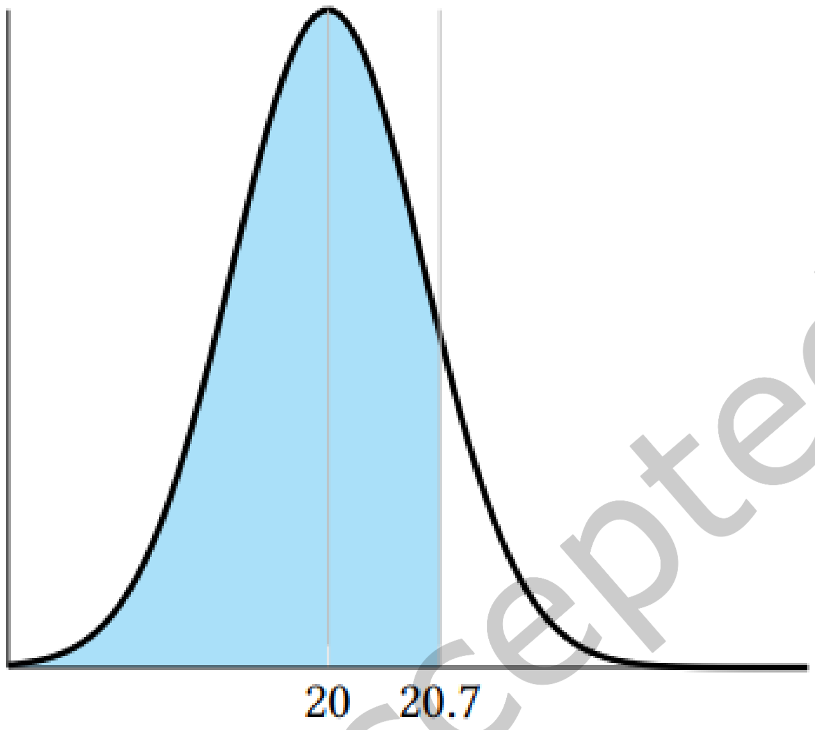

Fig. 5. Comparing one random variable and one Real number.

sub-optimal results, as illustrated in the Chasing Robot example. This is where adaptive control systems come into play.

3.2.4 Taming uncertainty: adaptive control. Adaptive control methods do not need a priori information about the bounds on the uncertain or time-varying parameters. In contrast, their safety bounds can dynamically adapt to improve the decisions made by the control loop. Random variables are commonly used instead of fixed-length intervals, and fixed-bound intervals become confidence intervals (CI). For example, assuming that the actual processing time of a request follows a normal distribution $X \sim N(x, \sigma)$ with standard deviation σ, we will use such a random variable X instead of the Real value x [18]. Comparison are no longer Boolean values, but become probabilities [3].

To illustrate the difference between the crisp, robust and adaptive approaches, consider the real values $x = 20.0$ and $y = 20.7$. Using Real arithmetic, $x < y =$ true. Assuming a precision of $\sigma = 0.6$ in the values of x, using a robust control approach we would define an interval of $\pm 5\sigma$ around x, *i.e.,* $\widehat{X} = [17, 23]$. In this case, given that $20.7 \in \widehat{X}$, then $x < y =$ false.

Finally, using the adaptive control strategy, variable x would be modeled by a random variable $X \sim N(20, 0.6)$ and the comparison $x < y$ becomes $P(X < 20.7) = 0.878$. Such probability coincides with the area shaded in blue in Figure 5.

To realize this adaptive control approach we only need to change the implementation of the two query operations that compare the two random variables of our example, namely `finishTime` and `now`:

```
fits(r:Request):Boolean =
  r.finishTime - r.arrivalTime + self.swapTime
  + self.config._processingTimeUnc * self.tolerance(config.robustness) < self.config.RMax

hasFinished(r:Request,now:Real):Boolean =
  (now-r.finishTime).abs() <= self.config._clockUnc * self.tolerance(config.robustness))
```

In these specifications, variables `_processingTimeUnc` and `_clockUnc` correspond to the precision of the requests processing times and the clock (1.0 and 10^{-6}, respectively). They are both stored as attributes of class `Config`. Operation `tolerance()` returns the number of σ's required to obtain a given robustness, *i.e.,* confidence. For example, assuming that the variables follow Normal distributions, to obtain a confidence of 3 nines (0.999) we need 3.29σ.

The results obtained for different levels of confidence using an adaptive control strategy are shown in columns CI:PTU and CI:PTU+CU of Table 2, and, graphically, in Figure 6. Column CI:PTU shows the percentage of overdue requests taking into account only the request performance time uncertainty (PTU). In turn, Column CI:PTU+CU shows the percentage of overdue requests taking into account both the request performance time uncertainty and the clock uncertainty (CU).

Table 2. Percentage of overdue requests depending on the confidence level.

Confidence	**CI:PTU**	**CI:PTU+CU**	**Stochastic**
Robust (1.0)	0.08%	0.00%	0.00%
0.9999	0.20%	0.11%	0.00%
0.999	0.72%	0.12%	0.00%
0.99	0.84%	0.15%	0.00%
0.98	0.85%	0.16%	0.00%
0.95	0.86%	0.22%	0.02%

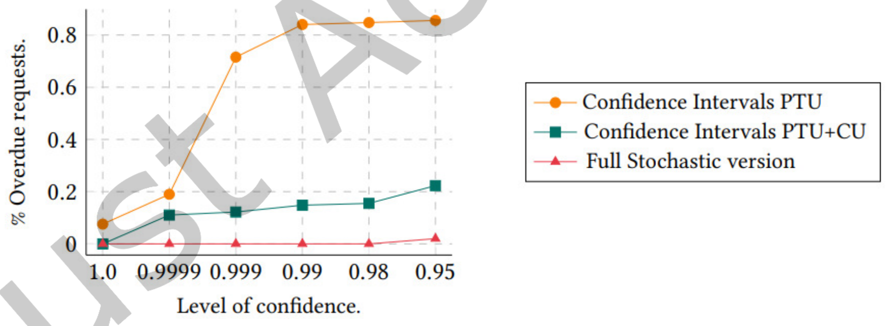

Fig. 6. Percentage of overdue requests depending on the confidence level.

3.2.5 Taming uncertainty with random variables. The adaptive strategy that uses random variables and confidence intervals to model some of the values of the system, but still uses *crisp* values with the rest. Our claim is that this is not realistic, because in physical systems there are no exact values, they *all* are subject to uncertainty, numerical approximations, or both. This is why physical variables should never be modeled by means of Real numbers, but using *uncertain* numbers.

For example, assuming that $X \sim N(20, 0.6)$ and y becomes a random variable $Y \sim N(20.7, 0.5)$, we get that $P(X < Y) = 0.48$. This is graphically depicted in Figure 7, where the shaded area shows the value of the comparison (check it against the area in Figure 5). By changing the standard deviation of the variables we obtain

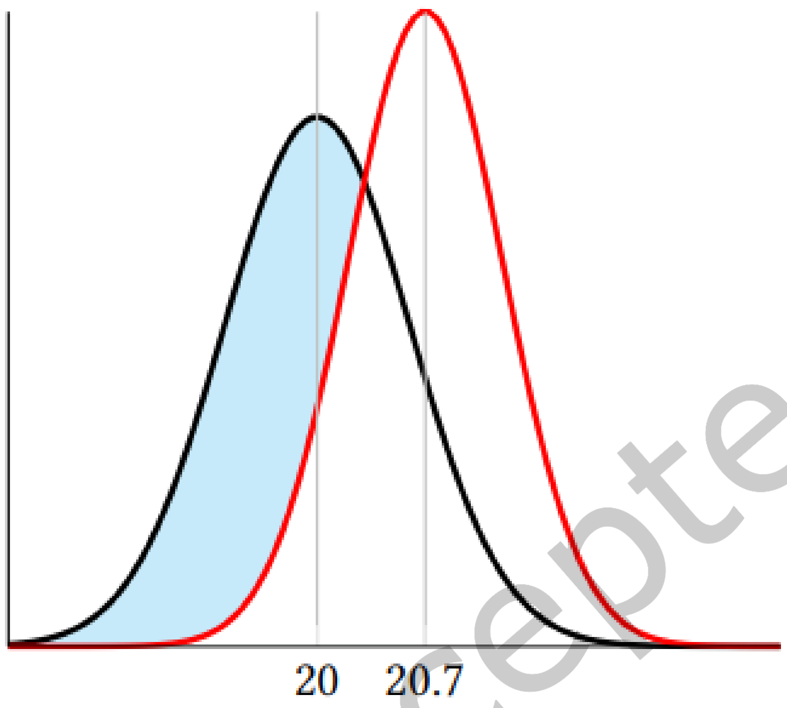

Fig. 7. Comparing two random variables.

different values for that probability. The larger the variance, the more difficult it is to tell the two values apart, and vice versa.

For implementing this approach we have used the Java library of datatypes extended with measurement uncertainty defined in [3], which is also implemented in the tool USE to support uncertain numbers in UML and OCL [29]. Essentially, this library extends the basic UML and OCL primitive datatypes (Real, Boolean, Integer,...) with uncertainty by defining super-types for them, as well as the set of operations defined on the values of these types. Thus, `Real` values with uncertainty are represented in terms of `UReal` values, which are composed of pairs (x, u), also noted as $x \pm u$, where x is the value, and u represents its uncertainty as the standard deviation of its possible variations, according to the GUM international standard [18]. Likewise, a `Boolean` value b is lifted to an `UBoolean` value B, which is a pair $B = (b, c)$ in which c is a real number between 0 and 1 that represents the confidence we assign to b. Comparison operators between `UReal` variables return `UBoolean` values. For example, if $a = 2.0 \pm 0.3$ and $b = 2.5 \pm 0.25$, then $a < b$ = `Boolean(true,0.893)`, meaning that $a < b$ with a confidence of 0.893 [3]. Projection operation `confidence()` applied to an uncertain Boolean returns a probability, *i.e.,* the confidence assigned to that Boolean. Uncertain values then become first-class entities of our models, and can be managed and operated in a natural way by the underlying type system. Propagation of uncertainty through operations is transparently taken care of by the type system, and comparisons are lifted to `UBoolean` values when required. This greatly simplifies the management of uncertain numbers in both Java programs and UML/OCL models.

Using these extended datatypes and operations, the critical queries in the ZNN system can be restated as follows:

```
fits(r:Request):Boolean =
    (r.finishTime - r.arrivalTime + self.swapTime < self.config.RMax).confidence()
      >= self.config.robustness

hasFinished(r:Request,now:UReal):Boolean =
    (now >= r.finishTime).confidence() >= self.config.robustness

isOverdue(r:Request):Boolean =
    (r.responseTime >  self.config.RMax).confidence() >= self.config.robustness
```

We can see how we can now play with the level of confidence (robustness) required, thus being able to quantify in a more precise way the degree of uncertainty with which we make decisions.

Last column (Stochastic) of Table 2 shows the percentage of overdue requests of a system that is simulated using the strategy of representing physical attributes with random variables, *i.e.,* uncertain reals. This is also shown graphically in Figure 6. The last 0.02% is to be expected, because we are assuming a confidence of only 0.95 in our decisions. As in the example of the chasing robot, with this strategy we can obtain more faithful simulations and therefore more accurate results.

4 DISCUSSION

So far we have shown how we are able to capture the inherent uncertainty of the possible values of the attributes used in a control system by means of random variables, and the benefits of handing them as first-class entities of our programs or models with the appropriate libraries. This section provides some methodological guidance on how our proposal can be used. Then, we discuss some further advantages and possible limitations of our proposal, and finish with some open questions.

Note that the two examples we have used to illustrate our approach in this paper come from the realm of physical systems, although our proposal is also applicable to scenarios where non-physical systems are considered. Ultimately, what we propose is a more effective modeling approach for any application where uncertainty plays a role, by considering uncertainty as a first-class entity. For example, our proposal can be used in scenarios where we are uncertain of the values of some parameters because the system is virtual, and decisions made by some underlying real system may materialize different values over time. Likewise, it is applicable in situations where we have to model the duration of tasks in software development environments, or where numerical errors in computations may lead to inaccurate results.

4.1 Methodological guidelines

If one wants to leverage the use of uncertain variables in a consistent way across an application, the following 3 steps might be followed.

- First, identify all possible *primary* sources of uncertainty in a model or a program. In cyber-physical systems, that is all the variables that store values which are read from system sensors. In other systems, it might be variables whose values depend on the hardware platform on which the program would execute. This is for example what may occur for virtual machines vs. hypervisors, *e.g.,* in terms of the completion time of a task.
- Then, for each uncertain variable, find a law that models its probability distribution (*e.g.,* normal, uniform, etc.). Alternatively, if the law is unknown, one could resort to measurements and store the measured distribution as a random variable (*i.e.,* Type A evaluation of uncertainty [18]).
- Finally, propagate uncertainty across the source code. That is, each time an uncertain variable is used in a computation, the result of the computation also becomes an uncertain variable. For instance, if x and y are `UReal` variables, the result b of their comparison, $b = x \leq y$, must be an uncertain variable, namely a

`UBoolean`. Some easy-to-implement static analysis can help ensure that this rule is enforced across the source code of an application.

This approach relies on the use of libraries supporting operations across uncertain basic data types (e.g., Ubooleans, UIntegers, UReals). Several such libraries already exist, for Python (with some limitations with respect to comparing uncertain variables), for Java (for instance the one we have developed in a previous work and which is freely available in our Github repository https://github.com/atenearesearchgroup/uncertainty), and for UML/OCL, this later one actually relying on the Java one.

4.2 Advantages

First, our proposal allows capturing the uncertainty of the data collected from the environment of the system in an accurate way. This uncertainty basically depends on the precision of the sources of these data, *i.e.,* on the possible variations of their values. Such precision values become input parameters for the control algorithms.

Using these input parameters we have shown how explicit uncertainty management allows us to choose the level of "risk" we want to take between the too naive (*i.e.,* crisp) vs. the too conservative (*i.e.,* robust) approaches. Thus, such a level of risk (*e.g.,* the acceptable failure rate, the admissible deviations from a theoretical ground truth, or the allowable degree of uncertainty in the value of an attribute) becomes a parameter we can play with — something essential for, *e.g.,* software certification.

In this way, we can make trade-offs to achieve acceptable compromises depending on the precision of the sensors, which is not the case now, as current control algorithms tend to use an all-or-nothing strategy. For the 'all' case, they decide the level of risk they want to take (maybe none in case of critical systems) and then build the control system based on this level. However, in our proposal the level of risk is a parameter of the controller, and we can decide (even at runtime) the trade-off we want to make and thus the level of risk acceptable for our system.

Working on the opposite direction, based on a given level of risk (or of robustness) and on the expected behavior of the system, we can decide about the required precision of the sensors that we need to install in our system to ensure that level of risk. This is very useful for systems where the costs of their parts (*e.g.,* the sensors) are important and should be maintained under control, but still ensure the required level of precision. Note that the use of random variables provides more accurate estimations than those provided by current control algorithms.

4.3 Potential limitations (and how to mitigate them)

Normality of the distributions. In the first place, our proposal makes some assumptions that might not hold in all situations. For example, we suppose that the random variables that represent the uncertainty of the attributes of our control algorithms follow Normal distributions (as usually done in measurement [18]). Should this not be the case, one solution would be to use Chebyshev's inequality to determine the range of standard deviations around the mean, and thus decide the level of robustness we are accepting. Note that the Chebyshev's inequality works for any type of distribution. Its practical usage is similar to the 68–95–99.7 rule, which applies only to normal distributions. Chebyshev's inequality is more general, stating that a minimum of just 75% of values must lie within two standard deviations of the mean and 88.89% within three standard deviations. Although this is a more conservative estimation that that used for the Normal distribution, it is still very useful to ensure acceptable levels of risk.

Variable independence. Secondly, in this work we have made some assumptions regarding the independence of the attributes when operating with their associated uncertainty using the closed-form solution. If such an independence cannot be ensured, there are several ways to deal with dependent (*i.e.,* correlated) variables. First, if we know their covariances, most specifications and implementations support closed-form expressions of the operations with uncertainty when variables are dependent. However, the values of such covariances are rarely known by users, and therefore they are not very useful in common practice. An alternative solution consists of using the implementation of the operations based on samples (*i.e.,* Type A evaluation of uncertainty [18]). This is also the approach proposed by ISO, which is very general and powerful. However, it may have a significant impact on the performance of the evaluation of the operations, given that they have to be applied to the samples, hence introducing an overhead proportional to the sample size.

Precision estimation. Sometimes, estimating the precision of data collected from the environment is not an easy task. Some common factors that make this task difficult include: the lack of information about the data sources and their uncertainty; the effect of unreliable communication channels and networks, which can produce large distortions in the values of the input data; or the degradation of data sources or communication channels themselves, which can make the quality of the data received increasingly worse. In this paper we have assumed that the precision of the input data is known and constant. As for the latter, it would not be difficult to deal with variable precision, since functions can be used to define the uncertainty of the `UReal` values. How to deal with unknown and imprecise precision (*i.e.,* a type of second-order uncertainty) remains part of our future work.

Usage complexity. This proposal adds a certain level of complexity related to the need to compare probabilistic values, which is not required in more conventional approaches, such as those based on the inclusion of error bounds. Instead of a single comparison, the developer must provide a piece of code that returns a Boolean value under the probabilistic comparison, depending on the confidence level that can be accepted. While this introduces some additional complexity, at the same time it clearly provides more refined and accurate results in terms of the final quality of the developed model.

4.4 Open questions

In addition to the potential benefits and limitations of our proposal, this section discusses some open issues that we have found during its evaluation.

Domain expert implication. When incorporating measurement uncertainty information into a model, sometimes it is difficult to identify the attributes that are subject to uncertainty. In general, all attributes that represent physical variables should be subject to uncertainty, but there might be others. For example, some constants should be endowed with uncertainty, too. The variability of the duration of tasks in certain processes, or of the cost of a given product due to currency exchange fluctuations, are uncertainties that need to be estimated. For this, the judgment of the domain expert is essential, and communication with them is needed to clarify which attributes should be endowed with this kind of information.

Representing and operating with uncertainty. There are different ways of representing measurement uncertainty, especially for the uncertainty associated to numeric values. They include ranges, probability distributions of the values, or the standard deviation of the variability of the measured attribute. From all the available alternatives, we decided to use a library that implements the ISO VIM recommended representation and management of measurement uncertainty, as defined in the GUM [18], which is also the notation used in most engineering disciplines. Ranges and other kinds of possible expressions of the measurements deviations can be reduced to this representation [18]. Similarly, Bayesian Probability [5], Fuzzy logic [46] or uncertainty theory [24] can be used to assign confidence to uncertain Boolean values. All these theories have advantages and limitations (see, *e.g.,* [6, 21, 24]) but, as previously mentioned, we decided to use Bayesian Probability, which is the one that, in our opinion, is the most well known and easily understood by software engineers. A proper comparison analysis between the different approaches is left for future work. Likewise, the use of Type A representation of uncertainty, which uses the value samples instead of closed-form equations to represent and propagate the uncertainty is something that we would like to explore further. Although a priori this would have a significant input on the performance of the uncertainty analysis, the use of cheap hardware accelerators such as graphic cards might provide effective solutions for these simple vector operations and therefore we could deal with uncertainty in a more statistically precise manner.

5 RELATED WORK

Uncertainty in control systems and their simulation has been traditionally represented and managed using two main approaches.

In the first place, intervals to represent the possible values of uncertain attributes have been extensively used in the simulation domain. For example, Fujimoto [11, 25] use time intervals to deal with the concepts of Approximate Time and Approximate Time Event Ordering in the context of DEVS [43]. In their proposal, two events are considered concurrent if the intervals representing their timestamps have a non-empty intersection. Other authors have proposed to introduce uncertainty on the spatial properties of the model for obtaining speed-ups [13, 31]. Saadawi and Wainer also explored replacing time datatype in DEVS models by intervals in their RTA-DEVS formalism [34]. Furthermore, two new extensions to DEVS, called UA-DEVS and IA-DEVS, provide methods to specify uncertainty in the state, input, and output variables in addition to the time variable [40]. The former defines a formal specification of models including uncertainty specifications as intervals. The latter enables the simulation of UA-DEVS models based on computational constraints (time, memory, etc.). This separation of concerns allows the domain expert to define the model once, and then simulate it with different constraints without redefining the model. Other approaches, such as [20], make conservative decisions based on intervals to robustify the specification of controllers of cyber-physical systems so that they satisfy safety requirements under uncertain conditions.

We see two major limitations of approaches based on intervals for specifying the possible values of uncertain variables. On the one hand, they are very coarse-grained as we have seen in the examples shown above, which results in very conservative (also called *cautious*) simulations [42]. On the other hand, specifying and operating with intervals require a significant effort by the modeler since there is no direct support for making computations with them, such as arithmetic operations or comparisons, which are really burdensome and error-prone tasks.

Other set of works study the relationship the uncertainty of the input parameters and that of the simulation results, aiming at defining measures for risk quantification under input uncertainty. In general, there are two sources of uncertainty in a typical stochastic simulation experiment: the extrinsic uncertainty on input parameters (also called input parameter uncertainty), and the intrinsic uncertainty on output response (referred to as stochastic uncertainty) that reflects the inherent stochasticity of the system. The variability of simulation output response depends on both input uncertainty and stochastic uncertainty. Some authors [16, 45] propose nested Monte Carlo simulation approaches to estimate them. Others [4] propose statistical methods for the calculation of confidence intervals for the mean of a simulation output. As in our case, they obtain more accurate results than those proposals that use interval arithmetic or very conservative (*i.e.,* robust) estimations. However, both the complexity of their calculations and their computational costs might hinder their applicability. In our case, the fact that we assume Normal distributions and that uncertainty propagation is achieved using closed-form solutions mitigate these issues.

Another group of papers provides alternative approaches to exploit approximation (hence uncertainty) for improving the trade-off between performance and representativeness of simulation output, under uncertain event occurrence [11, 31] or using approximated rollbacks [30]. Our proposal is orthogonal to these approaches, as each focuses on different aspects of uncertainty.

Similarly, existing schemes for adaptive control used in industry provide reasonable heuristic approaches, although they have the limitation that parameter uncertainties are not usually taken into account in the design of the controller. This has led to the notion of *dual control* [9, 39], which addresses this issue by considering parameter uncertainties. In particular, explicit dual control algorithms, such as the ones used in our examples, are based on the minimization of cost functions defined in terms of control losses and uncertainty measurements (the measure of precision of the parameter estimation) [37]. Basically, the controller has a dual action: it follows the control goal, i.e., the system output cautiously tracks the desired reference value; and it excites the plant so that the control quality becomes better in future time intervals. One of the known problems with such control algorithms is that they are complicated and not always feasible to implement in practical problems [37], which hinders their applicability in real systems. What we have shown in this paper is that the use of a type system that provides basic support for explicitly representing and operating with uncertain attributes and propagating their associated uncertainty transparently greatly simplifies these problems. This makes it possible to obtain the advantages of dual control algorithms while minimizing their limitations.

In this context, the explicit representation of uncertainty is also a challenge, especially in the context of software models. The survey [38] covers current approaches, although significant challenges remain to be addressed. In particular, there are very few libraries for programming or modeling languages that support measurement uncertainty, i.e., the representation and operation of uncertain datatypes [3]. Even those that support the propagation of uncertainty (e.g., [2, 22, 23, 41]) are quite complex to use and do not support the comparison between uncertain numbers. This is a general problem that we have observed in most uncertainty modeling proposals: they only deal with uncertain reals. However, in the physical world, all other primitive data types also have uncertain values. In particular, logical variables representing decisions or comparisons between quantities rarely have crisp true or false values. Instead, extensions to the Boolean logic enable dealing with this type of uncertainty, including probability theory [5, 8], possibility theory (based on fuzzy logic [33, 46]), plausibility (a measure in the Dempster-Shafer theory of evidence [36]), and uncertainty theory [24]. These approaches assign different probabilities to propositions, rather than truth values, and probability formulas replace truth tables. From the surveyed literature, in this paper we use the proposal presented in [3], which provides a Java library that supports all UML and OCL primitive datatypes endowed with uncertainty. Moreover, as mentioned above, probabilities tend to be easier for engineers to understand and manage than other measures that quantify confidence or the likelihood of failure.

6 CONCLUSIONS AND FUTURE WORK

In this paper we have proposed to include the sources of uncertainty in system models as first-class entities using random variables, in order to simulate control systems more faithfully, including not only the use of random variables to represent and operate with uncertain values, but also to represent decisions based on their comparisons. We have illustrated the problem with the toy example of a Chasing Robot, and validated our approach on the ZNN case study, which is a standard for the self-adaptive systems community.

Uncertainty is inherent in cyber-physical systems and we strongly believe that it should be handled explicitly at every level, from requirements to design to code and validation. We have shown that this is not so difficult to implement by leveraging emerging libraries for supporting sound computations on random variables. We hope this paper would help in triggering a wider adoption of stochastic approaches for software controlling cyber-physical systems.

ACKNOWLEDGMENTS

We would like to thank the reviewers for their very valuable comments and suggestions, which helped us significantly to improve the paper. This work was partially funded by the Spanish Government (FEDER/Ministerio de Ciencia e Innovación – Agencia Estatal de Investigación) under projects PID2021-125527NB-I00 and TED2021-130523B-I00.

REFERENCES

[1] Michael Athans. 1971. Editorial on the LQG problem. *IEEE Trans. Autom. Control* 16, 6 (1971), 528.

[2] Michaël Baudin, Anne Dutfoy, Bertrand Iooss, and Anne-Laure Popelin. 2016. *OpenTURNS: An Industrial Software for Uncertainty Quantification in Simulation.* Springer, 1–38. https://doi.org/10.1007/978-3-319-11259-6_64-1 https://openturns.github.io/.

[3] Manuel F. Bertoa, Loli Burgueño, Nathalie Moreno, and Antonio Vallecillo. 2020. Incorporating measurement uncertainty into OCL/UML primitive datatypes. *Softw. Syst. Model.* 19, 5 (2020), 1163–1189. https://doi.org/10.1007/s10270-019-00741-0

[4] R. C. H. Cheng and W. Holland. 2004. Calculation of Confidence Intervals for Simulation Output. *ACM Trans. Model. Comput. Simul.* 14, 4 (Oct. 2004), 344—-362. https://doi.org/10.1145/1029174.1029176

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

[6] Didier Dubois and Henri Prade. 1993. Fuzzy sets and probability: Misunderstandings, bridges and gaps. In *Proc. of the IEEE Conf. on Fuzzy Systems.* IEEE, 1059–1068. https://doi.org/10.1109/FUZZY.1993.327367

[7] Naeem Esfahani and Sam Malek. 2013. Uncertainty in Self-Adaptive Software Systems. In *Software Engineering for Self-Adaptive Systems II.* LNCS, Vol. 7475. Springer, 214–238.

[8] W. Feller. 2008. *An Introduction to Probability Theory and Its Applications.* Wiley.

[9] Nikolai M. Filatov and Heinz Unbehauen. 2000. Survey of adaptive dual control methods. *IEEE Proceedings Control Theory and Application* 147, 1 (2000), 118–128. https://doi.org/10.1049/ip-cta:20000107

[10] Antonio Filieri et al. 2015. Software Engineering Meets Control Theory. In *Proc. of SEAMS'15.* IEEE Computer Society, 71–82. https://doi.org/10.1109/SEAMS.2015.12

[11] Richard Fujimoto. 1999. Exploiting Temporal Uncertainty in Parallel and Distributed Simulations. In *Proc. of PADS'99.* IEEE Computer Society, 46–53. https://doi.org/10.1109/PADS.1999.766160

[12] David Garlan. 2010. Software engineering in an uncertain world. In *Proc. of FoSER'10.* 125–128. https://doi.org/10.1145/1882362.1882389

[13] Valerio Gheri, Giovanni Castellari, and Francesco Quaglia. 2008. Controlling Bias in Optimistic Simulations with Space Uncertain Events. In *Proc. of DS-RT'08.* IEEE Computer Society, 157–164. https://doi.org/10.1109/DS-RT.2008.37

[14] Martin Gogolla, Fabian Büttner, and Mark Richters. 2007. USE: A UML-based specification environment for validating UML and OCL. *Sci. Comput. Program.* 69, 1-3 (2007), 27–34. https://doi.org/10.1016/j.scico.2007.01.013

[15] Martin Gogolla, Frank Hilken, and Khanh-Hoang Doan. 2018. Achieving Model Quality through Model Validation, Verification and Exploration. *Computer Languages, Systems & Structures* 54 (Dec. 2018), 474–511. https://doi.org/10.1016/j.cl.2017.10.001

[16] Michael B. Gordy and Sandeep Juneja. 2010. Nested Simulation in Portfolio Risk Measurement. *Management Science* 56 (Aug. 2010), 1833—-1848. https://doi.org/10.1287/mnsc.1100.1213

[17] IEEE 1788-2015. 2015. IEEE Standard for Interval Arithmetic. https://standards.ieee.org/ieee/1788/4431/.

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

[19] Deepali Kholkar, Suman Roychoudhury, Vinay Kulkarni, and Sreedhar Reddy. 2022. Learning to Adapt – Software Engineering for Uncertainty. In *Proc. ISEC'22.* ACM, 21:1–21:5. https://doi.org/10.1145/3511430.3511449

[20] Tsutomu Kobayashi, Rick Salay, Ichiro Hasuo, Krzysztof Czarnecki, Fuyuki Ishikawa, and Shin-ya Katsumata. 2021. Robustifying Controller Specifications of Cyber-Physical Systems Against Perceptual Uncertainty. In *Proc. of NASA Formal Methods 2021 (LNCS, Vol. 12673).* Springer, 198–213. https://doi.org/10.1007/978-3-030-76384-8_13

[21] Bart Kosko. 1990. Fuzziness vs. Probability. *International Journal of General Systems* 17, 2–3 (1990), 211–240. https://doi.org/10.1080/03081079008935108

[22] Eric O. Lebigot. 2016. Uncertainties package. https://pythonhosted.org/uncertainties/. Accessed: May 30, 2022.

[23] Abraham Lee. 2013. SOERP Uncertainties package. https://pypi.org/project/soerp/. Accessed: May 30, 2022.

[24] Baoding Liu. 2018. *Uncertainty Theory* (5 ed.). Springer. http://orsc.edu.cn/liu/ut.pdf

[25] Margaret L. Loper and Richard M. Fujimoto. 2000. Pre-sampling as an approach for exploiting temporal uncertainty. In *Proc. of PADS'00.* IEEE Computer Society, 157–164. https://doi.org/10.1109/PADS.2000.847159

[26] Giovanni Lugaresi and Andrea Matta. 2018. Real-Time simulation in manufacturing Systems: Challenges and Research Directions. In *Proc. of WSC'18.* IEEE, 3319–3330. https://doi.org/10.1109/WSC.2018.8632542

[27] Sara Mahdavi-Hezavehi, Paris Avgeriou, and Danny Weyns. 2017. *A Classification Framework of Uncertainty in Architecture-Based Self-Adaptive Systems With Multiple Quality Requirements.* Morgan Kaufmann, Boston, Chapter 3, 45–77. https://doi.org/10.1016/B978-0-12-802855-1.00003-4

[28] Object Management Group. 2014. *Object Constraint Language (OCL) Specification. Version 2.4.* OMG Document formal/2014-02-03.

[29] Victor Ortiz, Loli Burgueño, Antonio Vallecillo, and Martin Gogolla. 2019. Native Support for UML and OCL Primitive Datatypes Enriched with Uncertainty in USE. In *Proc. of OCL@MODELS'19 (CEUR Workshop Proceedings, Vol. 2513).* CEUR-WS.org, 59–66. http://ceur-ws.org/Vol-2513/paper5.pdf

[30] Matteo Principe, Andrea Piccione, Alessandro Pellegrini, and Francesco Quaglia. 2020. Approximated Rollbacks. In *Proc. of SIGSIM-PADS'20*. ACM, 23–33. https://doi.org/10.1145/3384441.3395984

[31] Francesco Quaglia and Roberto Beraldi. 2004. Space Uncertain Simulation Events: Some Concepts and an Application to Optimistic Synchronization. In *Proc. of PADS'04*. IEEE Computer Society, 181–188. https://doi.org/10.1109/PADS.2004.1301299

[32] Jürgen Roßmann, Eric Guiffo Kaigom, Linus Atorf, Malte Rast, Georgij Grinshpun, and Christian Schlette. 2014. Mental Models for Intelligent Systems: eRobotics Enables New Approaches to Simulation-Based AI. *Künstliche Intell.* 28, 2 (2014), 101–110. https://doi.org/10.1007/s13218-014-0298-z

[33] Stuart J. Russell and Peter Norvig. 2010. *Artificial Intelligence. A Modern Approach* (3 ed.). Prentice Hall.

[34] Hesham Saadawi and Gabriel A. Wainer. 2010. Rational time-advance DEVS (RTA-DEVS). In *Proc. of SpringSim'10*. SCS/ACM, 143:1–143:8. https://doi.org/10.1145/1878537.1878686

[35] Bradley R. Schmerl, Javier Cámara, Jeffrey Gennari, David Garlan, Paulo Casanova, Gabriel A. Moreno, Thomas J. Glazier, and Jeffrey M. Barnes. 2014. Architecture-based self-protection: composing and reasoning about denial-of-service mitigations. In *Proc. of HotSoS'14*. ACM, 2:1–2:12. https://doi.org/10.1145/2600176.2600181

[36] Glenn Shafer. 1976. *A Mathematical Theory of Evidence*. Princeton University Press.

[37] Pankaj Swarnkar, Shailendra Kumar Jain, and R.K Nema. 2014. Adaptive Control Schemes for Improving the Control System Dynamics: A Review. *IETE Technical Review* 31, 1 (2014), 17–33. https://doi.org/10.1080/02564602.2014.890838

[38] Javier Troya, Nathalie Moreno, Manuel F. Bertoa, and Antonio Vallecillo. 2021. Uncertainty representation in software models: a survey. *Softw. Syst. Model.* 20, 4 (2021), 1183–1213. https://doi.org/10.1007/s10270-020-00842-1

[39] Heinz Unbehauen. 2000. Adaptive dual control systems: a survey. In *Proc. of AS-SPCC'00*. IEEE, 171–180. https://doi.org/10.1109/ASSPCC.2000.882466

[40] Damián Vicino, Gabriel A. Wainer, and Olivier Dalle. 2022. Uncertainty on Discrete-Event System Simulation. *ACM Trans. Model. Comput. Simul.* 32, 1 (2022), 2:1–2:27. https://doi.org/10.1145/3466169

[41] Wikipedia. Accessed: May 30, 2022. List of uncertainty propagation software. https://en.wikipedia.org/wiki/List_of_uncertainty_propagation_software.

[42] B. Wittenmark. 1975. Stochastic adaptive control methods: a survey. *Internat. J. Control* 21, 5 (1975), 705–730. https://doi.org/10.1080/00207177508922026

[43] Bernard P. Zeigler, Alexandre Muzy, and Ernesto Kofman. 2018. *Theory of modeling and design: Discrete Event and Iterative System Computational Foundations* (3 ed.). Academic Press.

[44] Man Zhang, Bran Selic, Shaukat Ali, Tao Yue, Oscar Okariz, and Roland Norgren. 2016. *Understanding Uncertainty in Cyber-Physical Systems: A Conceptual Model*. LNCS, Vol. 9764. Springer, 247–264. https://doi.org/10.1007/978-3-319-42061-5_16

[45] Helin Zhu, Tianyi Liu, and Enlu Zhou. 2020. Risk Quantification in Stochastic Simulation under Input Uncertainty. *ACM Trans. Model. Comput. Simul.* 30, 1 (Feb. 2020), 1:1–1:24. https://doi.org/10.1145/3329117

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

Received 22 July 2022; revised 7 February 2023; accepted 23 March 2023