Expressing Measurement Uncertainty in Software Models

Antonio Vallecillo, Carmen Morcillo, and Priscill Orue
Universidad de Málaga
Málaga, Spain
Email: {av,aixela,priscill.orue}@uma.es

***Abstract*—Uncertainty is an inherent property of any measure or estimation performed in any physical setting, and therefore it needs to be taken into consideration when modelling systems that manage real data. Although several modelling languages permit the representation of measurement uncertainty for describing certain system attributes, these aspects are not normally incorporated into their type systems. Thus, operating with uncertain values and propagating uncertainty are normally cumbersome processes and difficult to achieve at the model level. This paper proposes an extension of the UML and OCL types to incorporate data uncertainty coming from physical measurements or user estimations into the models, together with the set of operations defined for the values of these types.**

I. Introduction

The emergence of Cyber-Physical Systems (CPS) and the Internet of Things (IoT), as examples of systems that have to interact with the physical world, has made evident the need to faithfully represent some extra-functional properties of the modelled systems and of their elements, as well as the current limitations of existing modelling languages and tools.

Several authors [1], [2], [3] have already warned about the lack of expressiveness in current software models for capturing and manipulating in an appropriate manner some key aspects of the real world, such as concurrency, units, precision or real time properties. For example, mainstream programming languages do not have a first-class concept of "physical" values, and the problem is that user-defined types are not enough: a compiler would still not catch unit mismatches or know how to compare two or more values of such a type [3]. This is specially relevant in the case of CPS that combine software, hardware, network and physical elements [4].

One aspect of particular relevance when modelling physical systems and their properties is related to the *uncertainty* of the values of the attributes of the modelled entities, specially when dealing with certain of their *quality characteristics* such as precision, performance or accuracy.

Data uncertainty can come from different reasons, including variability of input variables; numerical errors or numerical approximations of some parameters; observation errors; measurement errors, or simply lack of knowledge of the true behavior of the system or of its underlying physics [5]. On other occasions estimations are needed because the exact values cannot be obtained since the associated properties are not directly measurable or accessible, because values are too costly to measure, or simply because they are unknown. For example, the duration of a given task in a software process, the cost of certain product, the number of bugs in a software component, or the duration of a battery. At other times we have to invite some domain experts to evaluate the belief degree that an event will happen, or to ask for their estimations of a given value. Such estimates normally feature ranges, or intervals, not exact values, which determine the possible lower and upper bounds for the exact values, or are given by a probability distribution that represents a range of its variation.

In general, all measurements are subject to uncertainty and a measurement result is complete only when it is accompanied by a statement of the associated uncertainty [5], [6]. In metrology, measurement uncertainty is a non-negative parameter characterizing the dispersion of the values attributed to a measured quantity. This parameter defines the interval that determines the range of possible values of the measured system property and its limits of variation [7].

Several modelling languages, such as MARTE [8] or SysML [9], already permit the representation of measurement uncertainty for describing some system properties. Normally, stereotypes added to class attributes are used to represent tolerance or precision. Likewise, some Business Process Modelling notations (e.g., [10]) also consider uncertainty, for example when modelling the arrival time of clients, the availability of some resources or the duration of some tasks. These works use probabilistic mass functions for modelling the values of the corresponding attributes, instead of fixed values. However, these aspects are not incorporated into their type systems and therefore operating with uncertain values or propagating uncertainty are normally cumbersome processes and difficult to achieve at the model level, specially when dealing with attributes whose values are given by arbitrary distributions.

Similarly, several mathematical libraries and tools already exist (see [11] for a comprehensive list) for propagating measurement uncertainty and operating with uncertain values. Their current integration with software models is, however, quite limited.

This paper shows how measurement uncertainty can be incorporated into software models. In particular, we discuss an extension of UML [12] and OCL [13] type `Real` and a set of operations defined on the values of this type. Both analytical and approximate algorithms have been developed to implement these operations, and a Java library provides several of these implementations. This approach can then be used to add this kind of information to further modelling elements.

This paper is structured as follows. First, Section II briefly introduces the concepts related to measurement uncertainty that will be used throughout the paper. Then, Section III describes our proposal and the algebra of operations on uncertain values and the implementations we have developed for these operations. Section IV illustrates some usage scenarios and applications of the proposal. Section V compares our work to similar proposals. Finally, we conclude the paper in Section VI with an outlook on future work.

II. Background

A. Uncertainty

Uncertainty is the quality or state that involves imperfect and/or unknown information. It applies to predictions of future events, estimations, physical measurements, or unknown properties of a system [5].

Measurement (or data) uncertainty refers to the inability to know with complete precision the value of a quantity. For instance, when modelling CPS, measurement uncertainty normally arises in partially observable and/or stochastic environments, or when the system properties are not directly measurable or accessible. However, software models use typed attributes (e.g., `Integer` or `Real`) to represent system properties, but these types only permit exact values.

Example 1: To illustrate our proposal, suppose the Java `Rectangle` object which specifies an area in a plane that is enclosed by the rectangle upper-left point (x, y) in the coordinate space, its width w and its height h. These four parameters are normally modeled by `Real` values that represent the values of the corresponding object's attributes. However, the measures of the rectangle's height and width are never fully accurate. Different measurements always produce slightly different results, depending on how the rule is calibrated, the skills of the person performing the measure, etc. Similarly, the values that determine the position of the rectangle are never 100% accurate and certain tolerance should be taken into consideration when the object represented by the rectangle should be picked up, manipulated or moved by a machine.

In this context, uncertainty depends on both the accuracy and precision of the measurement instrument. The lower the accuracy and precision of an instrument, the larger the measurement uncertainty is. Often, the uncertainty of a measurement is found by repeating the measurement enough times to get a good estimate of the standard deviation of the values. Then, any single value has an uncertainty equal to the standard deviation. This is why instead of giving a single number x to model a measurement result, engineers normally use $x \pm u$ to represent the result, being u the *associated standard uncertainty.*

For example, instead of saying that the height of the Rectangle is $h = 3.0$, we should say that its height is, e.g., $h = 3.0 \pm 5 \cdot 10^{-5}$ which gives a better indication not only of the result but also of the possible deviations due to measurement variability.

Example 2: Suppose now a Factory Assembly Line (FAL) that specifies a manufacturing process composed of steps that can be combined in parallel or in sequence. Each step has a set of inputs, produces a set of outputs, has a duration and consumes a set of resources, each one with an associated cost. BPMN models or UML Activity Diagrams are useful notations to represent such kinds of systems and to predict some of their properties, such as the shortest and longest times to produce a given set of outputs, or the average production cost per produced part. The problem is that some of these attributes are difficult to capture using exact values: normally intervals with estimated values are more appropriate because uncertainty is unavoidable in these cases. For instance, the system owner may want to represent that the duration of assembling a certain component may range between 2 and 3 seconds, depending on the person performing the task; or that the cost of a given part may vary between 1.30 and 1.36 Euros depending on the daily Dollar exchange rate. Current software models tend to use the average values, assigning a duration of 2.5 seconds to the task and a cost of 1.33 Euros to the part. However, it would be more expressive to represent these quantities as intervals $d = [2.0, 3.0]$ or $c = [1.30, 1.36]$ assuming we have mechanisms to operate with these interval types.

Furthermore, when composing several uncertain measures or estimations we have to consider that the individual deviations accumulate, producing significant variations in the results. For example, when a set of tasks is performed in sequence, the range of variation of the total duration of the process can significantly grow and this needs to be taken into account in the model if we want the model to be faithful and accurate. This can easily happen when we have a derived attribute that is calculated by aggregating others (for instance, the total duration of a task as the sum of the steps that comprise it). So we are talking about the need of not only representing uncertain values, but also operating with them in the models. This leads to the need to consider into our models the *propagation of uncertainty* that we will discuss later in Section II-C.

B. Measurement Uncertainty

Uncertainty of measurement is defined by the ISO VIM [7] as "a parameter, associated with the result of a measurement, that characterizes the dispersion of the values that could reasonably be attributed to the measurand. The parameter may be, for example, a standard deviation (or a given multiple of it), or the half-width of an interval having a stated level of confidence."

It is important to avoid confusing the terms *error* and *uncertainty*. The former is the difference between the measured value and the true value of the object. Uncertainty is a quantification of the doubt about the measurement result. It is also used to quantify measures when the measuring tools have some *tolerance*.

Uncertainty is normally expressed as the standard deviation of the probability distribution that describes the possible values of the quantity being measured. For example, if the measures of a given quantity X follow a Normal distribution with mean x and standard deviation $u = \sigma$, the interval $[x - \sigma, x + \sigma]$ represents a range of its variation, and we know that it will contain 68.3% of the possible values of X.

The "Guide to the Expression of Uncertainty in Measurement" (GUM) [5] defines the term *standard uncertainty* as "the uncertainty of the result of a measurement expressed as a standard deviation." In the following, we will refer to x to the estimated value and u (or u_x when we want to refer to the precise variable) to its standard uncertainty.

The GUM framework also identifies two ways of evaluating the uncertainty of a measurement, depending on whether the knowledge about the quantity X is inferred from repeated measured values ("Type A evaluation of uncertainty"), or scientific judgement or other information concerning the possible values of the quantity ("Type B evaluation of uncertainty").

In Type A evaluation of uncertainty, if $X = \{x_1, \ldots, x_n\}$ is the set of measured values, then the estimated value x is taken as the mean of these values, and the associated uncertainty u as their *experimental standard deviation*, i.e., $u_x^2 = \frac{1}{n(n-1)} \sum_{i=1}^{n} (x_i - x)^2$ [5].

In Type B evaluation, we are able to estimate only upper and lower bounds for the values of X and there is no further information about the possible values of X within the interval. Thus, we can only assume a uniform or rectangular distribution of the possible values of X. Then x is taken as the midpoint of the interval, $x = (a + b)/2$, and its associated variance as $u^2 = (b - a)^2/12$. Therefore, $u = (b - a)/(2\sqrt{3})$ [5].

With this, we can also calculate the uncertainty associated to variables whose variation is given in terms of percentages. Suppose that the value x of an attribute is subject to a possible variation of z% of its nominal value, i.e., x the expected value of x lies within the interval $[x(1 - z), x(1 + z)]$. Using the formula above,the associated uncertainty u corresponds to $u = xz/\sqrt{3})$. For instance, this means that if $x = 1.0$ and $z = \pm 10\%$, then $u = 0.057735027x$

Of course, in case there is information available about the distribution of the values within the $[a, b]$ interval (e.g., Normal if they symmetrically concentrate around the midpoint or U if they concentrate in the endpoints), the mean and standard deviation of such a distribution should be used instead.

Finally, given that normally the interval $[x - u, x + u]$ contains 68.3% of the expected values, the GUM also defines the *Extended Uncertainty* which multiplies the associated uncertainty by a constant positive integer factor (the *coverage factor k*) to improve the coverage. Then, if we need to consider a wider coverage of that interval (in order to, e.g., account for more values of the measured quantity), we can take the extended uncertainty with $k = 2$ that will account, in case of a Normal distribution, for the 95.4% of the values, or $k = 3$ that will account for 99.7% of them.

C. Indirect measurements and propagation of uncertainty

The discussion above considers only the measure or estimation of individual attributes. But in general we need to combine them to produce an aggregated measure, or to calculate a derived attribute. For example, to compute the area of the Rectangle we need to consider its height and its width, combining them by multiplication. The individual uncertainties of the input quantities need to be combined too, to produce the uncertainty of the result. This is known as the *propagation of uncertainty*, or *uncertainty analysis*.

Similarly for the second example, imagine two consecutive steps A, B with durations $d_A = [2, 3]$ and $d_B = [1.5, 3.5]$, which consume 2 resources each with associated costs $c_{A1} = [1.3, 1.5], c_{A2} = [3.0, 3.2], c_{B1} = [2.4, 2.6], c_{B2} = [3.8, 4.2]$. If we use single numbers and the midpoints of the intervals, we will conclude in our model that the duration of $A; B$ is 4 and its cost is 11.0. However, the duration of $A; B$ can range between 3.5 and 6.5, which could represent a significant delay of more than 50% w.r.t the calculated output. Similarly, the cost can range between 10.5 and 11.5, which may mean one Euro difference per part. Hence the importance of dealing with possible variations in the data, specially when the variations accumulate as data is aggregated and combined to produce the system outputs.

Uncertainty Analysis is in general a difficult problem since combining the probability distributions of the individual uncertainties is not a trivial task [5]. In fact, in the general case it does not have an analytical solution but requires simulations [6].

1) Analytical solutions: The most common case is when the individual uncertainties follow Normal or Rectangular distributions [5]. For them either analytical solutions exist or they can be calculated based on a first-order Taylor series approximation of the combination function. This is known as the *law of propagation of uncertainty* and it is fully described in [5]. We will use these analytical solutions for specifying our OCL and Java operations described later in Sect. III.

2) Simulation solutions: There are cases in which the two quantities to combine are quite different, and therefore analytical solutions for the aggregated uncertainty cannot be easily computed. This is also the case when the linearization of the model provides an inadequate representation, or the probability density function for the resulting quantity significantly differs from a Gaussian distribution or a scaled and shifted t-distribution (e.g. due to marked asymmetry). In this cases the GUM recommends to use simulation, using the Monte Carlo method described in [6]. The basis of this approach is a repeated random sampling from the probability density function of the input quantities $\{X_1, \ldots, X_n\}$ that need to be aggregated, and the combination of the samples to produce the samples of the derived quantity X. From these, the expected value x of X and its uncertainty u, are calculated as in Type A evaluation of uncertainty. That it, x as the mean of the samples and u as its standard deviation. In other words, operations on quantities specified in this way are performed on the samples, given that their distributions are unknown or their combinations do not permit analytical solutions.

The number of samples heavily depends on the shape and complexity of the probability density function, although in general a number of 10^6 can often be expected to deliver a 95% coverage interval for the output quantity such as this length is correct to one or two significant digits [6].

III. An Algebra of Operations

Our goal is to extend UML and OCL languages to declare uncertain variables. Main benefits include: the expression of uncertainty in variables of software models; being able to compute uncertainty at the model level (using the set of operations defined for these types); permit to transfer the information to standard algorithms and tools to compute complex types of uncertainties, including automatic coupling with Modelica (http://www.modelica.org/) and OpenTURNS (http://www.openturns.org/) tools. In other words, high-level representation and manipulation of uncertainty, and platform-independence.

Our proposed solution consists of defining a new type in UML, called `UReal` (Fig. 1) which can be considered as a superclass of `Real`. The values of `UReal` type are pairs of `Real` numbers $X = (x, u)$. They define the expected value (x) and associated standard uncertainty (u) of a quantity X. In the following, we indistinctly refer to them as (x, u) or as $x \pm u$ depending on the context, for readability purposes.

The immersion of the subclass into the superclass is naturally defined by identifying a real number r with the `UReal` value $(r, 0)$. The operations can be defined using the two ways previously mentioned: either analytically or by simulation (see sections III-A and II-C2 below).

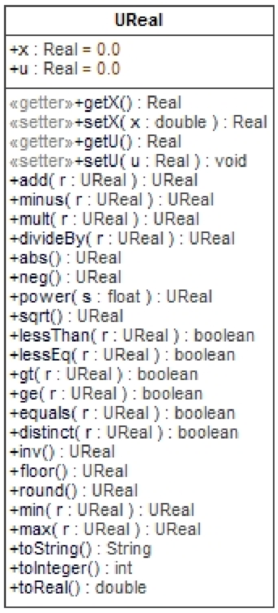
UReal
+x : Real = 0.0
+u : Real = 0.0
«getter»+getX() : Real
«setter»+setX(x : double) : Real
«getter»+getU() : Real
«setter»+setU(u : Real) : void
+add(r : UReal) : UReal
+minus(r : UReal) : UReal
+mult(r : UReal) : UReal
+divideBy(r : UReal) : UReal
+abs() : UReal
+neg() : UReal
+power(s : float) : UReal
+sqrt() : UReal
+lessThan(r : UReal) : boolean
+lessEq(r : UReal) : boolean
+gt(r : UReal) : boolean
+ge(r : UReal) : boolean
+equals(r : UReal) : boolean
+distinct(r : UReal) : boolean
+inv() : UReal
+floor() : UReal
+round() : UReal
+min(r : UReal) : UReal
+max(r : UReal) : UReal
+toString() : String
+toInteger() : int
+toReal() : double

Figure 1. UML `UReal` type values and operations.

A. Analytical definition of operations

The operations on type `UReal` values (shown in Fig. 1 are defined using the equations defined in [5] in case they follow Normal or Rectangular probability distributions. For simplicity, we will assume that all variables are mutually independent (and hence their covariances are 0). With this, the specifications in OCL of the `UReal` type operations can be described as follows.

```
context UReal::add(r:UReal) :UReal
post: result.x = self.x + r.x and
      result.u = (self.u*self.u + r.u*r.u).sqrt()

context UReal::minus(r:UReal):UReal
post: result.x = self.x - r.x and
      result.u = (self.u*self.u - r.u*r.u).sqrt()

context UReal::mult(r:UReal) :UReal
post: result.x = (self.x*r.x) and
      result.u = (r.u*r.u*self.x*self.x +
              self.u*self.u*r.x*r.x).sqrt()

context UReal::divideBy(r:UReal) :UReal
  pre:  not r.equals(0,0)
  post: result.x = (self.x/r.x +
                (self.x*r.u*r.u)/(r.x*r.x*r.x))
    and result.u = ((self.u*self.u/r.x) +
                  ((r.u*r.u*self.x*self.x)/
                   (r.x*r.x*r.x*r.x))).sqrt()

context UReal::abs() :UReal
  post: result.x = (self.x).abs() and
        result.u = self.u

context UReal::neg() :UReal
  post: result.x = -self.x and result.u = self.u

context UReal::power(s : Real) :UReal
  post: result.x = self.x.power(s)+((s*(s-1))/2)*
            self.x.power(s-2)*(self.u*self.u) and
        result.u = s*self.u*self.x.power(s-1)
```

These specifications require a library of operations for OCL that permits calculating square roots and other mathematical operations on real numbers, such as those currently available for ATL [14] or defined in [15], [16].

Given that type `UReal` should be a subtype of `oclAny`, it should also implement the equal "=" and distinct "<>" comparison operations. In fact one of the most important applications of error propagation is comparing two quantities with uncertainty. In general there is no experiment that can performed to prove that two quantities are equal; the best that can be done is to measure them so precisely that a very close bound can be stated on their difference [17]. A generally accepted and more practical equality test between quantities with uncertainty is to check if their intervals overlap, when expressed in the same units [18].

With this, the comparison operations can be defined as follows:

```
context UReal::equals(r :UReal) :Boolean
  post: result = (self.x-self.u).max(r.x-r.u) <=
                 (self.x+self.u).min(r.x + r.u)
context UReal::distinct(r :UReal) :Boolean
  post: result = not self.equals(r)
```

Note that in these formulae we are using a 68.3% confidence level. It would be possible to relax the equality test by increasing the interval to 2 or 3 times the standard deviation, using the *Extended Uncertainty* with a *coverage factor* of $k = 2$ or $k = 3$ that would cover, respectively, 95.4% or 99.7% of the possible values of these quantities, as we shall later discuss in Sect. III-C.

With the equality operation we can then define the rest of the comparison operations, which also return a `Boolean` value:

```
context UReal::lessThan(r :UReal) :Boolean
 post: result = (self.x<r.x) and
               ((self.x+self.u)<(r.x-r.u))
context UReal::lessEq(r :UReal) :Boolean
 post: result=self.lessThan(r) or self.equals(r)
```

With the comparison operations, maximums and minimums are easy to define:

```
context UReal::max(r :UReal) :UReal
  post: result = if self.lessThan(r) then r
                 else self endif
context UReal::min(r :UReal) :UReal
  post: result = if self.lessThan(r) then self
                 else r endif
```

We should also consider further operations on `UReal` values:

```
context UReal::floor() :UReal
  post: result.x = self.x.floor() and
        result.u = self.u
context UReal::round() :UReal
  post: result.x = self.x.round() and
        result.u = self.u
```

B. Simulation of operations

We have also developed a Java library to implement the operations defined for type `UReal`. For this we have defined one Java interface (`IUReal`), and two implementations.

```
public interface IUReal {
  public double getX();
  public void setX(double d);
  public double getU();
  public void setU(double u);
  public IUReal add(IUReal r);
  public IUReal minus(IUReal r);
  public IUReal mult(IUReal r);
  public IUReal divideBy(IUReal r);
  public IUReal abs();
  public IUReal neg();
  public IUReal power(float s);
  public IUReal sqrt();
  public boolean lessThan(IUReal r);
  public boolean lessEq(IUReal r);
  public boolean gt(IUReal r);
  public boolean ge(IUReal r);
  public boolean equals(IUReal r);
  public boolean distinct (IUReal r);
  public int hashcode();
  public String toString();
  public IUReal inverse();
  public IUReal floor();
  public IUReal round();
  public IUReal min(IUReal r);
  public IUReal max(IUReal r);
  public double toReal();
  public int toInteger();
}
```

The first implementation (`UReal`) assumes the quantities follow a Normal distribution and therefore it uses the analytical solutions for the operations. It defines two attributes and offers two type constructors:

```
public class UReal implements IUReal {
  private double x, u;
  public UReal () {x=0.0; u=0.0;}
  public UReal (double nx) {x=nx; u=0.0;}
  public UReal (double nx,double nu){x=nx;u=nu;}
  ...
}
```

The second implementation (`MC_UReal`) uses the Monte Carlo simulation method described in [5] and outlined in Sect. II-C2. Every object of this type contains, in addition to the values x and u, a vector S with the samples obtained according to the probability distribution we want to use. Initially we use 10000 samples for performance reasons, although that number can be increased to 10^6 if further precision required. Attribute x will always coincide with the mean of vector S, and u with its standard deviation. `MC_UReal` constructors currently support the most commonly used probability distributions in this domain (Uniform, Triangular, Truncated, Normal and U-shaped) for initially creating `MC_UReal` samples. Others can be easily implemented if required using standard statistical methods.

```
public class MC_UReal extends UReal {
  private final int MAXLENGTH = 100000;
  private double[] sample=new double[MAXLENGTH];

  public MC_UReal(){
    //no params, sample initialised to 0
    x = 0.0; u = 0.0; Arrays.fill(sample, 0);
  }
  public MC_UReal(double x){
    //"promotes" a real x to (x,0).
    this.x = x; u = 0.0;
    Arrays.fill(sample, x);
  }
  //Uniform distribution if no distribution given
  public MC_UReal(double x, double u) {
    this.x = x; this.u = u;
    fillSample(sample,x,u,Distribution.UNIFORM);
    //fillSample generates the samples for the
    //given distribution
  }
    public MC_UReal(double x, double u,
                 Distribution dist) {
```

```
    this.x = x; this.u = u;
    fillSample(sample,x,u,dist);
  }
  ...
}
```

The operations on elements of this type are executed on the samples first, and then the mean and standard deviation of the resulting vector are computed and assigned to x and u attributes.

For example, to compute the sum $Z = (z, u_z, S)$ of two `MC_UReal` numbers $X = (x, u_x, S')$ and $Y = (y, u_y, S'')$, we first compute the samples for Z as $S_i = S'_i + S''_i$, and then the resulting z and u_z are calculated as the mean and standard deviation of S. In this way we are able to deal with quantities independently from the distributions of their sample measurements. Similarly for the rest of the operations on uncertain values.

One of the advantages of this approach of separating the interface from the implementation(s) is that we also permit alternative implementations that make use of any of the software libraries and packages already available for propagating uncertainty, such as the ones listed in [11].

Both implementations available from our project web page: http://atenea.lcc.uma.es/index.php/Main_Page/Resources/DataUncertainty.

C. More accurate comparisons with Extended Uncertainty

Another interesting possibility is to define a more accurate specification of the `lessThan` ("<") comparison operation on uncertain real numbers. The one we defined in the previous section required the numbers to be distinct and one less than the other. To be different, the intervals $[x_1 - u_1, x_1 + u_1]$ and $[x_2 - u_2, x_2 + u_2]$ cannot overlap (remember that each of these roughly contain the 68.3% of the possible values of X_1 and X_2).

Imagine that we need even less tolerance and require that in order to be distinct, the intervals that contain 95% of the possible values of each variable cannot overlap. This is when Extended Uncertainty [5] comes into play. Assuming that the variability of both variables follow a Normal distribution, we know we can use a *coverage factor* $k = 2$ that would double the width of the interval to include 95.4% of the values, or $k = 3$ to cover 99.7% ("three sigma") of all possible values. More generally, with our approach we can define the following comparison operation:

```
context UReal::lessThanE(r:UReal,k:Real):Boolean
    post: result = (self.x < r.x)
                and ((self.x+k*self.u)<(r.x-k*r.u))
```

IV. Applications

A. Fitting parts

As a first example of application of our proposal, suppose we are dealing with objects whose bases are rectangles, and we need to model a robot arm that performs a set of operations on them including translations, rotations, etc. (an interesting example of this kind of systems in the Industry 4.0 context is described in [19]).

One example of the operations of interest, `fitsIn()`, decides whether a rectangle object fits in a place defined by another rectangle (the base). Such an operation can be specified in OCL as follows.

```
context Rectangle::fitsIn(base:Rectangle):Boolean
post: result = base.x.lessEq(self.x) and
    self.x.add(self.w).lessEq(base.x.add(base.w))
    and
    base.y.lessEq(self.y)
    and
    self.y.add(self.h).lessEq(base.y.add(base.h))
```

Note that its specification is exactly the same that we would produce when exact numbers are used, but now we are taking into consideration the standard uncertainty of all values simply by using `UReal` values instead of `Real` in the specification of a rectangle.

The result provided by this operation is a `Boolean` value that indicates whether the piece fits or not, with a confidence level of 68.3% in case the measurements are distributed according to a Gaussian distribution. As previously mentioned, it would be possible to strengthen the equality test by increasing the interval to 2 or 3 times the standard deviation, using the *Extended Uncertainty* with a *coverage factor* of $k = 2$ or $k = 3$ that would account for, respectively, 95.4% or 99.7% of the possible values of these quantities (despite being their probability of occurrence relatively small).

Let us remark here that, instead of returning a `Boolean` value, an alterative approach would have used a three-valued logic or even a fuzzy logic for specifying these kinds of operations. However, these would require an extension of the OCL logic, something which would have a stronger impact on the extension of the ULM an OCL types. Our extension is less disruptive, and still users can perfectly decide on the accepted level of "risk" (defined by the confidence level) they are happy to take when deciding whether a piece would fit in another or not.

B. Moving objects around

More interesting are those operations that move objects around. In a realistic setting they need to be lifted first, then moved and finally dropped in the target position. The problem is that all operations increase the uncertainty of the position of the object because normally all machines are not free from tolerance. For example, operation `pickUp()` on a `Rectangle` should not change its coordinates, at least in theory, but we all know that any moving arm that lifts an object has some tolerance. Likewise for the `drop()` operation. Thus, simply lifting an object and then dropping it may slightly change the coordinates of its position.

With our proposal, it is possible to take into account these possible deviations in a natural manner, since they are inherent to the type system used. For example, suppose that $u_{px}(h)$ and $u_{py}(h)$ determine the possible deviation introduced by operation `pickUp(h)` when the object is picked up and lifted h millimeters. Similarly, suppose that $u_{\mathtt{dx}}(h)$ and $u_{\mathtt{dy}}(h)$ represent the changes in the coordinates of the object when dropping it from a hight h. Then, if $M = (m_x \pm u_{\mathtt{mx}}(h), m_y \pm u_{\mathtt{mx}}(h))$ represents the movement along the two axis, the final coordinates (X', Y') of a rectangle initially in position (X, Y) after it has been lifted h units, moved and dropped to the floor again are the following:

$$X' = \left(x + m_x, \sqrt{u_x^2 + u_{\mathtt{px}}^2(h) + u_{\mathtt{mx}}^2(h) + u_{\mathtt{dx}}^2(h)}\right) \quad (1)$$

$$Y' = \left(y + m_y, \sqrt{u_y^2 + u_{\mathtt{py}}^2(h) + u_{\mathtt{my}}^2(h) + u_{\mathtt{dy}}^2(h)}\right) \quad (2)$$

In MDE, such objects movements are normally modelled by endogenous model transformations, using e.g. graph transformations rules that determine when an object should be moved and the effect of such a movement. By means of type `UReal` and its related operations, all aspects related to the uncertainty of the positions of objects can be incorporated into the high-level models and handled in a natural way (since all the operations that handle the uncertainty of the quantities are taken care of by the equations defined in Section III).

Thus, imagine that we start with an object in position $(0, 0)$ and we lift it, move it to position $(1, 1)$ and then drop it. Let us suppose that the tolerance of all these movements across any dimension is 0.05 millimeters. By using uncertainty in our variables, we can conclude that the coordinates of the final position of the rectangle will be $(1 \pm 0.070710678, 1 \pm 0.070710678)$, which is a more accurate expression of the result.

Something that should also be considered is the standard uncertainty introduced by the repetition of the lifting and dropping operations. As we have seen, every time we move an object we are increasing the uncertainty of its final position, due to the *propagation of uncertainty*. For example, suppose that we start with an object in a well-defined position given by coordinates (X_0, Y_0) with X_0 and Y_0 two `UReal` numbers defined by $X_0 = (x, 0)$ and $Y_0 = (y, 0)$. Suppose that we lift and drop it (without moving it) using a machine with tolerance t for each movement. Then, after the first move it will end in position (X_1, Y_1) with $X_1 = x \pm t\sqrt{2}$ and $Y_1 = y \pm t\sqrt{2}$. After the second lift-drop operation, the position is (X_2, Y_2) with $X_2 = x \pm t\sqrt{4}$ and $Y_2 = y \pm t\sqrt{4}$. Using formulae (1) and (2) above we can easily see that after lifting and dropping it n times, the final position will be (X_n, Y_n) with $X_n = x \pm t\sqrt{2n}$ and $Y_n = y \pm t\sqrt{2n}$. In particular, if $X_0 = (0, 0)$ and $Y_0 = (0, 0)$, and $t = 0.05$ millimeters, after only 50 operations the accumulated uncertainty will be of 0.5 millimeters, and after 200 operations of 1 millimeter—a divergence that cannot be neglected.

To address this issue, one solution is to physically recalculate the position of the object when the dispersion of the values of its coordinates (given by the associated standard uncertainty) goes above some predefined threshold. In this way we will be able to limit the deviation of the measurements.

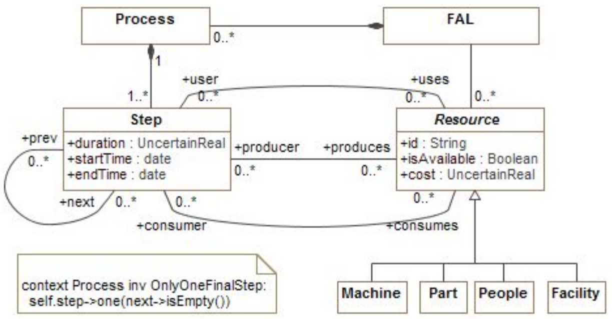

Figure 2. A simple metamodel for specifying processes.

C. Process planning with uncertainty

Going back to Example 2 in Section II, suppose that the assembly process of our factory conforms to the metamodel depicted in Fig. 2. We are interested in calculating the cost and duration of such processes, based on the information given in the model.

These operations can be specified in general by the following two OCL operations, `cost()` and `duration()`. The first one adds the cost of all input and consumed resources by the steps of a project. In order to compute the total duration of the process we need to calculate the duration of its longest branch, since steps can also be carried out in parallel. Operation `duration()` starts by the final step of the project (the one with no subsequent steps) and recursively computes the maximum cost of all its preceding steps. It assumes there is no contention due to unavailability of resources or other reasons.

```
context Process::cost() : UReal
post: result = self.step->
       iterate(t:Step; c:Bag(UReal) = Bag{}|
        c.including(t.uses.cost->sum() +
            t.consumes.cost->sum()))->sum()

context Process::duration() : UReal
post: result=self.step->select(next->
   isEmpty())->collect(t|t.accDuration())->max()

context Step::accDuration() : UReal
post: result = self.duration.add(self.prev->
       iterate(t:Step; d:Bag(UReal) = Bag{} |
          d.including(t.accDuration()))->max() )
```

The interesting aspect of our proposal is that the only change required to deal with uncertainty is the use of type `UReal` in the variables. All algorithms and calculations are exactly the same.

D. Accumulated errors and numerical stability

The algebra of operations we have defined takes care of calculating the uncertainty of the resulting quantities when they are aggregated by operations. Our next concern is about the potential excessive growth of accumulated uncertainty due to error propagation through the operations, i.e., the *stability* of the measures under the presence of accumulated uncertainty.

In our first example, we already discussed that every time we move an object we are increasing the uncertainty of its final position, due to the *propagation of uncertainty* by the sequence of lifting and dropping operations that accumulate errors. Its growth is determined by the uncertainty introduced by the movements, and we were able to compute it in Sect. IV-B. In our second example, the uncertainty of the total cost $C = c \pm u_c$ of the process is augmented by approximately the square of the sum of the standard uncertainties of the costs of all resources used (UR) or consumed (CR) during the process:

$$u_c = \sqrt{\sum_{i=1}^{UR} (\texttt{uses}_i\texttt{.cost.u})^2 + \sum_{j=1}^{CR} (\texttt{consumes}_j\texttt{.cost.u})^2} \tag{3}$$

Even for the small processes, the value shown in formula (3) may represent a rather significant inflation of the uncertainty of the final cost of the resulting product. And similarly for the duration of the complete process.

This leads us to the problem of studying the *stability* of the measures under the presence of accumulated uncertainty. Numerical stability is a generally desirable property of numerical algorithms, which is concerned with the growth of round-off errors and/or initially small fluctuations in initial data which might cause a large deviation of final answer from the exact solution. Calculations that can be proven not to magnify approximation errors are called *numerically stable* [20].

In our case, the relative error can be defined by $\eta(x) = |u|/|x|$ and what we need to prove is that $\eta(x) \leq \epsilon$ for a given small and constant ϵ (in floating point arithmetics, ϵ coincides with the unit round-off, or machine-epsilon).

The good news is that numerical analysis says that the basic operations we have defined on real numbers are *backward stable*, which means that they deliver the exactly correct results for inputs that are slightly perturbed from their correct values in a *relative* sense. More precisely, if the relative error is defined as $\eta(x) = |u|/|x|$, and X and Y are two quantities such that $\eta(x) \leq \epsilon$ and $\eta(y) \leq \epsilon$ for a given small and constant ϵ, and $\otimes$ is an operation from the set $\{+, -, \times, /\}$, then it can be proven that $\eta(x \otimes y) \leq \epsilon$. The same conclusion holds for other elementary operations such as sqrt, exp or sin [21]. Thus, an explosion of the *relative error* of our calculations due to the propagation of uncertainty cannot be expected.

This does not imply, however, that the *absolute error* does not grow. It will certainly build up, but it will always maintain a growth level proportionate to the uncertainty of the aggregated measures. In any case, we need to be careful if we need to maintain the absolute error (i.e. $|x + u|$) below a certain threshold. This is precisely what we saw in Sect. IV-B when an object was repeatedly lifted and dropped, and whereby the uncertainty in the coordinates of the resulting positions grew significantly and could not be neglected. Similarly, in the FAL processes situation, it is easy to see that if there is a fixed set of N resources involved in a process, $\{r_1, \ldots, r_N\}$, the aggregated uncertainty associated to the cost of the total process is upperly bounded by $N \times \max\{u(r_1.c), \ldots, u(r_N.c)\}$.

V. Related Work

The idea behind this proposal was initially triggered by Guesstimate [22], a spreadsheet that permits expressing uncertainty in the cell values, and performs calculations with them. Guesstimate follows a simulation approach for computing the propagation of uncertainties, with sample sizes of 5000 values. At the same time, we were working on the representation of some cyber-physical systems using high-level software models and realized about the importance of considering uncertainty in some of the attributes of the represented elements.

There are various modelling works that deal with uncertainty, but they usually focus on aspects of uncertainty different from the ones we have described here. For instance, on the uncertainty on the models themselves and on the best models to use depending on the system properties that we want to capture [23]. Other works deal with the uncertainty of the design decisions, of the modelling process, or of the domain being modeled [24], [25], [26], [27]. Our work differs from them since we are just concerned with the uncertainty of the values of the quantities being measured and represented in the attributes of our software models, which is a different problem.

Other authors have also identified the need of counting on mechanisms to represent and manipulate physical values in software models [3], in particular units or real-time properties. For example, some works on Business Process Models (e.g., [10]) and even some modelling languages also consider uncertainty when modelling the arrival time of clients, the availability of some resources or the duration of some tasks. These works use probabilistic mass functions for modelling the values of the corresponding attributes, instead of fixed values. We have preferred to use the way defined by the GUM [5], [6].

Similarly, the definition and management of uncertainty in measurements is widespread in other domains like real-time systems where, indeed, timing values are by nature uncertain (they are very often estimates and/or measured by means of monitoring). The real-time community is used to exploit probability distributions and intervals for timing properties, and their influence is clear in the MARTE UML Profile [8]. Such a modeling language defines *precision* as one of the qualifier attributes required to specify and qualify *NFP* values. Precision is defined as "a real number that determines the degree of refinement in the performance of a measurement operation, or the degree of perfection in the instruments and methods used to obtain a result. It is characterized in terms of a Real value, which is the standard deviation of the measurement." Clearly, it corresponds to the concept of measurement uncertainty defined in the GUM and used in this paper. However, MARTE does not offer any algebra of operations for making calculations with these stereotyped values. This lack of a neat integration with the type system hinders its usability and easy of use when having to define derived attributes or to perform computations that deal with uncertainty in OCL. In this respect, our work could be used to complement the MARTE standard with a set of operations for elements stereotyped with a non-null precision.

Anyway, both approaches are equivalent and therefore a bijective transformation can be easily established between them to transform one representation into the other, and viceversa, depending on the designer's choice.

Note as well that we could have also proposed an alternative representation for expressing uncertain quantities in UML, by means of defining a new stereotype `<<uncertain>>` that is attached to attributes of type `Real`. Such a stereotype will need just one tagged value (`u:Real`) to represent the associated standard uncertainty. This is a similar approach to the one followed in the MARTE UML Profile. The main advantage of this approach is that it is less intrusive, since this stereotype can serve to decorate existing UML models indicating the attributes that should be considered as uncertain quantities. However, as in the MARTE approach, the use of stereotypes significantly complicates the specification of OCL expressions and invariants over the model elements.

Finally, the work in [28] defines an XML-based modelling language for measurement uncertainty evaluation based on the GUM, and a simulation framework for it. This work can be in principle considered closely related to our proposal, but the fact that it is not integrated with the type system of a mainstream modelling language (such as UML or OCL), and its low-level syntax (based on plain XML) hindered its usability. Similarly, the work in [29] defines a data type that incorporates measurement uncertainty and provides some libraries to perform computations with its values. The integration of these works with UML/OCL models is not straightforward, and therefore their adoption and usage by UML modelers might be limited. These works seem to be more closely related to the mathematical libraries and tools already existing [11] for propagating measurement uncertainty and operating with uncertain values, than to the work we propose here.

VI. Conclusion and Future Work

In this paper we have focused on representing and managing measurement uncertainty in UML and OCL software models, something required in order to precisely capture and manipulate some of the essential quality properties of any physical system. We have introduced a new UML and OCL type and the set of related operations to perform computations with its values. OCL and Java libraries have also been developed to implement the type and its operations in MDE settings.

This work opens several interesting lines of research that we would like to explore next. First, uncertainty can go beyond the values of attributes. In particular, we plan to extend our study to represent and operate with uncertainty in the context of further modelling elements, including decisions made in branches and selections, classifiers, and other behavioural elements. Second, we are currently integrating our proposal into existing modelling tools, namely Papyrus, MagicDraw and USE [30]. In particular, its connection with fUML [31], [32] represents a natural step forward. Similarly, the integration of this type and its operation into the simulation and analysis tools that we use to reason about the behaviour of the systems and their properties represents an interesting challenge. Likewise, the connection of our types with existing mathematical tools for dealing with measurement uncertainty could provide more powerful computing capabilities to our approach in case they are needed. Finally, larger case studies should give more feedback about the expressiveness and applicability of the proposal.

Acknowledgements. We would like to thank the reviewers for the useful comments and suggestions on previous versions of this paper. This work is funded by Spanish Research Project TIN2014-52034-R and by Universidad de Málaga (Campus de Excelencia Internacional Andalucía Tech).

References

[1] E. A. Lee, "Cyber Physical Systems: Design Challenges," in *Proc. of ISORC'08.* IEEE, 2008, pp. 363–369.

[2] M. Broy, "Challenges in modeling Cyber-Physical Systems," in *Proc. of ISPN'13.* IEEE, 2013, pp. 5–6.

[3] B. Selic, "Beyond Mere Logic – A Vision of Modeling Languages for the 21st Century," in *Proc. of MODELSWARD 2015 and PECCS 2015.* SciTePress, 2015, pp. IS–5. [Online]. Available: http://cescit2015.um.si/Presentations/KN_Selic.pdf

[4] P. J. Mosterman and J. Zander, "Cyber-physical systems challenges: a needs analysis for collaborating embedded software systems," *Software and System Modeling*, vol. 15, no. 1, pp. 5–16, 2016.

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

[6] JCGM 101:2008, *Evaluation of measurement data – Supplement 1 to the "Guide to the expression of uncertainty in measurement" – Propagation of distributions using a Monte Carlo method*, Joint Committee for Guides in Metrology, 2008, http://www.bipm.org/utils/common/documents/jcgm/JCGM_101_2008_E.pdf.

[7] JCGM 200:2012, *International Vocabulary of Metrology – Basic and general concepts and associated terms (VIM), 3rd edition*, Joint Committee for Guides in Metrology, 2012, http://www.bipm.org/utils/common/documents/jcgm/JCGM_200_2012.pdf.

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

[9] ——, *OMG Systems Modeling Language (SysML), version 1.4*, Jan. 2016, OMG Document formal/2016-01-05.

[10] A. Jiménez-Ramírez, B. Weber, I. Barba, and C. del Valle, "Generating optimized configurable business process models in scenarios subject to uncertainty," *Information & Software Technology*, vol. 57, pp. 571–594, 2015.

[11] Wikipedia, "List of uncertainty propagation software," last accessed June 21, 2016, https://en.wikipedia.org/wiki/List_of_uncertainty_propagation_software.

[12] Object Management Group, *Unified Modeling Language (UML) Specification. Version 2.5*, Mar. 2015, oMG Document formal/2015-03-01.

[13] ——, *Object Constraint Language (OCL) Specification. Version 2.4*, Feb. 2014, oMG Document formal/2014-02-03.

[14] F. Jouault, F. Allilaire, J. Bézivin, and I. Kurtev, "ATL: A model transformation tool," *Science of Computer Programming*, vol. 72, no. 1-2, pp. 31–39, 2008.

[15] J. Cabot, J.-N. Mazón, J. Pardillo, and J. Trujillo, *Proc. of ER 2010*, ser. LNCS. Springer, 2010, vol. 6412, ch. Specifying Aggregation Functions in Multidimensional Models with OCL, pp. 419–432. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-16373-9_30

[16] M. Gogolla, "Benefits and problems of formal methods," in *Proc. of Ada-Europe 2004*, ser. LNCS, vol. 3063. Springer, 2004, pp. 1–15.

[17] Instructional Physics Laboratory, Harvard University, "A summary of error propagation," 2014, http://ipl.physics.harvard.edu/wp-uploads/2014/01/ps2_fa13_err.pdf.

[18] K. Angoni, "Uncertainties in measurement," 2015, http://gauss.vaniercollege.qc.ca/pwiki/index.php/Uncertainties_in_Measurement.

[19] P. J. Mosterman and J. Zander, "Industry 4.0 as a cyber-physical system study," *Software and System Modeling*, vol. 15, no. 1, pp. 17–29, 2016.

[20] N. J. Higham, *Accuracy and Stability of Numerical Algorithms.* Society of Industrial and Applied Mathematics, 1996.

[21] J.-M. Muller *et al.*, *Handbook of Floating-Point Arithmetic.* Birkhäuser, 2010.

[22] O. Gooen, "The Guesstimate Blog," Dec. 2015, https://medium.com/guesstimate-blog.

[23] B. Littlewood, M. Neil, and G. Ostrolenk, "The role of models in managing the uncertainty of software-intensive systems," *Reliability Engineering & System Safety*, vol. 50, no. 1, pp. 87 – 95, 1995.

[24] D. Garlan, "Software Engineering in an Uncertain World," in *Proc. of the FSE/SDP Workshop on Future of Software Engineering Research (FoSER'10).* ACM, 2010, pp. 125–128.

[25] M. Famelis, R. Salay, and M. Chechik, "Partial models: Towards modeling and reasoning with uncertainty," in *Proceedings of the 34th International Conference on Software Engineering*, ser. ICSE '12. IEEE Press, 2012, pp. 573–583.

[26] N. Esfahani and S. Malek, "Uncertainty in self-adaptive software systems," in *Software Engineering for Self-Adaptive Systems II*, ser. LNCS, R. de Lemos *et al.*, Eds., no. 7475. Springer, 2013, pp. 214–238.

[27] R. Salay, M. Chechik, J. Horkoff, and A. Sandro, "Managing requirements uncertainty with partial models," *Requirements Engineering*, vol. 18, no. 2, pp. 107–128, 2013.

[28] M. Wolf, "A modeling language for measurement uncertainty evaluation," Ph.D. dissertation, ETH Zurich, 2009.

[29] B. D. Hall, "Component interfaces that support measurement uncertainty," *Computer Standards & Interfaces*, vol. 28, no. 3, pp. 306–310, 2006.

[30] M. Gogolla, F. Büttner, and M. Richters, "USE: A UML-based specification environment for validating UML and OCL," *Science of Computer Programming*, vol. 69, pp. 27–34, 2007.

[31] Object Management Group, *Semantics Of A Foundational Subset For Executable UML Models (FUML), version 1.4*, Jun. 2015, OMG Document formal/2015-06-03, http://www.omg.org/spec/FUML/1.2.1/PDF.

[32] T. Mayerhofer, P. Langer, M. Wimmer, and G. Kappel, "xMOF: Executable DSMLs Based on fUML," in *Proc. of SLE 2013*, ser. LNCS, vol. 8225. Springer, 2013, pp. 56–75.