JOT
JOURNAL OF
OBJECT TECHNOLOGY

Extending OCL with Subjective Logic

Paula Muñoz[*], **Lola Burgueño**[†], **Victor Ortiz**[‡], **and Antonio Vallecillo**[*]
[*]ITIS Software, Universidad de Málaga, Spain
[†]IN3, Open University of Catalonia, Spain, and Institut LIST, CEA, Université Paris-Saclay, France
[‡]Minsait, Indra, Málaga, Spain

ABSTRACT The logic of the UML and OCL modeling languages is based on crisp values, e.g., true or false. However, when modeling systems that work in physical environments or where human actors are involved, different users may have subjective opinions about the reality that they perceive, and thus may need to assign different levels of confidence to the logic predicates of the models. These different views, or opinions, may also be subject to uncertainty when there is a lack of knowledge about the system, adding the dimension of ignorance to the traditional belief-disbelief dichotomy. This paper proposes an extension of the OCL/UML datatype Boolean that enables the representation of subjective uncertain opinions, together with a set of logical operators for reasoning with uncertain propositions in order to reach better informed decisions. The proposal has been implemented as an extension of the UML-based Specification Environment (USE) tool, and validated with several applications and case studies.

KEYWORDS OCL, uncertainty modeling, subjective logic.

1. Introduction

Conceptual modeling (Olivé 2007) enables engineers to represent a system at a level of abstraction that contains the relevant information for the purpose of the model, abstracting away irrelevant details. Despite the success of UML (Object Management Group 2015) and OCL (Object Management Group 2014) for modeling information systems, the birth of new paradigms such as the Internet of Things (IoT), Cyber-physical Systems (CPS), the advances in Robotics and Artificial Intelligence (AI), have raised the need to model systems that represent or interact with new environments (i.e., physical factors), which is challenging these notations. In particular, the basic datatypes and mechanisms that they provide are falling short for expressing behavioral aspects which are essential to these kinds of systems such as concurrency, units, precision, uncertainty or social behavior (Lee 2008; Selic 2015; Zhang et al. 2016; Bertoa et al. 2019; Burgueño, Mayerhofer, et al. 2019; Bucchiarone et al. 2020).

JOT reference format:
Paula Muñoz, Lola Burgueño, Victor Ortiz, and Antonio Vallecillo. *Extending OCL with Subjective Logic.* Journal of Object Technology. Vol. 19, No. 3, 2020. Licensed under Attribution 4.0 International (CC BY 4.0)
http://dx.doi.org/10.5381/jot.2020.19.3.a1

Focusing on *uncertainty*, it applies to physical measurements, estimations, predictions of future events, and unknown properties of a system. Uncertainty is defined as "the quality or state that involves imperfect and/or unknown information" (JCGM 2008). In this paper, we are interested in representing one particular kind of uncertainty, namely the uncertainty due to *imprecision and subjectivity of logical predicates*, whereby a user is not sure about a statement, i.e., a Boolean predicate, made about the system.

In UML, logic predicates are represented by OCL expressions of type `Boolean`. This datatype represents the binary logic values `true` and `false`, which fit very well with an ideal view of a perfect world. In many real situations, however, we cannot be completely sure about the truth of these predicates. For example, whether a sensor is working properly or not, or whether your favorite team will win the league this year.

Several extensions to Boolean logic allow dealing with this kind of uncertainty. A particular case is *probability theory* (Feller 2008; de Finetti 2017), which assigns probabilities to propositions, rather than truth values, and where formulas of probability calculus replace truth tables. In other words, they let Boolean values or predicates to be *partially* true. In (Bertoa et al. 2019), we proposed an extension of UML and OCL type `Boolean`, called `UBoolean` (Uncertain Boolean), which adds

to each Boolean value a real number in the range [0,1] that represents the likelihood that such a value is true. This datatype provides a probabilistic extension to binary logic.

However, a fundamental limitation of probabilistic logic is the inability to account for the modeler's level of confidence in the assigned probability, or to handle the situation in which the modeler cannot assign probabilities to a predicate. For example, when the modeler has total ignorance about some statement x, it might be preferable to say "I don't know" than assigning x a confidence of 0.5, because a confidence of 0.5 would mean that x and $\neg x$ are equally likely, which does not represent ignorance since it is already quite informative (Jøsang 2016). Moreover, forcing a modeler to set probabilities with little or no confidence could lead to unreliable conclusions. Therefore, we need a formalism that explicitly states the level of uncertainty that someone holds about a predicate, and distinguishes between the degrees of belief, disbelief and uncertainty, since they are different.

Subjective logic (Jøsang 2001, 2016) is a probabilistic logic that explicitly takes uncertainty and trust into account (Jøsang et al. 2005), and allows different individuals to hold different subjective opinions about the same statement. Expressions in subjective logic are called *opinions*, and are defined by quadruples (b, d, u, a). The b, d and u components represent, respectively, the degrees of belief, disbelief and uncertainty that the agent has about the statement, i.e., the degree of trust; and a is the (objective) prior probability assigned to the statement. For example, $(0.7, 0.1, 0.2, 0.5)$ means that there is an statement with a (prior) probability of 0.5 to be true, but the agent has a degree of belief of 0.7 that the statement to which the quadruple is associated is true, 0.1 that it is false, and is 0.2 uncertain about it. In other words, initially the statement was equally likely to be true or false (hence the prior probability of 0.5), but agent's stakes are 70% in favor, 10% against, and 20% unsure.

In this paper, we propose an extension of the UML and OCL primitive datatype `Boolean`, and of its extension `UBoolean`, which enables the representation of subjective opinions in order to make more reliable and informed decisions about OCL statements (i.e., specific situations and future events). The extended datatype, called `SBoolean`, provides a set of operators that can be used for logical reasoning with uncertain propositions. We present how the former logic types can be naturally embedded into the new type, and how OCL expressions can be seamlessly evaluated in the extended type. The proposal has been implemented as an extension of the UML-based Specification Environment (USE) tool (Gogolla et al. 2007), and validated with several applications and case studies.

This paper is structured as follows: Sect. 2 introduces the concepts that will be used throughout the paper. Then, Sect. 3 describes our proposal: the type `SBoolean` and its values, as well as the algebra of operations on subjective values that we have defined. The algebraic properties of these operations are also studied. Tool support is described in Sect. 4, while Sect. 5 presents the usability evaluation we have performed on the proposal. Finally, Sect. 6 compares our work to similar proposals and Sect. 7 concludes the paper with an outlook on future work.

2. Background

2.1. Belief Uncertainty

Uncertainty can be defined as "the quality or state that involves imperfect and/or unknown information" (JCGM 2008). Various types of uncertainties can be considered when modeling a system (Oberkampf et al. 2002; Thunnissen 2003; Zhang et al. 2016, 2019; Object Management Group 2017). For example, *aleatory* uncertainty refers to the inherent variation associated with the physical system under consideration, or its environment. In contrast, *epistemic* uncertainty refers to the potential inaccuracy or vagueness that is due to the lack of knowledge (Oberkampf et al. 2002).

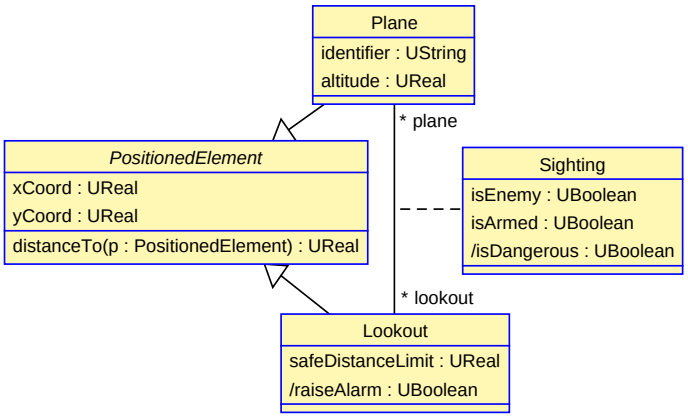
Logic predicates that refer to physical systems should be able to capture uncertainty of both aleatory and epistemic natures. For instance, aleatory uncertainty happens when we compare two uncertain real numbers (e.g., $3.0 \pm 0.1 < 3.1 \pm 0.1$). The result can be expressed by the probability that one is in fact less than the other: in this case, 0.383, cf. (Bertoa et al. 2019). On the other hand, epistemic uncertainty happens, for example, when somebody is asked whether it will rain tomorrow or not.

A particular kind of epistemic uncertainty, called *Belief Uncertainty*, occurs when a user is not sure about the truth of an statement, i.e., a Boolean predicate. This is directly related to trust (Jøsang et al. 2005). Several extensions to the Boolean logic enable dealing with belief uncertainty, including probability theory (Feller 2008; de Finetti 2017), possibility theory (based on fuzzy logic (Zimmermann 2001; Russell & Norvig 2010)), plausibility (a measure in the Dempster-Shafer theory of evidence (Shafer 1976)) and uncertainty theory (Liu 2018). These proposals assign different probabilities to propositions, rather than truth values, and probability formulas replace truth tables.

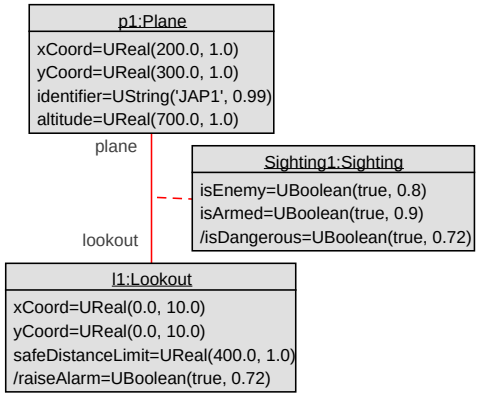
In (Bertoa et al. 2019), we proposed an extension of all UML and OCL primitive datatypes (Boolean, Real, String, Integer) able to deal with uncertainty. Type embedding was used to define the extensions, and subtyping ensured safe replaceability of values and operations. In particular, type `UReal` extends the OCL type `Real` by adding the uncertainty of measurement associated to its values, expressed as the standard deviation of their potential measurements (JCGM 2008). Thus, possible `UReal` values are 3.0 ± 0.1 or 4.5 ± 0.0. This last uncertain real number represents the embedding of the `Real` value 4.5 into `UReal`.

The type `UBoolean` extends the type `Boolean` by adding the probability that expresses the likelihood that the value is `true`. In other words, it allows Boolean values or predicates to be *partially* true. Possible `UBoolean` values are $(true, 0.90)$ or $(false, 0.70)$. The type `Boolean` can be naturally embedded into `UBoolean` by lifting *true* to $(true, 1.0)$ and *false* to $(true, 0.0)$.

However, as we mentioned in the introduction, this probabilistic extension to binary logic presents some limitations when the modeler is uncertain about the probability that she has to assign to a logic predicate or to a Boolean attribute. This uncertainty is typically called *second-order probability* or *second-order uncertainty* in the literature of statistics and economics, and needs to be explicitly represented, propagated and taken



(a) The Lookout model class diagram.



(b) Example of an object diagram.

Figure 1 A lookout system for enemy planes.

into account in the results, in order to make informed decisions about the system. Therefore the need to count on notations that deal with such uncertainty as a first-class concept, and on a type system that extends that of UML and OCL and enables its transparent manipulation and propagation.

2.2. A motivating example

To illustrate our proposal, let us suppose there are a set of lookouts on the Pacific islands during World War II, in charge of monitoring airspace and detecting enemy planes that may pose a threat to the allies. This system can be specified in UML as shown in Fig. 1a. The three main entity types are `Lookout`, `Plane` and `Sighting`. Associated to every sighting of a plane, the lookout system identifies whether the plane is an enemy or not (`isEnemy`), and whether it is armed (`isArmed`). The values of these two variables are determined by, e.g., checking the identifier of the plane against the list of known allies, or whether bombs are visible on the plane wings. These two decisions are difficult in practice due to complicated weather conditions, lack of visibility, lack of operator experience or visual acuity. The third attribute, `isDangerous`, is derived using the expression specified in Listing 1. It checks whether the plane is considered to be an enemy, it is armed, and is inside the safe area defined for the lookout. Note the use of uncertain datatypes in the model, namely `UReal` to represent distances, `UString` to represent identifiers (given that character recognition system may not be fully accurate under poor visibility conditions), and `UBoolean` to represent logical values with associated probabilities. Finally, the attribute `raiseAlarm` of the class `Lookout` decides if the alarm should be raised, with a probability that corresponds to the plane considered as most dangerous.

Figure 1b shows an object diagram with a lookout, a plane and one sighting. Given the confidence of the attributes of the sighting, and the confidence of the comparison between the two `UReal` numbers that determine the distance between the plane and the lookout, the plane is considered to be dangerous with a confidence of 0.72. However, the operator at the lookout post

```
context Sighting::isDangerous : UBoolean derive:
  s.isEnemy and s.isArmed and
  (self.lookout.distanceTo(self.plane) <=
     self.lookout.safeDistanceLimit)
context Lookout::raiseAlarm : UBoolean derive:
  UBoolean(true, self.sighting.isDangerous->
     collect(i|i.confidence())->max())
```

Listing 1 Derivation expression of attributes `isDangerous` and `raiseAlarm`.

may have a different (subjective) opinion based on their trust on the measuring devices and identification instruments and their own skills using them. Therefore, the subjective confidence (i.e., the trust on these `Boolean` values) may be different from the objective confidence (i.e., `UBoolean` value) assigned to them using probabilities.

Here, we are concerned about how to represent such subjective opinions, which qualify the objective probabilities and add uncertainty to them, and how to take them into account when reasoning about the system. For example, to avoid raising the alarm in false or not raising it when we should.

2.3. Subjective logic

Subjective logic, invented by Audun Jøsang (Jøsang 2001, 2016), is a type of probabilistic logic that explicitly takes uncertainty and trust into account. Subjective opinions express beliefs about the truth of propositions under degrees of uncertainty.

Let x be a state value in a binary domain, e.g., a Boolean predicate. A binomial *opinion* about the truth of state value x is the quadruple $\omega_x = (b_x, d_x, u_x, a_x)$ where:

- b_x (*belief mass*) is the degree of belief that x is true.
- d_x (*disbelief mass*) is the degree of belief that x is false.
- u_x (*uncertainty mass*) is the degree of uncertainty about x, i.e., the amount of uncommitted belief.
- a_x (*base rate*) is the prior probability in the absence of belief or disbelief.

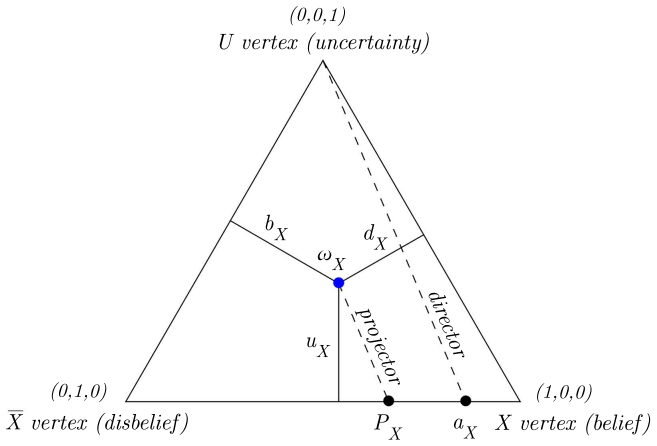


Figure 2 Graphical representation of a binomial opinion (Jøsang 2016).

These values satisfy $b_x + d_x + u_x = 1$, and $b_x, d_x, u_x, a_x \in [0,1]$. Opinions where $b_x = 1$ or $d_x = 1$ are called *absolute* opinions, and are equivalent to the Boolean values *true* and *false*, respectively. An opinion where $b_x + d_x = 1$ is a *dogmatic* opinion which is equivalent to a traditional probability. If $b_x + d_x < 1$, we have an *uncertain* opinion which expresses a degree of uncertainty. Finally, if $b_x + d_x = 0$ (i.e., $u_x = 1$) we have a *vacuous* opinion that expresses total uncertainty, or vacuity of belief.

Opinions can be represented on an equilateral triangle using barycentric coordinates as shown in Fig. 2. A point inside the triangle represents a (b_x, d_x, u_x) triple. Vertices at the bottom represent absolute opinions, and the vertex at the top represents the vacuous opinion ($u_x = 1$). Dogmatic opinions belong to the base line ($u_x = 0$), and correspond to probabilities. The base rate a_x, or prior probability, is shown along the base line, too.

The *projected probability* of an opinion is defined as $P_x = b_x + a_x u_x$. Graphically, it is formed by projecting the opinion ω_x onto the base, parallel to the base rate projector line (i.e., parallel to the *director* line).

Logic operators (and, or, not, implies, equivalent, etc.) are defined for opinions, generalizing those of binary and probabilistic logic. The behaviors of the extended operators respect those of the base types when applied to base values. In case the argument opinions contain degrees of uncertainty, the operators produce derived opinions that always have correct projected probabilities, which ensures a correct subtyping relation between probabilities and opinions.

3. Extending the type Boolean

The main objective of this paper is to extend the OCL and UML languages by declaring a new type, `SBoolean`, that enables the representation and management of binomial opinions of subjective logic. The benefits are twofold: First, opinions can be expressed in software models, enriching the current representation of subjective belief (or trust) on the logic predicates stated in a software model. Second, this information can be managed in a transparent manner by the type system, providing a useful mechanism for reasoning about imprecise knowledge which can be supported by OCL tools.

Similar to the embedding relationship `Boolean` $\hookrightarrow$ `UBoolean` that lifts `Boolean` values to their corresponding probabilities (Bertoa et al. 2019), we also define the embedding relationship `UBoolean` $\hookrightarrow$ `SBoolean` that lifts probabilities to opinions—hence obtaining the complete embedding chain `Boolean` $\hookrightarrow$ `UBoolean` $\hookrightarrow$ `SBoolean`.

3.1. Extension strategy

To extend the datatypes, we apply *type embedding* (Boute 1990), which is one kind of subtyping (Liskov & Wing 1994). Note that embedding, subtyping and inheritance are different concepts (Clerici & Orejas 1988). In broad terms, *inheritance* among classes represents that objects of the subclass inherit the internal structure and code of the superclass and, in addition, can have new features (attributes, methods, relationships, etc.).

In contrast, *subtyping* refers to the part of the objects' behavior that can be observed from the outside (America 1991), namely, the operations that are applied to them. In algebraic terms, subtyping leads to a conceptual hierarchy that is based on behavioral specification. Then, we say that type A is a *subtype* of type B (denoted as $A <: B$) if all elements of A belong to B and the operations of B, when applied to elements of A, behave the same as those of A (America 1987). For instance, Integer is a subtype of Real because an Integer number can be viewed as a Real number whose decimal part is zero. Moreover, operations that are defined on the type Real, when applied to numbers of the type Integer, behave as those operations of type Integer. As stated in (Bertoa et al. 2019), the subtyping relation cannot be directly applied between all the OCL and UML types. For example, one might think that the type `Boolean` could be viewed as a subtype of the type `Integer`, but this would imply the definition of its values as $\{0,1\}$ instead of $\{false, true\}$.

Type *embedding* permits the definition of such relationship by specifying the corresponding (*injection*) isomorphism $\{false, true\} \leftrightarrow \{0,1\}$ and then using subtyping, which results in many useful mathematical properties. For example, Boolean values were embedded into probabilities in (Bertoa et al. 2019) by considering the corresponding isomorphism $\{false, true\} \leftrightarrow \{0.0, 1.0\}$ and by replacing truth tables by probabilistic calculus. With respect to the behavior of the operations, the fact that $\leftrightarrow$ is an isomorphism and that $<:$ is a subtyping relation, ensures that the behavior of the operations of the embedded type is respected when *lifted* to the embedding supertype.

3.2. Type `SBoolean`

The new type `SBoolean`, which extends types `Boolean` and `UBoolean`, is defined by a quadruple (b, d, u, a), where all the components are in the range [0,1], and $b + d + u = 1$. Values of type `SBoolean` represent opinions in subjective logic.

The embedding of a `UBoolean` value $x = (true, c)$ into type `SBoolean` is achieved by assigning the opinion $w_x = (c, 1 - c, 0, c)$ to x. Considering the embedding of type `Boolean` into `UBoolean`, we have that Boolean values `true` and `false` correspond, respectively, to opinions $(1,0,0,1)$ and $(0,1,0,0)$.

```
not() : SBoolean
    post: (result.b = self.d) and (result.d = self.b) and (result.u = self.u) and (result.a = 1-self.a)
and(s : SBoolean) : SBoolean
    post: result.b = self.b*s.b +  if (self.a = 1 and s.a = 1)  then 0 -- to avoid that (1-self.a*s.a) = 0
            else ((1-self.a)*s.a*self.b*s.u+self.a*(1-s.a)*self.u*s.b)/(1-self.a*s.a)) endif and
        result.d = (self.d + s.d- self.d*s.d) and
        result.u = (self.u*s.u + if (self.a = 1 and s.a = 1) then 0 -- to avoid that (1-self.a*s.a) = 0
            else ((1-s.a)*self.b*s.u+(1-self.a)*self.u*s.b)/(1-self.a*s.a)) endif and
        result.a = self.a*s.a
or(s : SBoolean) : SBoolean
    post: result.b = (self.b + s.b - self.b*s.b) and
        result.d = self.d*s.d + if (self.a=0 and s.a=0) then 0 -- to avoid that self.a+s.a-self.a*s.a = 0
            else (self.a*(1-s.a)*self.d*s.u+s.a*(1-self.a)*self.u*s.d)/(self.a+s.a-self.a*s.a)) endif and
        result.u = self.u*s.u + if (self.a=0 and s.a=0) then 0 -- to avoid that self.a+s.a-self.a*s.a = 0
            else (s.a*self.d*s.u+self.a*self.u*s.d)/(self.a + s.a - self.a*s.a)) endif and
        result.a =  self.a + s.a - self.a*s.a
implies(s : SBoolean) : SBoolean
    post: result = self.not().or(s)
xor(s : SBoolean) : SBoolean
    post: result.b = (self.b - s.b).abs() and result.d = 1.0 - (self.b - s.b).abs() - self.u*s.u and
          result.u = self.u*s.u and result.a = (self.a - s.a).abs()
equivalent(s : SBoolean) : SBoolean
    post: result = self.xor(s).not()
equals(s : SBoolean) : Boolean
    post: result = (self.b=s.b) and (self.d=s.d) and (self.u=s.u) and (self.a=s.a)
distinct(s : SBoolean) : Boolean
    post: result = not (self.equals(s))
projection() : Real  -- returns the projected probability of the opinion
    post: result = self.b + self.a*self.u
toUBoolean() : UBoolean  -- conversion to UBoolean
    post: (result.b) and (result.c = self.projection())
```

Listing 2 OCL specification of `SBoolean` operations.

The operations supported by type `SBoolean` extend those of type `UBoolean`, as defined in (Bertoa et al. 2019). We have defined the basic operations (`not`, `and`, and `or`) and secondary operations (`implies`, `equivalent`, and `xor`) of the traditional Boolean algebra by extending them to subjective logic. Assuming that all values are independent, Listing 2 specifies the `SBoolean`-type operations.

We have preserved the semantics of the operations `equals()` and `distinct()`: two `SBoolean` values are the same if and only if their quadruples match (both operations return a `Boolean` value). Other comparison operations, namely `equivalent()` and `xor()`, compare two `SBoolean` values and return another `SBoolean` value.

Finally, a conversion operation, `toUBoolean()`, allows `SBoolean` values to be converted into `UBoolean` by using the *projected probability* defined for subjective opinions, using the operation `projection()`, which projects the opinion into the corresponding probability at the base of the triangle (see Fig. 2).

3.3. Algebraic properties of `SBoolean` operations

The operations defined for type `SBoolean` are generalisations of binary logic and probability operations, as discussed in (Jøsang 2016). In case the opinions acting as operands of the operations are equivalent to Boolean values *true* or *false*, the result of any subjective logic operation is always equal to that of the corresponding propositional/binary logic operation. Similarly, when the opinions are equivalent to traditional probabilities, the result of any subjective logic operator is always equal to that of the corresponding probability operator.

In case the operand opinions contain degrees of uncertainty, the operations will produce derived opinions that always have correct projected probabilities. However, not all the algebraic properties of `SBoolean` operations can be ensured when dealing with values that have some associated uncertainty.

In particular, due to the way in which we have defined the operations, it is easy to prove that the following properties hold for every pair of opinions A and B of type `SBoolean` (operator "$\doteq$" corresponds to the operation `equals()` of type `SBoolean`, see above).

- `not(not(A))` $\doteq$ `A`
- `not(A or B)` $\doteq$ `not(A) and not(B)` (AND Morgan's Law)
- `not(A and B)` $\doteq$ `not(A) or not(B)` (OR Morgan's Law)
- Operation `and` is commutative, associative and its identity element is $(1,0,0,1)$.
- Operation `or` is commutative, associative and its identity element is $(0,1,0,0)$.

The secondary operations (`xor`, `implies` and `equivalent`) of type `SBoolean` work with `Boolean` and `UBoolean` values as before and respect their properties, even when lifted to `SBoolean` values. In particular:

- Operation `implies` is non-commutative and associative, since `(A implies B)` $\doteq$ `(not A or B)`.
- Operation `equivalent` is commutative and associative.
- Similarly, `xor` is commutative and associative.

Special care must be taken when dealing with dependent opinions, because the expressions above assume independence. To deal with this, in case A and B are dependent, we should use A and $B|A$ instead (or, equivalently, $A|B$ and B).

In addition, different logic formulas that are traditionally equivalent in propositional logic do not necessarily have equal

```
uncertaintyMaximized() : SBoolean
-- returns the corresponding SBoolean with max. uncertainty
  post: let p = self.projection() in
    if (self.a=1 and (p=1 or self.u=1)) or
       (self.a=0 and self.b=0) then
    result.b = 0 and result.d = 0 and
    result.u = 1 and result.a = self.a
  else if (p < self.a) then
    result.b = 0 and result.d = 1 - (p/self.a) and
    result.u = p/self.a and result.a=self.a
  else
    result.b = (p-self.a)/(1.0-self.a) and result.d = 0.0 and
    result.u = (1.0-p)/(1.0-self.a) and result.a = self.a
  endif endif endif
```

Listing 4 Specification of `uncertaintyMaximized()`.

opinions. For example, the distributivity of conjunction over disjunction, expressed as $x \wedge (y \vee z) \Leftrightarrow (x \wedge y) \vee (x \wedge z)$, does not hold with opinions. In general, all operations that propagate non-null uncertainty may produce slightly different results. Therefore, in this context, formulas that involve `SBoolean` values should be algebraically simplified, if possible, before the final results are computed to avoid both the unnecessary propagation of uncertainty and to respect the independence of the expression variables. For instance, although the formulas $(A \wedge B) \vee (B \wedge A)$ and $A \wedge B$ are equivalent, we should evaluate the latter.

3.4. Further operations

3.4.1. Conflicting opinions A fundamental assumption behind subjective logic is that different individuals may hold different opinions about the same statement. The *degree of conflict* is a measure of the difference between opinions. It is defined as the product of other two measures between opinions: the *projected distance* (difference between the projected probabilities of the opinions) and the *conjunctive certainty* (common certainty of the opinions). Then, given two opinions w_x^A and w_x^B, we can define (Jøsang 2016):

```
projectiveDistance(s:SBoolean) : Real =
    (self.projection() - s.projection()).abs()/2
conjunctiveCertainty(s:SBoolean) : Real =
    (1.0-self.u)*(1-s.u)
degreeOfConflict(s:SBoolean) : Real =
    self.projectiveDistance(s)*self.conjunctiveCertainty(s)
```

Listing 3 Operations for defining conflicting opinions.

3.4.2. Uncertainty maximized An interesting operation is `uncertaintyMaximized()`, which computes the opinion with the same projected probability but with maximum uncertainty. Graphically, it corresponds to the opinion Ω_x that is obtained by projecting the original opinion ω_x upwards along the projector line, parallel to the base rate director line. Therefore, the projected probabilities of ω_x and Ω_x are the same.

The specification of the `uncertaintyMaximized()` operation is shown in Listing 4.

We can define equivalence classes using the `projection()` operation, whereby two opinions belong to the same class if their projections are the same. Given one opinion "`X`", the opinion with maximum uncertainty of the equivalence class of `X` is `X.uncertaintyMaximized()`, and the one with less degree of uncertainty is `SBoolean(X.projection(), 1-X.projection(),0,X.baseRate())`. Graphically, they correspond to the intersection points of the projector line in the uncertainty triangle. The points in the segment of the projector line inside the triangle belong to the same equivalent class, because they all have the same projected probability.

3.5. Collections of `SBoolean`

OCL defines an abstract datatype `Collection`, with a set of operations common to all kinds of collections, plus a set of operations which are specific to each subtype: `Set`, `OrderedSet`, `Bag` and `Sequence`. We also need to extend these operations to deal with `SBoolean` values.

As described in (Bertoa et al. 2019), when extending collections with uncertain values, they are evaluated in the higher type of the type hierarchy of the elements of the collection. In particular, when a `Sequence` is composed of values of types `Boolean`, `UBoolean` and `SBoolean`, the type of the collection would be `Sequence(SBoolean)` and the corresponding operations will be evaluated in this type.

Logic predicates in collection operations that in the original OCL/UML specification return `Boolean` values — such as `forAll` and `exists` — may now be of type `SBoolean`, and therefore the operations may also return a `SBoolean` value.

As we did for `UBoolean`, we do not allow logic predicates of type `SBoolean` to act as filters to select elements from collections, since in order to do this, we clearly need to decide whether an element belongs or not to a collection; this is the case, for instance, of operations such as `select`, `any`, or `collect`. The operation `projection()`, which allows us to know the probability of a `SBoolean` value, is used to map `SBoolean` values into `Boolean`. This is analogous to the use of the `confidence()` operation to convert `UBoolean` values to `Boolean`, using a threshold value above which the uncertain Boolean is considered to be `true`.

3.6. The Lookout system revisited

Let us go back to the Lookout example described in Sect. 2.2, where attributes `isEnemy` and `isArmed` had some associated probabilities, e.g., 0.8 and 0.9, respectively. Columns 1 and 2 of Table 1 show different values for the subjective opinions by the operator of the lookout about these attributes. Note that we have maintained the (*objective*) probabilities of attributes `isEnemy` and `isArmed` as their base rates (i.e., their prior probabilities). Column 3 shows the result of the derived expression that calculated the value of attribute `isDangerous`, which is now a *subjective* opinion, i.e., a `SBoolean` value. The last column shows the projected probability for that resulting opinion.

The first row of the table shows the situation in which the operator simply respects the initial probabilities, and therefore the projected probability is the same as before (i.e., 0.72). However, when changing the operator's degrees of belief, disbelief and uncertainty, the results also change. This has a significant influence on the projected probabilities of the results, which now range between 0.630 and 0.970. This means that now we are able to take into account the level of trust that the operator has on the prior probabilities assigned to the two variables

Table 1 Results for `isDangerous` depending on various values of `isEnemy` and `isArmed`.

`isEnemy`	`isArmed`	`isDangerous`	**Proj.**
(0.80, 0.20, 0.00, 0.80)	(0.90, 0.10, 0.00, 0.90)	(0.720, 0.280, 0.000, 0.720)	**0.720**
(0.80, 0.10, 0.10, 0.80)	(0.90, 0.10, 0.00, 0.90)	(0.746, 0.190, 0.064, 0.720)	0.792
(0.80, 0.00, 0.20, 0.80)	(0.90, 0.10, 0.00, 0.90)	(0.771, 0.100, 0.129, 0.720)	0.864
(0.90, 0.10, 0.00, 0.80)	(0.90, 0.10, 0.00, 0.90)	(0.810, 0.190, 0.000, 0.720)	0.810
(0.90, 0.05, 0.05, 0.80)	(0.90, 0.10, 0.00, 0.90)	(0.823, 0.145, 0.032, 0.720)	0.846
(0.90, 0.00, 0.10, 0.80)	(0.90, 0.10, 0.00, 0.90)	(0.836, 0.100, 0.064, 0.720)	0.882
(0.70, 0.30, 0.00, 0.80)	(0.90, 0.10, 0.00, 0.90)	(0.630, 0.370, 0.000, 0.720)	**0.630**
(0.70, 0.15, 0.15, 0.80)	(0.90, 0.10, 0.00, 0.90)	(0.669, 0.235, 0.096, 0.720)	0.738
(0.70, 0.00, 0.30, 0.80)	(0.90, 0.10, 0.00, 0.90)	(0.707, 0.100, 0.193, 0.720)	0.846
(0.80, 0.20, 0.00, 0.80)	(0.90, 0.05, 0.05, 0.90)	(0.746, 0.240, 0.014, 0.720)	0.756
(0.80, 0.10, 0.10, 0.80)	(0.90, 0.05, 0.05, 0.90)	(0.771, 0.145, 0.084, 0.720)	0.831
(0.80, 0.00, 0.20, 0.80)	(0.90, 0.05, 0.05, 0.90)	(0.797, 0.050, 0.153, 0.720)	0.907
(0.90, 0.10, 0.00, 0.80)	(0.90, 0.05, 0.05, 0.90)	(0.839, 0.145, 0.016, 0.720)	0.850
(0.90, 0.05, 0.05, 0.80)	(0.90, 0.05, 0.05, 0.90)	(0.852, 0.098, 0.051, 0.720)	0.888
(0.90, 0.00, 0.10, 0.80)	(0.90, 0.05, 0.05, 0.90)	(0.865, 0.050, 0.085, 0.720)	0.926
(0.70, 0.30, 0.00, 0.80)	(0.90, 0.05, 0.05, 0.90)	(0.653, 0.335, 0.013, 0.720)	0.661
(0.70, 0.15, 0.15, 0.80)	(0.90, 0.05, 0.05, 0.90)	(0.691, 0.193, 0.116, 0.720)	0.775
(0.70, 0.00, 0.30, 0.80)	(0.90, 0.05, 0.05, 0.90)	(0.730, 0.050, 0.220, 0.720)	0.888
(0.80, 0.20, 0.00, 0.80)	(0.90, 0.00, 0.10, 0.90)	(0.771, 0.200, 0.029, 0.720)	0.792
(0.80, 0.10, 0.10, 0.80)	(0.90, 0.00, 0.10, 0.90)	(0.797, 0.100, 0.103, 0.720)	0.871
(0.80, 0.00, 0.20, 0.80)	(0.90, 0.00, 0.10, 0.90)	(0.823, 0.000, 0.177, 0.720)	0.950
(0.90, 0.10, 0.00, 0.80)	(0.90, 0.00, 0.10, 0.90)	(0.868, 0.100, 0.032, 0.720)	0.891
(0.90, 0.05, 0.05, 0.80)	(0.90, 0.00, 0.10, 0.90)	(0.881, 0.050, 0.069, 0.720)	0.931
(0.90, 0.00, 0.10, 0.80)	(0.90, 0.00, 0.10, 0.90)	(0.894, 0.000, 0.106, 0.720)	**0.970**
(0.70, 0.30, 0.00, 0.80)	(0.90, 0.00, 0.10, 0.90)	(0.675, 0.300, 0.025, 0.720)	0.693
(0.70, 0.15, 0.15, 0.80)	(0.90, 0.00, 0.10, 0.90)	(0.714, 0.150, 0.136, 0.720)	0.812
(0.70, 0.00, 0.30, 0.80)	(0.90, 0.00, 0.10, 0.90)	(0.752, 0.000, 0.248, 0.720)	0.931

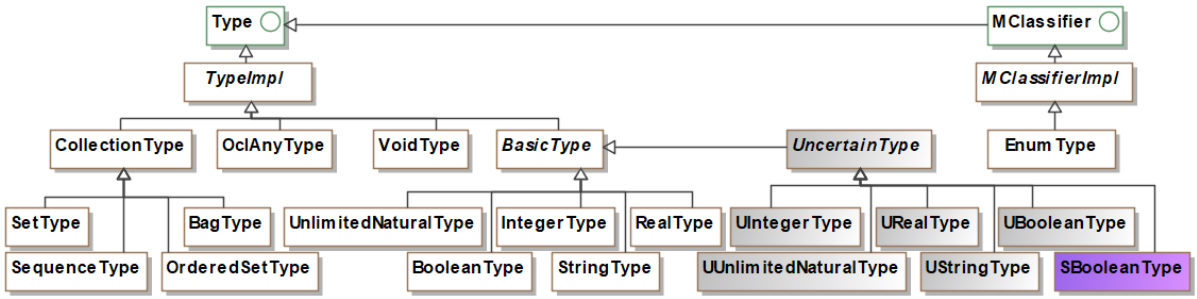


Figure 3 Content of the extended `org.tzi.use.uml.ocl.type` package.

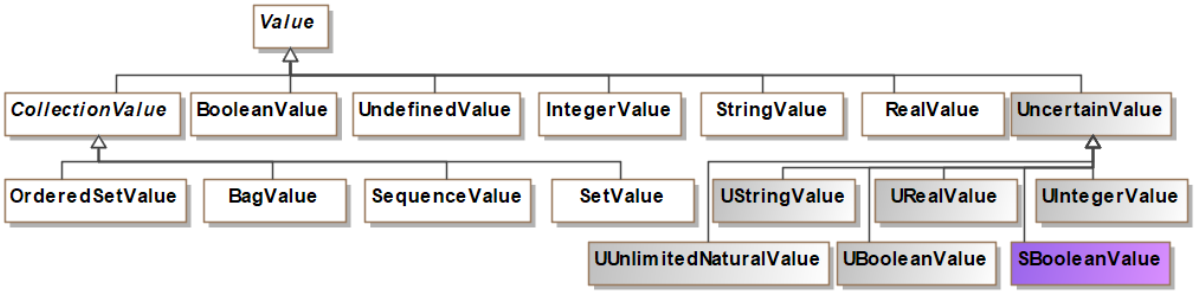


Figure 4 Content of the extended `org.tzi.use.uml.ocl.value` package.

`isEnemy` and `isArmed`, and the degree of uncertainty that the operator holds about them, which may influence the decision about whether the alarm should be raised or not. For example, if the threshold for raising the alarm is 0.75, there are many situations described in Table 1 where it will be raised, although it was not when just plain probabilities were used. This way, now we are able to take into consideration additional relevant information when making our decisions, which was neglected before. Likewise, we could dismiss opinions with very high degrees of uncertainty, since they could be misleading.

Note that the only modification required in the model is to replace the type of the appropriate `UBoolean` variables to `SBoolean`. All OCL expressions remain the same, the only change being that they are now evaluated in the upper type of the hierarchy.

```
literal returns [ASTExpression n] :
   t=`true`   { $n = new ASTBooleanLiteral(true); }
 | f=`false`  { $n = new ASTBooleanLiteral(false); }
 | i=INT      { $n = new ASTIntegerLiteral($i); }
 | r=REAL     { $n = new ASTRealLiteral($r); }
 | s=STRING   { $n = new ASTStringLiteral($s); }
   ...
 | `UBoolean` LPAREN ubve=conditionalImpliesExpression COMMA
    ubpe=additiveExpression RPAREN
    { $n = new ASTUBooleanLiteral($ubve.n, $ubpe.n); }
 | `SBoolean` LPAREN ubve=additiveExpression COMMA
        udve=additiveExpression COMMA
        uuve=additiveExpression COMMA
        uave=additiveExpression RPAREN
    { $n = new ASTSBooleanLiteral($ubve.n, $udve.n,
                                  $uuve.n, $uave.n); }
    ...;
uncertaintyType returns [ASTType n] :
    name=(`UReal`|`UInteger`|`UBoolean`|`UString`|`SBoolean`)
    { $n = new ASTSimpleType($name); } ;
```

Listing 5 Extended USE grammar.

4. Tool Support

USE (Gogolla et al. 2007) is an open-source modeling tool that enables the development and validation of UML models enriched with OCL expressions. In (Bertoa et al. 2019), we extended the USE native implementation to include uncertain datatypes as a proof of concept, and we provided a stable implementation in (Ortiz et al. 2019). Now, we have added the type `SBoolean` to our implementation.[1]

USE is implemented in Java, and its modular architecture separates the `Types`, `Values` and `Expressions` in three different packages. Despite its graphical interface, the specification of models in USE is textual, and comprises different languages which are used for different purposes: a language for the definition of class models, namely OCL; SOIL (Büttner & Gogolla 2014) for the generation of object diagrams, and a language for invoking commands from the USE console shell, among others.

The first step when adding a new datatype to the tool is to update the ANTLR grammar of its languages. For illustration purposes, Listing 5 shows an excerpt of the modified OCL grammar.

The structure of the packages `Types` and `Values` is reflected in figures 3 and 4, respectively. The elements in white are the USE original classes, the interfaces and the elements shaded in gray are the classes we have added in our previous work, and the classes shaded in purple are the contribution of this work.

The new classes `SBooleanType` and `SBooleanValue` implement the Adapter design pattern (Gamma et al. 1995), acting

[1] The code is available for download from our Git repository: https://github.com/atenearesearchgroup/uncertainty

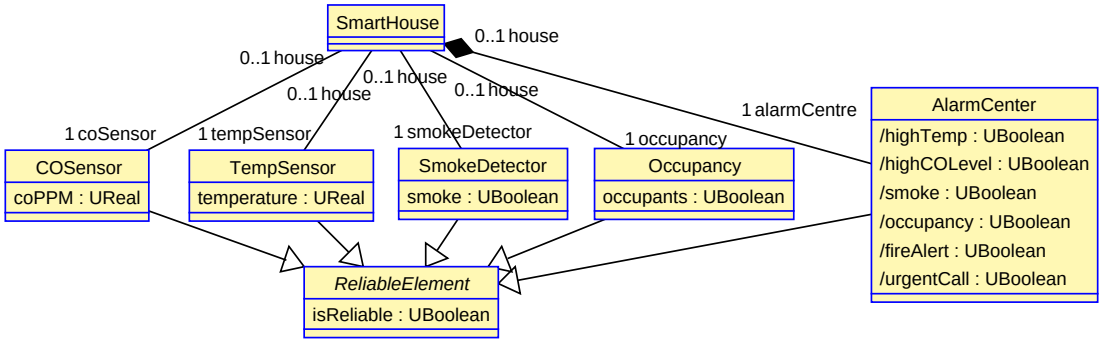


Figure 5 Class diagram for the SmartHouse system.

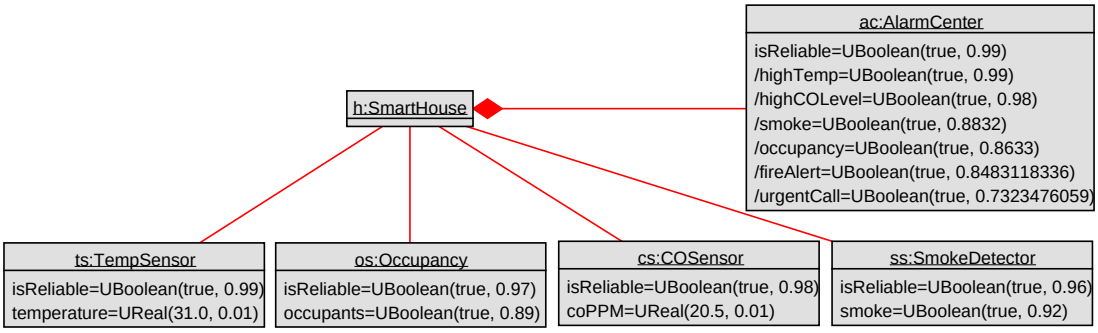


Figure 6 Object diagram for the SmartHouse system.

as a wrapper for the already existing classes. We have also modified the original USE classes to add the operations that allow the identification of the new datatypes as their supertypes of the original datatypes, when applicable. For instance, we have added to the class `BooleanType` the methods `isTypeOfSBoolean()` and `isKindOfSBoolean()`.

To make the new type available for its use as part of OCL expressions, we decided to overload the existing `Boolean` operators such as `and`, `or`, `implies`, etc. In this way, a user can perform any logic operation with any `Boolean` parameter regardless of whether it is uncertain or not.

5. Evaluation

To evaluate our proposal we have developed several case studies using the subjective logic extension to OCL to assess its *expressiveness*. One of them is presented next in subsection 5.1. It was the example that we used in our usability tests. In addition, we have carried out an empirical evaluation with users to study the *usability* of our proposal by checking that it could be effectively used by UML and OCL modelers. The experiments conducted and their results are described in subsection 5.2.

5.1. A Smart House System

Imagine a smart house equipped with four types of sensors that monitor its temperature, level of CO (measured in parts per million, PPM), smoke, and the presence of people inside. Figure 5 shows the system modeled in USE. Sensor measurements are represented using the uncertain types `UReal` or `UBoolean` to capture the inherent imprecision (measurement uncertainty) of these devices. In addition, the class `ReliableElement` is used to represent the *confidence* (also referred to as trust, or degree of belief (Burgueño et al. 2018; Burgueño, Clarisó, et al. 2019)) that a user assigns to the readings of each sensor, by means of a `UBoolean` value that specifies how trustworthy the device is.

Other devices (whose type is `AlarmCenter`) are installed in the houses, too. They provide four indicators, one for each possible situation that can be detected by the sensors: high temperature, high CO level, smoke and occupancy (i.e., presence of people in the house). Furthermore, a `fireAlert` signal is raised if the first three warnings (high temperature, CO and smoke) are set. An `urgentCall` signal is activated if warnings `fireAlert` and `occupancy` are on. The values of the attributes of the `AlarmCenter` can be derived from the values of the sensors, using the derivation rules shown in Listing 6. Note how the degree of belief (i.e., the trust) on each device is taken into account when computing the resulting value of each signal.

Figure 6 shows an object diagram with a concrete house. The reliability of each sensor varies from the more trustworthy temperature sensor to the cheaper smoke detector. Based on the sensor readings and on their assigned confidences, the likelihood of a fire in that house is almost 0.85, while the need for urgent reaction due to the presence of people inside the house is 0.73.

The problem with this model, again, is that there is no indication about the uncertainty, or lack of knowledge, of the user about some of the statements and confidence values assigned

```
context AlarmCenter
  highTemp : UBoolean derive:
    self.house.tempSensor.temperature > UReal(30.0,0.01) and
    self.house.tempSensor.isReliable
  highCOLevel : UBoolean derive:
    self.house.coSensor.coPPM > UReal(20.0,0.01) and
    self.house.coSensor.isReliable
  smoke : UBoolean derive:
    self.house.smokeDetector.smoke and
    self.house.smokeDetector.isReliable
  occupancy: UBoolean derive:
    self.house.occupancy.occupants and
    self.house.occupancy.isReliable
  fireAlert: UBoolean derive:
    highTemp and highCOLevel and smoke and self.isReliable
  urgentCall: UBoolean derive:
    fireAlert and occupancy
```

Listing 6 Derivation expressions for attributes of class `AlarmCenter`.

to the smart house elements, e.g., about their reliability or level of trust. Subjective logic, and in particular type `SBoolean`, can be very useful to represent and manage such kind of epistemic uncertainty. For example, it can be used to specify a user's subjective belief in the occurrence of a smoke alert from a particular room, which is near the kitchen. The alert is raised quite often and therefore no one trusts it very much.

The advantage of our extension is that, as it happened with the Lookout case study, updating the model to explicitly represent the users level of ignorance and uncertainty can be achieved by simply upgrading some of the attribute types to `SBoolean`. In this case, we only need to change the type of attribute `isRealiable` in the abstract class `ReliableElement`, so it can represent subjective opinions. The types of the derived attributes of class `AlarmCenter` need to be updated, too, since they use the value of attribute `isRealiable` in their derivation expressions, now of type `SBoolean`. Apart from that, no other changes are required because the type system automatically evaluates all expressions in the appropriate types.

5.2. Usability

The use of subjective logic in UML variables and OCL expressions increases the expressiveness of the models. Logical variables are now much more informative because they include the level of uncertainty of the belief agent, which helps to understand both the strength of the agent's belief that a proposition is *true* (called *credence* in statistical terms (Critch 2019)), and their degree of ignorance about it.

However, the complexity of having to handle a quadruple of values instead of a single number cannot be neglected. In particular, with such a rich type, the usability of our OCL extension could be compromised. In general, the trade-off between usability and expressiveness is common in all notations, and it has been extensively studied in query languages (Freitas et al. 2012) and lately in different kinds of software models (Sirjani 2018; Ivanchikj & Pautasso 2019). The fundamental reason for this trade-off is that greater expressiveness increases complexity, hindering usability as users require more time and effort to understand the new concepts, learn the notation, and operate with them.

To evaluate the usability of the subjective logic OCL extension, we designed a usability experiment following the US Government guidelines and recommendations on usability testing (US Department of Health & Human Services 2020). According to that report, *usability testing* aims at evaluating a product or service by testing it with representative users. Normally, during a test, participants try to complete typical tasks with the goal to identify usability problems, collect qualitative and quantitative data, and determine the participants' satisfaction with the product.

5.2.1. Usability testing *Usability* is defined by ISO/IEC as "the degree to which a product or system can be used by specified users to achieve specified goals with effectiveness, efficiency and satisfaction in a specified context of use" (ISO/IEC 25010:2011 2011). This quality characteristic is composed and described by five sub-characteristics: *Appropriateness recognizability*, *Learnability*, *Operability*, *User error protection*, and *User interface aesthetics*. Given the nature of our proposal, in this work, we are particularly interested in two of these sub-characteristics:

- *Learnability*: Degree to which a product or system can be used by specified users to achieve specified goals of learning to use the product or system with effectiveness, efficiency, freedom from risk and satisfaction in a specified context of use.
- *Operability*: Degree to which a product or system has attributes that make it easy to operate and control.

It is important to distinguish between the *perceived* and the *objective* usability. The former one (also called *subjective* usability) refers to how a user experiences the interaction, being dependent on perception and personal attitudes. In contrast, the objective usability is independent of user experience. For example, a perceived usability would measure how long a user felt he or she waited, while the objective usability would measure how long the user actually waited. In our case, the perceived usability was measured by directly asking our participants to rate the learnability and operability of the elements that they had to handle (in this case, subjective logic attributes and OCL expressions). The objective usability was measured by asking them to perform a set of tasks with UML models and OCL expressions, and evaluating their responses according to the correctness of their answers.

5.2.2. The experiment To evaluate the usability of the new concepts and of their implementations in terms of the extended UML and OCL datatypes, and in particular the new `SBoolean` type, we designed an experiment that was carried out in March 2020. The experiment consisted of two parts. The first one (with a duration of 60 minutes) was dedicated to explaining the main concepts and datatypes used in the experiment, employing a case study to illustrate them. The second part (also with a duration of 60 minutes) consisted in a test with several questions that the experiment participants should answer about a different case study, to check whether they understood the concepts and were able to successfully apply them.

The test used an online questionnaire for participants to respond, which is available at https://encuestas.uma.es/16117/lang-en. Its description points to the link from where the subjects could download the bundle that contains all the experiment documentation, the models, and the USE tool extended with uncertain datatypes.

We run the experiment with 8 subjects on March 13, 2020, and with 6 participants on March 19, 2020. The first experiment was conducted partly in the University of Malaga faculty premises, with some participants and instructors on-site and others attending remotely via teleconference. We used a projector for the people in the lab, and the instructor screen was shared for remote participants. The second experiment was entirely conducted remotely. Since the test was designed from the start to be carried out using remote access, and based on an online questionnaire, no significant problems occurred and participation went smoothly.

In both experiment executions, the first part (explanations of general uncertainty and of subjective logic) took 1 hour, as planned. Slides were used to explain the concepts, and a short document with the contents of the slides was given in advance to participants. The second part, where participants answered the questionnaire, was conducted offline. One of the instructors remained online during two hours after the experiment, in case any subject required assistance with the USE tool, had any incident, or required any clarification about the questions. No questions were asked nor assistance was required by the subjects.

The materials used during the experiment, as well as the complete responses and results, are available from our web site (Muñoz et al. 2020). In the following, we describe the main sections of the questionnaire and the results of the experiment.

Pre-test. The first section of the questionnaire was dedicated to collect the basic information about the participants, namely their degree (BSc, MSc, PhD), and level of knowledge and expertise with OCL and USE.

Learnability. To evaluate the Learnability of our proposal, we asked participants to answer six questions about the concepts that we have explained during the first part of the experiment. They included both general questions about uncertainty and concrete questions on the new type system, in particular about type `SBoolean` and the interpretation of its values.

Operability. To assess the Operability of our approach, we introduced a new model of a system, namely the Smart House example described above in subsection 5.1, which did not contain any subjective information. Then, we asked participants five questions about how they would introduce type `SBoolean` into that system. They had to implement their proposed changes in USE and test them for correctness. Finally, they had to respond some questions about the results of executing the system with a concrete object diagram, and how the results obtained could be interpreted.

Perceived Usability. The third and last section of the questionnaire asked participants about how easy or difficult they thought that the new concepts were to understand and to operate with. One of the questions asked them about their personal and subjective opinion about the expressiveness of the new type system.

Degree	Group1	Group2	All subjects
PhD	4	6	10
MSc	3	0	3
BSc	1	0	1
Total	**8**	**6**	**14**

USE Experience	Group1	Group2	All subjects
None	2	2	4
Low	4	1	5
Medium	2	2	4
High	0	1	1

OCL Experience	Group1	Group2	All subjects
Knowledge level	4.4	7.2	5.6
Years of experience	6.0	7.3	6.6

Table 2 Participants' profiles.

		Group1	Group2	All subjects
Perceived	Learnability	6.25	7.00	6.57
	Operability	7.50	8.17	7.79
	Usability	**6.88**	**7.58**	**7.18**
Objective	Learnability	9.38	8.89	9.17
	Operability	7.08	9.67	8.19
	Usability	**8.23**	**9.28**	**8.68**

Table 3 Usability test results (scores between 0 and 10).

5.3. Results

A total of 14 subjects participated in the experiment, split into two groups of 8 and 6 people each. Although Nielsen and other authors maintain that five users are enough for usability testing (Turner et al. 2006; Nielsen 2020), other authors suggest the rule of 16 ± 4 participants (Alroobaea & Mayhew 2014). By using two groups of 8 and 6 subjects with the same experiment, we tried to cover both situations. In fact, the similarity of the results obtained by both groups, and their correlation with the results of the two groups combined, seem to support Nielsen's theory.

Table 2 shows the profiles and experience with OCL and with the tool USE of the experiment participants. Ten of them (71%) held a PhD, and four of them (29%) did not have any previous experience with the tool USE. We asked participants to rate their knowledge about OCL (0-10) and the number of years of experience with UML and OCL. The level of knowledge about OCL ranged between 1 and 9, with an average of 5.6 (4.4 in the first group, 7.2 in the second). Participants' experience with UML and OCL ranged between 1 and 15 years, with an average of 6.6 years (6.0 in the first group, 7.3 in the second)

The general results of the experiment are presented in Table 3. Participant responses were scored between 0 and 10.

The upper part of the table shows the results for the *perceived* learnability and operability questions, as well as the combined usability. The lower part shows the results of the tests for the *objective* usability evaluation, i.e., the one actually exhibited by the experiment participants. Interestingly, the users considered (perceived usability) that the proposal is more difficult to use that their actual achievements show (objective usability).

When analysing in detail the outcomes of the experiment by groups, the results were generally quite homogeneous. For example, having no previous experience with USE did not have a significant influence on the results. More precisely, excluding the users with no previous experience with USE implied only a change of 1.78% with respect to the global results. The results of the PhD candidates were also similar in average (1.77% higher) when compared with the global results regarding the objective usability, although their perceived usability was 16% lower. The slightly higher scores of the second group are correlated to the higher level of knowledge and years of experience with OCL of its participants.

6. Related Work

Representing and reasoning about degrees of belief can be done using different theories. First, probability theory (Feller 2008), and in particular Bayesian probabilities, are the classical models used for quantifying subjective beliefs (de Finetti 2017), where Boolean values are assigned probabilities that represent the likelihood that the values are *true*. Related to this, the concept of *credence* is a statistical term that refers to a measure of belief strength, which expresses how much an agent believes that a proposition is *true* (Critch 2019). This approach has the advantage of simplicity: probability theory is well-known and understood by most domain experts, who could more easily use it to represent confidence in their model elements—particularly when the betting analogy is used to determine the values of their degrees of belief.

There are some extensions to OCL to deal with probabilities. For example, the Predictive, Probabilistic Architecture Modeling Framework (P2AMF) (Johnson et al. 2014), adds a probabilistic inference mechanism to OCL which is capable of probabilistic reasoning about business and IT architecture models expressed in UML. In (Bertoa et al. 2019), we proposed an extension of UML and OCL type `Boolean`, called `UBoolean` (Uncertain Boolean), which provides a probabilistic extension to binary logic. It was used in (Burgueño et al. 2018) to assign confidence to model elements, and in (Burgueño, Clarisó, et al. 2019) to assign credence to model statements.

Bayesian probabilities have been criticized for not being able to effectively represent uncertainty, with various counterexamples that cannot be successfully addressed by probability theory, e.g., the Ellsberg paradox (Liu 2018; Jøsang 2001). Several authors have proposed other approaches to tackle these problems, including possibility theory (based on fuzzy logic (Zimmermann 2001; Russell & Norvig 2010; Troegner 2010)), plausibility (a measure in the Dempster-Shafer theory of evidence (Shafer 1976)) or uncertainty theory (Liu 2018). The comparison among these theories falls out of the scope of this paper, although interesting discussions can be found in (Liu 2018) and in (Jøsang 2016).

However, these approaches also exhibit some problems when dealing with partial knowledge. The Transferable Belief Model (TBM) (Smets & Kennes 1994) addresses the same concepts considered by the Bayesian model, except it does not rely on probabilistic quantification, but on a more general system that defines belief functions based on the Dempster-Shafer model (Shafer 1976). Belief functions are transformed into probabilities only when decisions need to be made. This enables the modeler to take partial knowledge into account, which is essential when dealing with subjective beliefs.

Unfortunately, all these logics and formalisms for representing degrees of beliefs, including the TBM, have the disadvantage that the degree of uncertainty held by a agent about a system statement is not explicitly represented, as we discussed in the introduction of the paper. This is why we have explored the use of an alternative option, Subjective Logic, to introduce the explicit representation of uncertainty as an extension of Bayesian probability theory, hence combining the simplicity of classical probabilities with an expressive treatment of uncertainty as a first-class element. As our evaluation experiments show, the usability of this approach is fairly acceptable.

The OMG is working on a metamodel for the precise specification of uncertainty (PSUM) (Object Management Group 2017). It is based on the U-Model (Zhang et al. 2016) and the *Uncertum* conceptual model (Zhang et al. 2019), which is supported by a UML profile (UUP, the UML Uncertainty Profile) that enables the inclusion of uncertainty in test models. UUP defines three measure packages (Probability, Ambiguity, and Vagueness) to facilitate annotating modeling elements with different uncertainty information and measures. We consider that these specifications are complementary to our work, since they remain at a higher level of abstraction (most of the information the UML profile captures is in textual form, i.e., using Strings), whilst we are interested in not only representing belief statements but also, and more importantly, in operating with them at the OCL logic level. Therefore we need our specifications (statements and their types) to be part of the OCL type system.

Finally, other authors have proposed different extensions to OCL to incorporate new features, such as aggregation functions (Cabot et al. 2010), temporal logic (Dou et al. 2014; Ziemann & Gogolla 2003), fuzzy logic (Troegner 2010), aspects (Khan et al. 2019), or randomness (Vallecillo & Gogolla 2017). These proposals suggest the addition of new operations, but not extensions to the OCL primitive datatypes. Other works propose changes to the underlying OCL type system (Kyas 2005). This is not our case since we do not propose any modification, just an extension.

7. Conclusions and Future Work

When working with systems that operate in real environments or interact with physical elements, we are often required to manage knowledge that is vague, incomplete, or approximate. As stated in (Jøsang 2016), one of the advantages of subjective logic over traditional probabilistic logic is that the user's partial ignorance

and lack of evidence can be explicitly represented in the models, taken into account during the analyses, and explicitly expressed in the conclusions.

This paper has described an extension of the OCL and UML datatype `Boolean` that enables the representation of subjective uncertain opinions, together with a set of logical operators for reasoning with uncertain propositions in order to reach better informed decisions.

We analyzed the typical trade-off between expressiveness and usability of the proposal, conducting an empirical experiment to assess the learnability and operability of the new type and of its values and operators. Participants agreed that the new type is more informative since it now contains information of interest that could not be captured before. However, they also felt that the new values were more complex to understand and to operate with. Interestingly, this perception was completely subjective, and the final results of the test proved it wrong — participants were in fact able to handle and operate with these concepts quite successfully.

As part of our future work, we plan to conduct further experiments and usability tests to confirm these results with different groups, e.g., industrial modelers, or participants with wider ranges of skills. Carrying out usability tests comparing Subjective Logic with other logics (such as Fuzzy Logic or the TBM) could be interesting, too. We also plan to extend our previous works on the representation of Belief Uncertainty (Burgueño, Clarisó, et al. 2019) by using Subjective Logic instead of probabilities.

Dedication

This work is dedicated to Prof. Martin Gogolla on the occasion of his 65th birthday. His contributions to the modeling community, particularly on the rigorous and solid foundations of databases, modeling languages, OCLs and abstract data types, have been essential to the development and formalization of many of the concepts that are part of our current modelling practices and notations. We would like to thank him for his inspiration, enthusiasm, and thorough work, which has significantly helped the entire community learn about the exciting world of conceptual modeling. In Málaga, we are particularly grateful to him for his frequent visits and for the development of the USE tool, which we have used intensively in our lectures and research projects over the past 10 years, and which has enabled us to implement many of our ideas and proposals. Without it, neither our lectures nor our research would be the same. Finally, we will always be indebted to him for his amazing scientific legacy and, more importantly, for his generosity and extraordinary personality. We hope that his retirement does not mean that he will only jog in the mornings, but that he will continue to guide the community with his ideas, projects, *beasts* and tools. Thank you, Martin!

Acknowledgments

We would like to thank the anonymous reviewers for their useful comments and suggestions that have helped to notably improve the paper. This work has been supported by Spanish Research projects TIN2016-75944-R and PGC2018-094905-B-I00.

References

Alroobaea, R., & Mayhew, P. J. (2014). How many participants are really enough for usability studies? In *Proc. of the 2014 science and information conference (sai)* (p. 48-56). IEEE. doi: 10.1109/SAI.2014.6918171

America, P. (1987). Inheritance and subtyping in a parallel object-oriented language. In *Proc. of ECOOP'87* (Vol. 276, pp. 234–242). Springer. doi: 10.1007/3-540-47891-4_22

America, P. (1991). A behavioural approach to subtyping in object-oriented programming languages. In M. Lenzerini, D. Nardi, & M. Simi (Eds.), *Inheritance hierarchies in knowledge representation and programming languages* (pp. 173–190). John Wiley and Sons.

Bertoa, M. F., Burgueño, L., Moreno, N., & Vallecillo, A. (2019). Incorporating measurement uncertainty into OCL/UML primitive datatypes. *Software and Systems Modeling*. doi: 10.1007/s10270-019-00741-0

Boute, R. T. (1990). A heretical view on type embedding. *SIGPLAN Notices*, *25*(1), 25–28. doi: 10.1145/74105.74108

Bucchiarone, A., Cabot, J., Paige, R. F., & Pierantonio, A. (2020). Grand challenges in model-driven engineering: an analysis of the state of the research. *Software and Systems Modeling*, *19*(1), 5–13. doi: 10.1007/s10270-019-00773-6

Burgueño, L., Bertoa, M. F., Moreno, N., & Vallecillo, A. (2018). Expressing confidence in models and in model transformation elements. In *Proc. of MODELS'18* (pp. 57–66). ACM. doi: 10.1145/3239372.3239394

Burgueño, L., Clarisó, R., Cabot, J., Gérard, S., & Vallecillo, A. (2019). Belief uncertainty in software models. In *Proc. of MiSE@ICSE'19* (pp. 19–26). ACM. doi: 10.1109/MiSE.2019.00011

Burgueño, L., Mayerhofer, T., Wimmer, M., & Vallecillo, A. (2019). Specifying quantities in software models. *Information & Software Technology*, *113*, 82–97. doi: 10.1016/j.infsof.2019.05.006

Büttner, F., & Gogolla, M. (2014). On OCL-based imperative languages. *Sci. Comput. Program.*, *92*, 162–178. doi: 10.1016/j.scico.2013.10.003

Cabot, J., Mazón, J., Pardillo, J., & Trujillo, J. (2010). Specifying aggregation functions in multidimensional models with OCL. In *Proc. of ER'10* (Vol. 6412, pp. 419–432). Springer. doi: 10.1007/978-3-642-16373-9_30

Clerici, S., & Orejas, F. (1988). GSBL: an algebraic specification language based on inheritance. In *Proc. of ECOOP'88* (Vol. 322, pp. 78–92). Springer. doi: 10.1007/3-540-45910-3_5

Critch, A. (2019). *Credence — using subjective probabilities to express belief strengths.* Retrieved from http://acritch.com/credence/

de Finetti, B. (2017). *Theory of probability: A critical introductory treatment*. John Wiley & Sons. doi: 10.1002/9781119286387

Dou, W., Bianculli, D., & Briand, L. C. (2014). OCLR: A more expressive, pattern-based temporal extension of OCL. In *Proc. of ECMFA'14* (Vol. 8569, pp. 51–66). Springer. doi: 10.1007/978-3-319-09195-2_4

Feller, W. (2008). *An introduction to probability theory and its applications*. Wiley.

Freitas, A., Curry, E., Oliveira, J. G., & O'Riain, S. (2012). Querying heterogeneous datasets on the linked data web: Challenges, approaches, and trends. *IEEE Internet Comput.*, *16*(1), 24–33. doi: 10.1109/MIC.2011.141

Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1995). *Design patterns: Elements of reusable object-oriented software*. Addison-Wesley Longman.

Gogolla, M., Büttner, F., & Richters, M. (2007). USE: A UML-based specification environment for validating UML and OCL. *Sci. Comput. Program.*, *69*(1-3), 27–34. doi: 10.1016/j.scico.2007.01.013

ISO/IEC 25010:2011. (2011). *Systems and software engineering – Systems and software Quality Requirements and Evaluation (SQuaRE) – System and software quality models.* Retrieved from https://www.iso.org/obp/ui/#iso:std:iso-iec:25010:ed-1:v1:en

Ivanchikj, A., & Pautasso, C. (2019). Sketching process models by mining participant stories. In *Proc. of BPM forum 2019* (Vol. 360, pp. 3–19). Springer. doi: 10.1007/978-3-030-26643-1_1

JCGM. (2008). *Evaluation of measurement data – Guide to the expression of uncertainty in measurement (GUM).* Retrieved from http://www.bipm.org/utils/common/documents/jcgm/JCGM_100_2008_E.pdf (JCGM 100:2008)

Johnson, P., Ullberg, J., Buschle, M., Franke, U., & Shahzad, K. (2014). An architecture modeling framework for probabilistic prediction. *Inf. Syst. E-Business Management*, *12*(4), 595–622. doi: 10.1007/s10257-014-0241-8

Jøsang, A. (2001). A logic for uncertain probabilities. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, *9*(3), 279–212. doi: 10.1142/S0218488501000831

Jøsang, A. (2016). *Subjective logic – A formalism for reasoning under uncertainty*. Springer.

Jøsang, A., Keser, C., & Dimitrakos, T. (2005). Can we manage trust? In *Proc. of itrust* (Vol. 3477, pp. 93–107). Springer. doi: 10.1007/11429760_7

Khan, M. U., Sartaj, H., Iqbal, M. Z., Usman, M., & Arshad, N. (2019). AspectOCL: using aspects to ease maintenance of evolving constraint specification. *Empirical Software Engineering*, *24*(4), 2674–2724. doi: 10.1007/s10664-019-09717-6

Kyas, M. (2005). An extended type system for OCL supporting templates and transformations. In *Proc. of FMOODS'05* (Vol. 3535, pp. 83–98). Springer. doi: 10.1007/11494881_6

Lee, E. A. (2008). Cyber physical systems: Design challenges. In *Proc. of ISORC'08* (pp. 363–369). IEEE Computer Society. doi: 10.1109/ISORC.2008.25

Liskov, B. H., & Wing, J. M. (1994, November). A behavioral notion of subtyping. *ACM Trans. Program. Lang. Syst.*, *16*(6), 1811–1841. doi: 10.1145/197320.197383

Liu, B. (2018). *Uncertainty theory* (5th ed.). Springer.

Muñoz, P., Burgueño, L., Ortiz, V., & Vallecillo, A. (2020, April). *Extending OCL with Subjective Logic.* Retrieved from http://atenea.lcc.uma.es/projects/SubjectiveLogic.html

Nielsen, J. (2020). *How many test users in a usability study?* Retrieved from https://www.nngroup.com/articles/how-many-test-users/

Oberkampf, W. L., DeLand, S. M., Rutherford, B. M., Diegert, K. V., & Alvin, K. F. (2002). Error and uncertainty in modeling and simulation. *Reliability Engineering & System Safety*, *75*(3), 333–357.

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

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

Object Management Group. (2017, May). *Precise Semantics for Uncertainty Modeling (PSUM) RFP.* Retrieved from https://www.omg.org/cgi-bin/doc.cgi?ad/2017-12-1 (OMG Document ad/2017-12-1)

Olivé, A. (2007). *Conceptual modeling of information systems*. Springer. doi: 10.1007/978-3-540-39390-0

Ortiz, V., Burgueño, L., Vallecillo, A., & Gogolla, M. (2019). Native support for UML and OCL primitive datatypes enriched with uncertainty in USE. In *Proc. of OCL'19* (Vol. 2513, pp. 59–66). CEUR-WS.org. Retrieved from http://ceur-ws.org/Vol-2513/paper5.pdf

Russell, S. J., & Norvig, P. (2010). *Artificial intelligence. a modern approach* (3rd ed.). Prentice Hall.

Selic, B. (2015). Beyond Mere Logic – A Vision of Modeling Languages for the 21st Century. In *Proc. of MODELSWARD'15 and PECCS'15* (pp. IS–5). SciTePress. Retrieved from http://cescit2015.um.si/Presentations/KN_Selic.pdf

Shafer, G. (1976). *A mathematical theory of evidence*. Princeton University Press.

Sirjani, M. (2018). Power is overrated, go for friendliness! expressiveness, faithfulness, and usability in modeling: The actor experience. In *Principles of modeling - essays dedicated to edward a. lee on the occasion of his 60th birthday* (Vol. 10760, pp. 423–448). Springer. doi: 10.1007/978-3-319-95246-8_25

Smets, P., & Kennes, R. (1994). The transferable belief model. *Artificial Intelligence*, *66*(2), 191–234. doi: 10.1016/0004-3702(94)90026-4

Thunnissen, D. P. (2003). Uncertainty classification for the design and development of complex systems. In *Proc. of the 3rd annual predictive methods conference, veros software.* Retrieved from http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.128.133

Troegner, D. (2010). Combination of fuzzy sets with the object constraint language (OCL). In *Proc. of informatik'10* (Vol. P-176, pp. 705–710). Retrieved from https://dl.gi.de/20.500.12116/19308

Turner, C. W., Lewis, J. R., & Nielsen, J. (2006). Determining usability test sample size. In *The international encyclopedia of ergonomics and human factors* (2nd ed., Vol. 3, pp. 3084–3088). CRC Press.

US Department of Health & Human Services. (2020, March). *Usability testing.* Retrieved from https://www.usability.gov/how-to-and-tools/methods/usability-testing.html

Vallecillo, A., & Gogolla, M. (2017). Adding random operations to OCL. In *Proc. of MODEVVA'17* (pp. 324–328).
Zhang, M., Ali, S., Yue, T., Norgren, R., & Okariz, O. (2019). Uncertainty-wise cyber-physical system test modeling. *Software and System Modeling*, *18*(2), 1379–1418. doi: 10.1007/s10270-017-0609-6
Zhang, M., Selic, B., Ali, S., Yue, T., Okariz, O., & Norgren, R. (2016). Understanding uncertainty in cyber-physical systems: A conceptual model. In *Proc. of ECMFA'16* (Vol. 9764, pp. 247–264). Springer. doi: 10.1007/978-3-319-42061-5_16
Ziemann, P., & Gogolla, M. (2003). OCL extended with temporal logic. In *Proc. of PSI'03* (Vol. 2890, pp. 351–357). Springer. doi: 10.1007/978-3-540-39866-0_35
Zimmermann, H.-J. (2001). *Fuzzy set theory – and its applications* (4th ed.). Springer Science+Business Media.

About the authors

Paula Muñoz is a PhD candidate at the University of Málaga. She graduated in Software Engineering from the University of Málaga in June 2019. Her research focuses on the precise specification and testing of software systems using models. Contact her at paulam@lcc.uma.es.

Loli Burgueño is a postdoctoral researcher and lecturer at the Open University of Catalonia (Spain) and CEA LIST (France). Her research interests focus on Model-Driven Engineering, in particular the performance, scalability and testing of model transformations, the modeling of uncertainty in software models for its use in the Industry 4.0 and the integration of Artificial Intelligence techniques into modeling tools and processes. You can contact the author at lburguenoc@uoc.edu or visit https://som-research.uoc.edu/loli-burgueno/.

Victor Ortiz is Software Engineer at Minsait, Indra (Spain). He graduated in Computer Science and Engineering from the University of Málaga in June 2019. He works on web-oriented multi-environment projects in J2EE. His main interests are related to modeling, methodology, testing and software quality. You can contact the author at vikour92@gmail.com or visit https://www.linkedin.com/in/vikour/.

Antonio Vallecillo is full Professor at the University of Málaga. He leads the Atenea Research Group on Software and Systems Modeling, which is focused on basic and applied research on modeling software systems, and on the provision of engineering tools for designing, analysing, evaluating and implementing distributed information systems. His main research interests include Open Distributed Processing, Model-based Engineering and Software Quality. You can contact the author at av@lcc.uma.es or visit http://www.lcc.uma.es/~av/.