Test Case Generation in Software Modeling

1. Introduction

When designing software and systems, model generators play an important role to automatically generate test cases (e.g., [1], [2]). Nevertheless, Hernandez-López et al. [3] have reported that current model validators do not generate models that can be considered realistic from a human point of view. This means that, in real scenarios, when model generators are used, the models generated need to be checked by humans before they are accepted as-is, modified or discarded.

In this report, we present a case where Classifying Terms (CTs) [4] have been used to guide the test case generation [5] and where our approach is used to add beliefs to the generated test cases (i.e., model instances) and to reason about them before they are accepted and used for testing purposes or discarded.

Let us assume that we are dealing with the domain model from [4] that represents Families (shown in Figure 1). In this model, each person has a first name (`fName`), a last name (`lName`) and a birth year (`yearB`), and the Parenthood relation between persons is captured by means of an association.

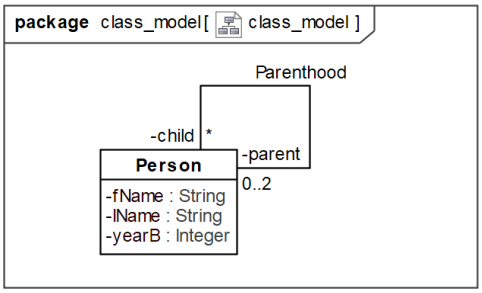

Figure 1. Families class diagram (from [5])

Using Classifying Terms, a set of diverse test cases were automatically constructed[1]. Figure 2 shows one of the test cases (represented as an UML object diagram) that was generated for the class diagram from Figure 1.

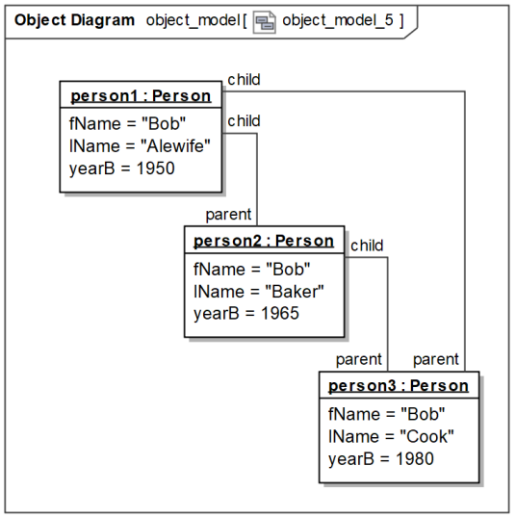

Figure 2. Generated object model (from [5])

[1] We refer the reader to [4] for more details about the concept and use of Classifying Terms.

2. Assigning degrees of belief to model elements

By looking at the object model in Figure 2, we soon observe that this particular test case has several issues that makes us doubt its chances to reflect a real situation. Although it conforms to its metamodel and respects all of its constraints, it does not seem usual to find a person named Bob that has a child named Bob when he was fifteen, and later, these two people, with a direct blood relation, have together another child when the older Bob is 30 and the second Bob is 15.

When a designer faces a situation like this, instead of discarding the whole model – which could lead to discarding all the automatically generated models [3] – they could annotate the model elements they are not sure about with their own belief.

With this goal in mind, in this work, we are interested in representing, characterizing, and processing uncertain information in UML software models, considering the stakeholders' opinions and beliefs. We aim at associating "beliefs" to model elements, and how to propagate and operate with their associated uncertainty so that domain experts can reason about software models enriched with individual opinions.

To represent degrees of belief we use Subjective Logic [Jos16] which is an extension of both Boolean and Probabilistic logics to account for uncertainty. Values in this logic are 4-tuples called "opinions." More precisely, given a Boolean logic predicate x, an opinion o(x) is the 4-tuple (*b*,*d*,*u*,*a*) where:

- *b* (belief) is the degree of belief that x is true.
- *d* (disbelief) is the degree of belief that x is false
- *u* (uncertainty) is the degree of uncertainty about x, i.e., the amount of uncommitted belief.
- *a* (base rate) is the prior probability of x without any previous evidence.

All four components of the tuple are in the range [0,1] and satisfy that b + d + u = 1.

Boolean logic is embedded in Subjective logic by making the value "true" correspond to opinion (1,0,0,1), and false to (0,1,0,0). Probabilistic logic (i.e., a logic whose values are probabilities) is embedded in Subject logic by making a value *p* (with 0 <= *p* <= 1) correspond to opinion (*p*, 1 – *p*, 0, *p*). Given an opinion (*b*,*d*,*u*,*a*), its "*projection*" enables defining a projection from Subjective logic to Probabilistic logic. The projection of an opinion is defined by the formula *P* = *b* + *a***u.*

To assign degrees of beliefs to UML model elements we have defined a UML profile, which is shown below:

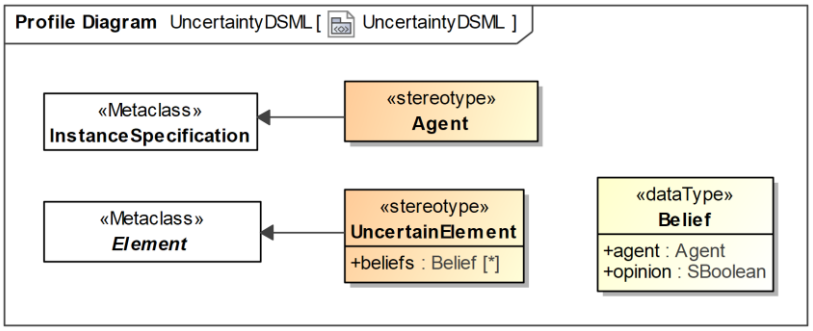

Figure 3 The UML Profile for expressing degrees of belief on UML model elements

Stereotype `Agent` is used to indicate the entities that hold the opinions, which can be domain experts, users, modelers or any other model stakeholder. Stereotype `UncertainElement` is used to mark a model element as uncertain, and assign a degree of belief to it. For this, datatype `Belief` represents a pair `(agent:Agent, opinion:SBoolean)` that describes the agent holding an opinion, and the opinion held by that agent. We can see how an uncertain element can be assigned a set of beliefs, so using this stereotype different agents can express their opinion about the same model element.

This profile is normally used to assign degrees of belief either to entity instances or to attributes' values. In the former case, the degree of belief expresses how sure we are about the actual occurrence (or existence) of the instance. When applied on attributes' values, it means how sure or unsure we are about these values. Note that assigning no belief to an element is equivalent to assigning a TRUE belief (i.e., complete certainty) to that model element.

Given the doubts that our designer had about the automatically generated object model, they decided to annotate those elements that they are not sure about using our profile. For instance, they have decided that the associations between `person1` and `person2` and between `person2` and `person3` are very unlikely. Therefore, they have a belief with an opinion `SBoolean(0.1, 0.8, 0.1, 0.5)`. If these two associations did not exist, the designer could believe that the remaining association (between `person1` and `person3`) is correct. Thus, they have not assigned any belief to it (meaning that they fully trust the existence of the association). Furthermore, given that the parent's last name is Alewife and that usually children take their father's last name, the designer has also assigned the belief `b2` to the last name of `person3`. All this information is captured in the object model presented in Figure 4.

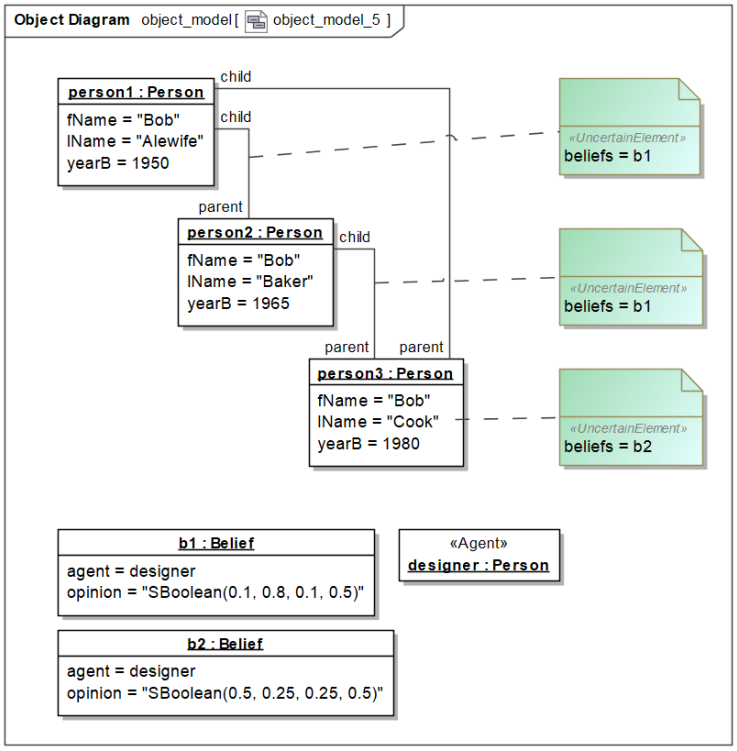

Figure 4. Test case annotated by designer – opinions on elements.

3. Assigning degrees of belief to test cases

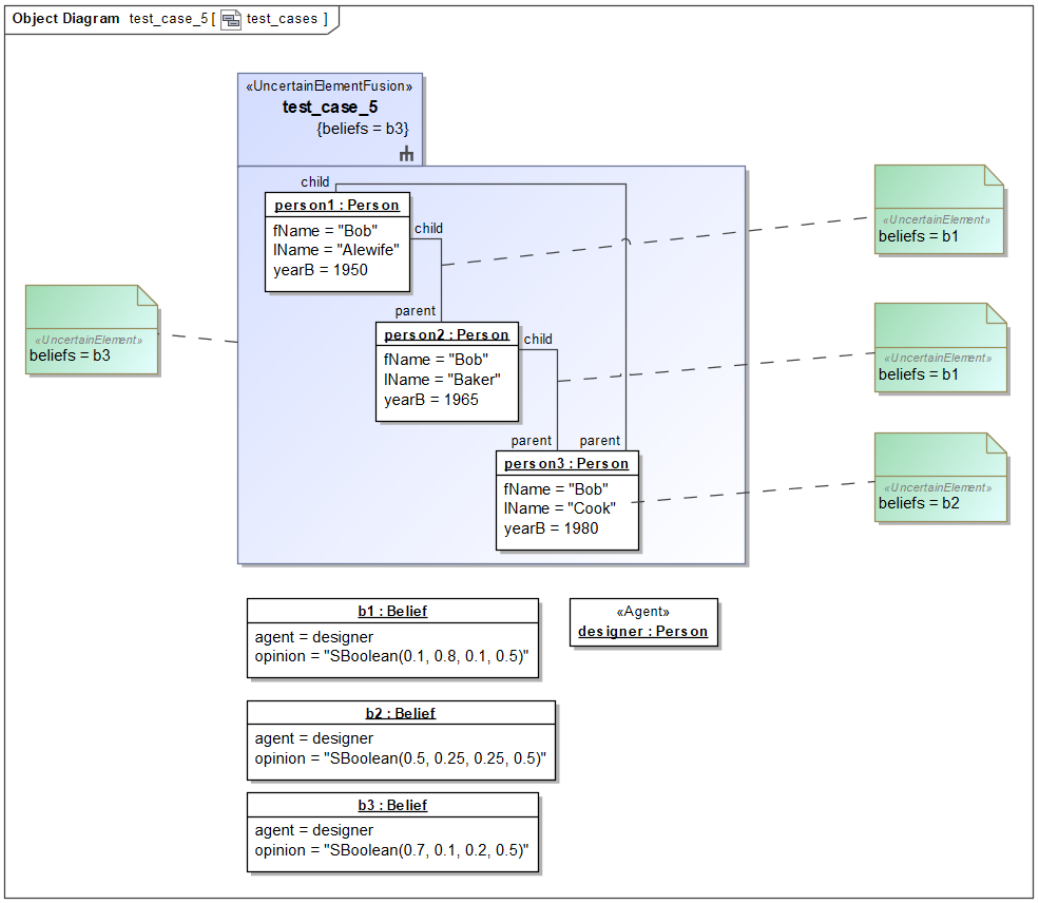

Figure 5. Test case annotated by designer – opinions on elements and test case

Apart from assigning opinions to the model elements that compose the test cases, an opinion could be assigned to a test case as a whole – considering it as a single entity. This can be done by assigning a belief to the package that contains the model elements that compose the test case. Figure 5 shows an example in which the package has been stereotyped as an uncertain element and the belief b3 has been assigned to it.

In such a case, the belief that an agent holds about a concrete model element is conditional to the belief stated on the test case. This is, the designer could compute the updated belief of a model element using the and operator. For example, for the association between p1 and p2, the updated value could be calculated with the formula: `b1 and b3`.

4. Combining opinions from different agents

There could be the case that there were more than one person deciding on the value and usefulness of the test cases that have been automatically generated. Then, this group of people would have to work together to reach a consensus.

Let us assume that our designer is working together with a QA tester, who is also asked to annotate the test case with their beliefs. Figure 6 shows the result after this step. We can observe that the QA tester decided to only question the test case (and none of its model elements) with a belief b4 – whose opinion is `SBoolean(0.9, 0.0, 0.1, 0.5)` – claiming that the test case is not bad as it could cover a corner case, and hence be useful when testing the system.

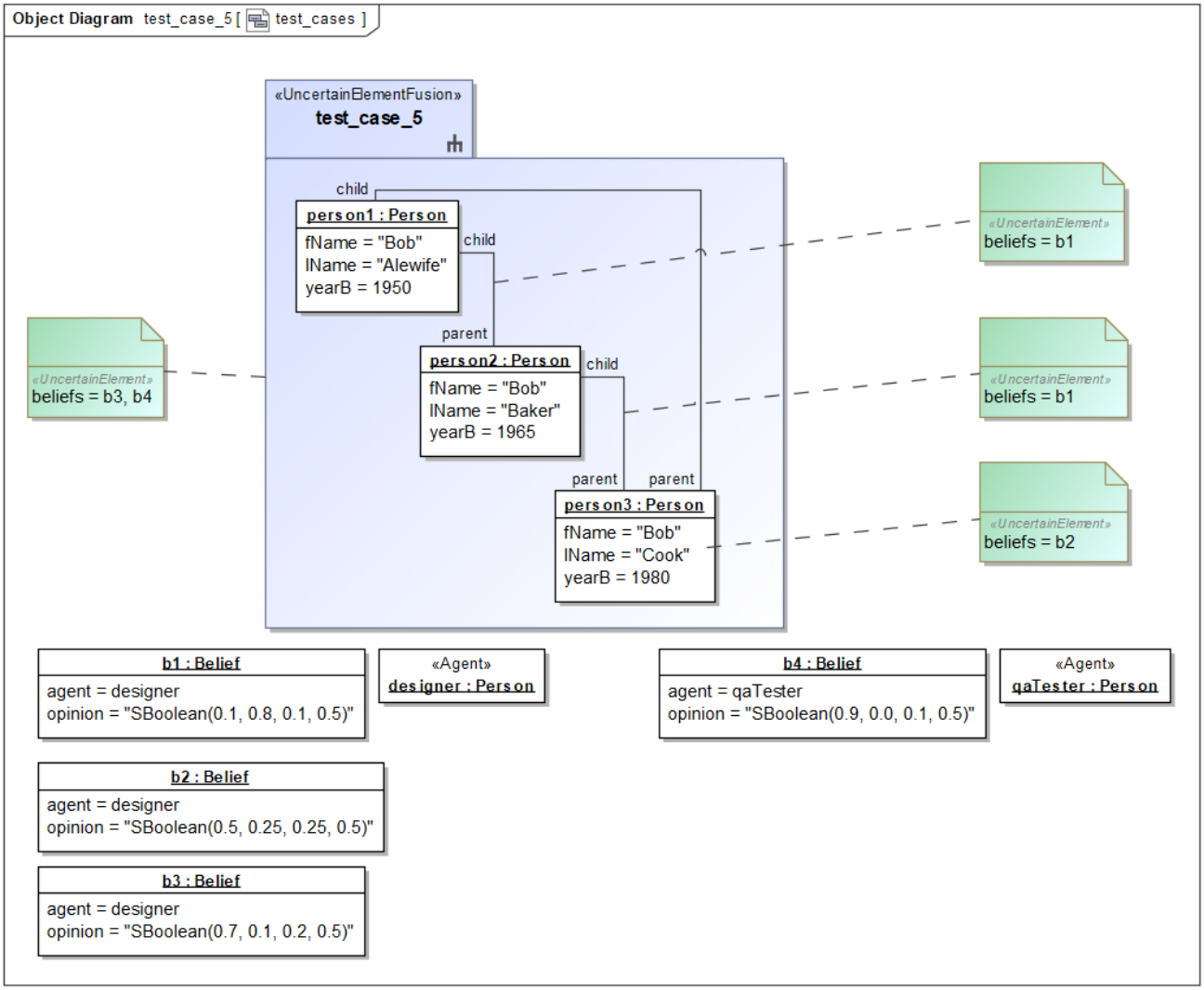

Figure 6. Test case annotated by the designer and the QA tester

Finally, the designer and QA tester need to merge their opinions for the test case.

The fact that the QA tester gives their opinion increases the amount of independent evidence (one more agent/source of knowledge) and decreases the degree of uncertainty (due to their expertise). Therefore, the Aleatory Cumulative Belief Fusion (ACBF) operator is the most appropriate in this particular situation.

```
ACBF(b3, b4) = SBoolean(0.89, 0.04, 0.07, 0.5)
```

Given the result of the fusion operator (a high degree of belief with low uncertainty), both the designer and QA tester decide to keep the test case as-is, i.e., without any manual modification.

References

[1] "EMF random instantiator," AtlanMod (Inria, Mines-Nantes), 2021. [Online]. Available: https://github.com/atlanmod/mondo-atlzoo-benchmark/tree/master/fr.inria.atlanmod.instantiator.

[2] M. Scheidgen, "Generation of large random models for benchmarking," in *BigMDE@STAF*, 2015.

[3] J. S. C. Jose Antonio Hernández López, "Towards the Characterization of Realistic Model," in *MODELS'21*, Fukuoka, Japan, 2021.

[4] M. Gogolla, A. Vallecillo, L. Burgueño and F. Hilken, "Employing classifying terms for testing model transformations," in *MODELS*, 2015.

[5] F. Hilken, M. Gogolla, L. Burgueño and A. Vallecillo, "Testing models and model transformations using classifying terms," *Software and System Modeling,* vol. 17, no. 3, pp. 885-912, 2018.