JOT
JOURNAL OF
OBJECT TECHNOLOGY

Journal of Object Technology | **RESEARCH ARTICLE**

Modeling Objects with Uncertain Behaviors

Paula Muñoz*, Priyanka Karkhanis†, Mark van den Brand†, and Antonio Vallecillo*
*ITIS Software. Universidad de Málaga, Spain
†Technical University of Eindhoven, The Netherlands

ABSTRACT Modeling the behavior of complex systems that operate in real environments, deal with physical elements, or interact with humans is a challenging task. It involves the explicit representation of aspects of behavioral uncertainty that are inherent in the system, but generally neglected in software models. In this paper, we focus on the explicit representation of the behavior of objects of complex systems, considering their motivations, randomness, and the different types of underlying uncertainty that affect their actions. We show how such uncertain behaviors can be effectively modeled in UML and OCL, and how the specifications produced can be used to simulate and analyze these systems.

KEYWORDS Software models, UML, uncertain behavior, uncertainty, randomness, simulation.

1. Introduction

One of the current challenges of software models is related to their ability to accurately represent systems that exhibit complex and uncertain behavior, such as those that operate in real environments, deal with physical elements, or interact with humans (Bucchiarone et al. 2020). We are particularly interested in systems where humans play a role, either because the system needs to interact with them, or emulate their behavior (Pedersen et al. 2018). Examples of such systems include those that model the behavior of cars and pedestrians at intersections (Ridel et al. 2018), those of autonomous vehicles (Liu et al. 2018), or those of cyberphysical systems (Kirchhof et al. 2020). They are common in different domains such as intelligent transportation systems (ITS) (Dajsuren & van den Brand 2019; Karkhanis et al. 2018) or in the Industry 4.0 applications (Mosterman & Zander 2016; Wortmann et al. 2020). In these domains, system properties such as correctness or safety are critical.

Modeling the behavior of agents in these systems is non-trivial due to their complex behavior. This may depend on their profile and abilities, current state and personal motivation, or on random events (Jay et al. 2020; Nassal & Tichy 2016). Furthermore, the environment of such systems normally involves unknown factors and circumstances (Shi et al. 2005; Geller & Bradley 2012).

Starting from a traditional behavioral specification based on state machines, our proposal for modeling these types of behaviors uses a combination of:

- probabilistic state machines (Novák 2009) and influence diagrams (Howard & Matheson 2005) for representing the causalities and effects of the agents' actions (Pearl 1994; Darwiche 2009; Pearl 2000);
- the motivational forces that influence the agents' decisions towards achieving their goals (Nassal & Tichy 2016);
- the explicit representation and propagation of the aleatory uncertainty due to the imprecision in the measuring tools or unreliable sources (JCGM 100:2008 2008);
- the subjective interpretation of the environment by each individual agent and the confidence assigned to its data sources (Jøsang 2016; Burgueño, Clarisó, et al. 2019; Muñoz et al. 2020); and
- the representation of the randomness inherent to any real environment (Oberkampf et al. 2002; Garlan 2010).

Although there are proposals to model these aspects using various approaches, typically using agent systems, they usually do not cover all these factors. In addition, they tend to use lower level languages and platforms, which makes the simulation of these types of systems rather complex and cumbersome.

Our goal is to show how these behaviors and associated uncertainties can be effectively modeled in high-level languages such as UML (Object Management Group 2015) and OCL (Ob-

JOT reference format:
Paula Muñoz, Priyanka Karkhanis, Mark van den Brand, and Antonio Vallecillo. *Modeling Objects with Uncertain Behaviors.* Journal of Object Technology. Vol. 20, No. 3, 2021. Licensed under Attribution 4.0 International (CC BY 4.0) http://dx.doi.org/10.5381/jot.2021.20.3.a8

ject Management Group 2014), and how these specifications can be used to simulate and analyze systems. We demonstrate our approach with two exemplar applications, one from the traffic domain and another from the games simulation field.

After this introduction, Sect. 2 briefly presents the background of our work and provides an overview of related works that model agent and human behavior. Then, Sect. 3 presents our proposal, describing how to represent the behavior of agents under the presence of uncertainty. It includes two applications that we have developed to demonstrate and evaluate the proposal. Sect. 4 discusses the types of analyses that can be carried out with our specifications. Finally, Sect. 5 concludes with an outlook on future work.

2. Related Work and Background

2.1. Uncertainty

Uncertainty is an inherent property of any system that operates in a real environment or that interacts with physical elements. Uncertainty can be due to imprecision in the measuring tools; lack of knowledge about the system or its environment; imperfect, incorrect, or missing information; unreliable data sources and communication networks; numerical approximations; different interpretations of the same evidences by different parties, or the inability to determine whether a particular event has actually occurred or not (JCGM 100:2008 2008).

There are several uncertainty classifications. The primary one divides it into aleatory and epistemic (Kiureghian & Ditlevsen 2009). *Aleatory uncertainty* refers to the inherent probabilistic variability or randomness of a phenomenon. For example, measuring the distance between two objects, or the duration of a software development process. This type of uncertainty is irreducible, in that there will always be variability in the underlying variables (JCGM 100:2008 2008). *Epistemic uncertainty* refers to the lack of knowledge we have about the system (modeled or real) or its elements. For example, the confidence we have on the actual occurrence of a modeled event. This type of uncertainty is reducible, in that additional information or knowledge may reduce it.

The increasing need to model physical systems has led to several modeling proposals that explicitly represent different types of uncertainty in software models (Troya et al. 2021). The OMG is also working on a metamodel for the precise specification of uncertainty (PSUM) (Object Management Group 2017), based on the U-Model (Zhang et al. 2016) and the *Uncertum* conceptual model (Zhang et al. 2019). Some works address particular types of uncertainties using different notations and logics, namely, Measurement uncertainty (Burgueño, Mayerhofer, et al. 2019; Bertoa et al. 2020); Design uncertainty (Zhang et al. 2018; Famelis et al. 2012; Salay et al. 2013); Occurrence uncertainty (Burgueño et al. 2018); Belief uncertainty (Burgueño, Clarisó, et al. 2019; Martín-Rodilla & Gonzalez-Perez 2019), or Data uncertainty (Jing et al. 2008; Zhou et al. 2009; Wang & Bai 2019). However, the specification of the behavior of agents subject to uncertainty, which combines several of these types of uncertainty, has received less attention by the modeling community (Giese et al. 2014).

2.2. Modeling the behavior of agents

In UML, a Behavior is a specification of events that may occur dynamically over time (Object Management Group 2015). Behavior can be specified in terms of operations, state machines, sequence diagrams and activities. Typically, reactive and real time systems are specified using state machines, with many tools available for the specification, simulation and analysis of such systems. These are normally deterministic specifications.

Several approaches have been proposed for the specification of models of agent behavior. First, *rule-based systems* define actions that are triggered based on the rules specified by the system modeler. The set of rules can even be dynamically changed during the system simulation. They are common in simulation games such as SimSE (Navarro & van der Hoek 2004), SESAM (Ludewig et al. 1992), SimVBSE (Jain & Boehm 2006), or SimjavaSP (Shaw & Dermoudy 2005). They simulate humans following precisely the instructions given by the rules, which make their behavior very predictable and therefore different from human behavior.

Another approach to model the behaviour of agents emulating humans is by using parameterized feedback loops represented by multiple equations that describe their actions or trajectories (Ridel et al. 2018; Albrecht & Stone 2018). Although the models are very accurate, the behaviour of the agents is again predictable. Besides, they normally neglect the inherent randomness and uncertainties existing in real environments (Thompson & Smith 2019).

There is a vast literature on the specification of probabilistic or stochastic behavior, which allows the description of different alternative actions and their probability of occurrence. They allow simulations of systems that calculate alternative outcomes and their probabilities. For example, some works provide extensions to UML for modeling stochastic statecharts (Novák 2009) or sequence diagrams (Refsdal 2008; Refsdal & Stølen 2008). Fuzzy-DEVS (Kwon et al. 1996) extends the DEVS formalism for simulating uncertain behavior. Models@runtime are also successfully used for self-adaptive robots under uncertainty (Giese et al. 2014). The main drawback of these approaches is that they are not able to describe the causes that motivate the resulting behavior (*motivational forces*), and how these causes influence the behavior.

To address these limitations, different approaches, e.g., (Sharma et al. 2018), make use of Bayesian networks (Pearl 1994; Darwiche 2009), or Causal networks (Pearl 2000; Pearl & Mackenzie 2018) to model the behaviour of agents. These networks can implement the decision-making process conducted by the agents to decide the next action to perform. One of the main benefits of these proposals is that they enable the specification of different features of the agents (character, skills or current state). These approaches provide more realistic simulations of human behavior and how they influence their decisions, normally to maximize utility (Pedersen et al. 2018; Neumann & Morgenstern 1953).

The *Social Force Model* is used by other authors to explicitly represent and reason about the motivations of agents to decide the action to perform next (Helbing & Molnár 1995; Huang et al. 2017). Similar approaches use the concept of *Motivational*

Force (Nassal & Tichy 2016) for every possible action, in order to perform the one with the highest motivation. In particular, the motivational force of a given action is computed as the maximum benefit (called *valence*) of all possible results of that action. Typically, the action with the highest valence will be performed by the agent. However, incorporating uncertainty into these behavioral specifications is still an open issue (Albrecht & Stone 2018), and one of the goals of this paper.

3. Modeling the behavior of agents

This section describes our approach to modeling uncertainty in the behavior of agents in complex systems, such as those involving humans with unpredictable behavior, environmental conditions and imperfect physical elements.

3.1. Methodology

Our proposal consists of a set of steps that incorporates the different uncertainty factors to the behavior of the system objects.

- First, we specify the possible actions that each agent can perform, and the agent's state machines that describe its basic behavior using a traditional (deterministic) approach.
- Second, we specify the possible situations in which the agents can be depending on their environmental conditions and the subjective interpretation of the environment by each individual agent, and the confidence assigned to its data sources (*situational context*). These situations will be used to further refine the state machines of the agents.
- Third, we should identify the motivational forces that influence the agents' decisions towards achieving their goals (Nassal & Tichy 2016).
- Fourth, specify the corresponding weights of the situational context and the motivational forces when combining them to decide the action to perform (Fig. 4). This results in a set of decision tables with their associated weights that define probabilistic state machines (Novák 2009) and influence diagrams (Howard & Matheson 2005) for representing the causalities and effects of the agents' actions (Pearl 1994; Darwiche 2009; Pearl 2000)
- In the fifth step, we determine the attributes of the agents that need to consider measurement or belief uncertainty due to the imprecision in the measuring tools or unreliable sources, and how they get propagated.
- Finally, we specify the degree of randomness of the behaviour of the agents and of their environments, capturing those unpredictable decisions or unforeseen events.

We have evaluated this approach with different applications. In this paper we will illustrate the proposal with two case studies. The first corresponds to a pedestrian crossing system in which cars and pedestrians cross the street in diverse situations, trying to minimize hard brakes, near-collisions and accidents. The second emulates a war of tanks chasing and shooting at each other (or trying to escape in case of low ammunition or numerical inferiority). These examples are described below, as well as how the uncertainty in the behavior of its agents can be modeled and simulated with UML and OCL tools.

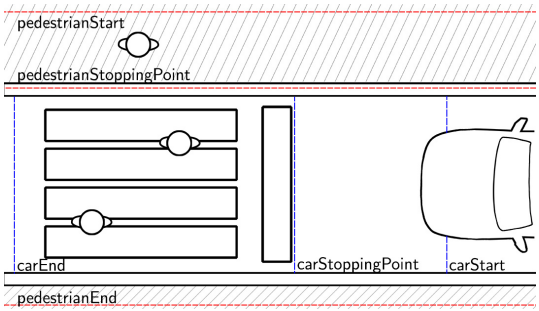

Figure 1 Graphical representation of the Crosswalk system.

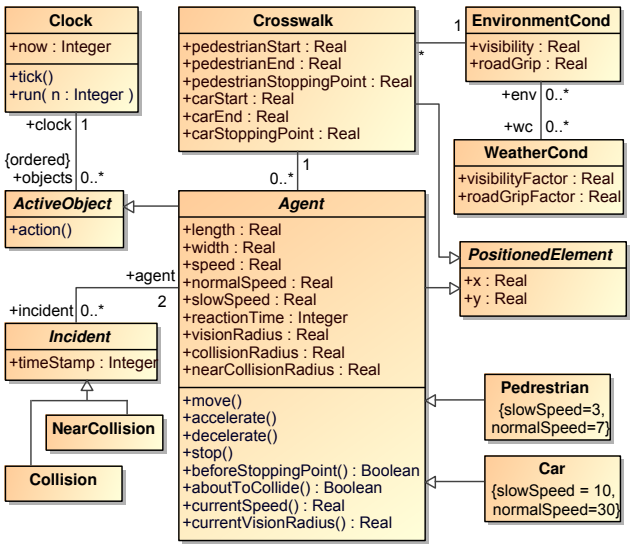

Figure 2 The Crosswalk system metamodel.

3.2. Motivating Example: A crosswalk system

Suppose that we want to model a crosswalk system that emulates the behavior of two types of agents: car drivers and pedestrians. The structure of the crosswalk system is depicted in Fig. 1.

Each agent has a position, given by its planar coordinates, its dimension (length and width) and its current speed, which can be either zero (when stopped), slow or normal. Their age, experience in manoeuvring, vision range, or mental stress influence the decision to take an action. Moreover, there are environmental conditions (class `EnvironmentCond`) such as road visibility and road grip which are influenced by weather conditions class `WeatherCond`, which represents, e.g., fog, rain, ice or darkness (Czarnecki & Salay 2018; Leibowitz et al. 1998). These factors impact the crosswalk system and could cause collision or near-collision events (Sanders 2015). Being common to all agents at the same crosswalk, the environmental conditions are associated to `CrossWalk` objects, and then used by each individual agent to compute its own speed or visibility.

The crosswalk system is not guided by any traffic lights, and hence the agents have to decide intuitively to take an action. As shown in Fig. 2, Class `Crosswalk` defines several attributes indicating the agents starting point, stopping point, and end point. The stopping point defines where the agent should stop before entering the crosswalk. The stopping point of a car coincides

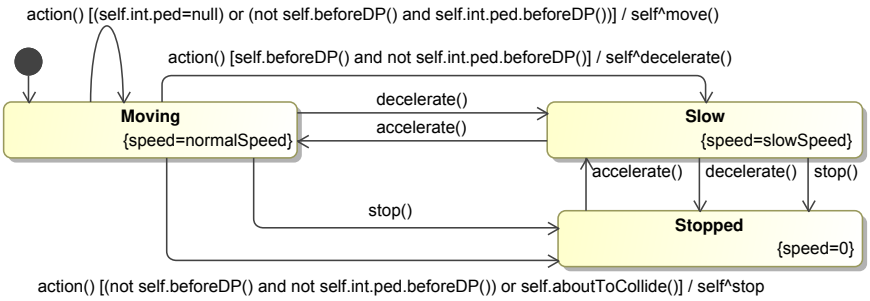

Figure 3 Excerpt of the state machine for `Car` objects in the Crosswalk system.

with the crosswalk's stop line. The stopping point of a pedestrian is at the end of the sidewalk, before starting to cross it (see Fig. 1). Agents implement a query operation (`beforeSP()`) that decides if the agent has not yet passed its stopping point. Another operation (`aboutToCollide()`) determines if the agent is about to collide with another one. Whenever a collision occurs, an instance of class `Incident` is created.

Time is modeled by a `Clock` object. On every `tick` (whose resolution can be defined by the user, in our simulations we have used 0.1 seconds), it invokes operation `action()` on all active objects associated to the clock. When an agent receives an `action()` operation, it decides what to do next. Possible options are: `move()` at the current speed until it receives a new `action()` operation; `accelerate()`, which increases the speed and then moves at the new speed; `decelerate()`, which decreases the speed and then moves; or `stop()`, which suddenly brakes and changes the speed to 0. In the `Slow` state, it is the same for the pedestrian to decelerate as to stop; in the case of a car, when `stop()` is invoked, it first moves with half speed and then stops. Accelerating when moving at normal speed does not have any effect. Likewise, decelerating, moving or stopping in the `Stopped` state do nothing. For simplicity, the intersection is located at point (0,0), cars move along the X-axis from right to left, and pedestrians move top-down along the Y-axis. Methods `currentSpeed()` and `currentVisionRadius()` are affected by the environmental conditions. They recalculate the current speed and vision radius of the agent according to the current road visibility and grip.

A common way to specify the behavior of these types of real-time systems is by means of state machines (Harel 1987; Object Management Group 2015). Operations define the signature of the events accepted by the objects, and state machines define how the state of each object changes upon receipt of an operation. For example, Fig. 3 shows an excerpt of the state machine for `Car` objects. In addition to showing the effects of operations `move()`, `accelerate()`, `decelerate()` and `stop()`, it specifies how the object decides what to do when an `action()` operation is received (only shown for the `Moving` state). Roughly, if the car is moving and there is no pedestrian in sight, or both the pedestrian and the car are before the crosswalk stopping points, the car keeps moving; if the car is before its stopping point and the pedestrian has already passed it, the car slows down; and if both the car and the pedestrian have passed their stopping points, or they are about to collide, the car stops. Similar rules can be defined for states `Slow` and `Stopped`. They are not displayed in Fig. 3 but summarized in the state-transition tables 1 and 2. The left columns show the *situation* of the agent when the `action()` operation is received, requesting it to decide what to do next. Depending on the situation and on the current agent state (`Moving`, `Slow` or `Stopped`), its reaction can be different.

Table 1 Car decision table.

	Car state		
Situation of car	Moving	Slow	Stopped
C1. No pedestrian in vision radius	`move`	`accel.`	`accel.`
C2. Car before SP, ped. before SP	`decel.`	`move`	`accel.`
C3. Car after SP, ped. before SP	`move`	`accel.`	`accel.`
C4. Car before SP, ped. after SP	`decel.`	`move`	`accel.`
C5. Car after SP, ped. after SP	`stop`	`stop`	`stop`
C6. Car in near collision with ped.	`stop`	`stop`	`stop`

Table 2 Pedestrian decision table.

	Pedrestian state		
Situation of pedestrian	Moving	Slow	Stopped
P1. No car in vision radius	`move`	`accel.`	`accel.`
P2. Ped. before SP, car before SP	`move`	`accel.`	`accel.`
P3. Ped. after SP, car before SP	`move`	`accel.`	`accel.`
P4. Ped. before SP, car after SP	`decel.`	`decel.`	`stop`
P5. Ped. after SP, car after SP	`move`	`accel.`	`accel.`
P6. Ped. in near collision with car	`stop`	`stop`	`stop`

We have modeled the Crosswalk system in UML and OCL, employing the tool USE (Gogolla et al. 2007) that provides an action language called SOIL (Büttner & Gogolla 2014) that allows modelers to create instances and links, perform assignments of values to attributes, and invoke operations on objects. This way, the system can be simulated. Further validation and verification tests are also available with USE, as described in (Gogolla et al. 2018). However, one of the limitations of these specifications is that they are deterministic, failing to account for both the stochastic nature of the agents' behavior and the uncertainty aspects of these kinds of systems.

The following subsections describe how we propose to model such a behavior, illustrating it using the Crosswalk system.

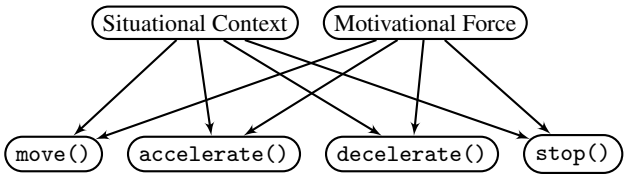

Figure 4 Basic actions and their causes.

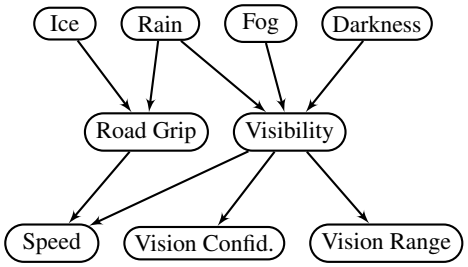

Figure 5 Environment influence on agent parameters.

3.3. Modeling the basic behavior: actions and forces

First, we need to identify the **actions** that each agent can perform during the system execution. In the Crosswalk system, the actions that both car drivers and pedestrians can perform are: `move()`, `accelerate()`, `decelerate()` and `stop()`.

Second, we need to specify *how* each agent decides to perform an action, as well the factors that motivate such a decision, i.e., the **forces** that influence the agents' decision process. To specify these influences we will use both the *Situational context* and the *Motivational Force* of each action. The first one defines the situation in which an agent is, something that complements the agent's state machine to decide the action to perform (cf. tables 1 and 2). For example, one possible situation of the car is before the stopping point and with other agent inside the crosswalk (situation C4). In contrast, the Motivational Force defines the motivations of an agent to decide the action to perform next, ranking them according to their utility (Neumann & Morgenstern 1953) and how they satisfy the agent's needs (Nassal & Tichy 2016). For example, when a pedestrian is in a real rush, it will prioritize acceleration over all other actions.

Figure 4 shows an Influence Diagram (Howard & Matheson 2005) with these relations. This diagram is a generalization of Bayesian and Causal Networks to achieve not only probabilistic inference but also decision making, following the *maximum expected utility* criterion (Neumann & Morgenstern 1953).

Finally, we also need to specify how the *environment and personal conditions* influence the **parameters** of the actions. For example, difficult weather conditions like heavy fog or rain can affect the visibility of the agent, and also the road grip, which in turn influence the speed of the agent and its visibility. This is shown in the influence diagram depicted in Fig. 5.

These elements are discussed in detail in next subsections.

3.4. Situational context

In our proposal, the *Situational Context* defines the possible situations in which an agent can be, when having to decide the action to perform next, as well as their associated probabilities. In the Crosswalk system, the Situational Context of the car is defined by situations `{C1...C6}` described in Table 1.

As previously mentioned in Sect. 3.2, these tables define the state machines that specified the agents' behavior. Based on the situation the object is in, and the current state of the object (`Moving`, `Slow`, `Stopped`), the tables determine the next action to perform.

In order to define the influence of the Situational Context on each action (the arrows that connect the *Situational Context* node with each *action* node in the influence diagram shown in Fig. 4), our proposal extends these traditional state machines in two ways. First, we replace them with probabilistic decision tables that determine, for each situation in the agent's Situational Context, the probability of each possible transition. In other words, they become probabilistic state machines (Novák 2009). And second, probabilities are also assigned to situations, to represent the fact that sometimes it is difficult to decide precisely in which situation the agent is (see below).

For example, Table 3 shows the probabilities associated with each possible action of a `Car` agent, when it is in the `Slow` state. Basically, it extends column 3 of Table 1, which determined for each situation the (only) action to take by the Car, with the set of possible actions that can be performed by the Car, and their associated probabilities.

Second, we assign probabilities to the situations, which represent the likelihood of the agent to be in each one. These probabilities are normally due to the incorporation of measurement uncertainty (Bertoa et al. 2020) into the models, because comparing two distances with imprecision no longer returns a Boolean value, but a probability (Burgueño et al. 2018), which gets propagated when operating with imprecise attributes. In this way, instead of being able to crisply decide whether we are close or not to a given point, or to another object, we can only *estimate* how close we are to it. This naturally reflects the human incertitude in evaluating distances or times, and allows us to model the imprecise behavior, or *hesitation* (Jay et al. 2020), that occurs when a pedestrian or car evaluates whether it is 'close enough' to the intersection to stop or cross.

Therefore, in our proposal a Situational Context will be defined by a tuple of Real numbers, one for each possible situation, which represent their associated probabilities. The sum of the elements of the tuple must be 1. For example, at one moment in time, the Situational Context of the car agent can be defined by the tuple `{c1=0,c2=0.95,c3=0.05,c4=0,c5=0,c6=0}`, meaning that the evaluation of the car situation has dictated that the car is before the stopping point with a likelihood of 95%, and after the stopping point with a likelihood of 5%.

The probabilities of each situation are considered together with the probabilities in the probabilistic transition tables. This computes the degree of influence for the next action to be performed by the agent.

3.5. Motivational Force

As shown in the influence diagram in Fig. 4, agents also take into account the *Motivational Force* of each action (Nassal & Tichy 2016), which computes the maximum benefit (*valence*) of all possible results of the action, according to the agent's

Table 3 Probabilistic `Car` decision table, in the `Slow` state.

	Car state: `Slow`			
Situational context of car	`move`	`accel`	`decel`	`stop`
C1. No pedestrian in vision radius	0.1	0.9		
C2. Car before SP, ped. before SP	0.8	0.1	0.1	
C3. Car after SP, ped. before SP	0.2	0.7	0.1	
C4. Car before SP, ped. after SP			0.9	0.1
C5. Car after SP, ped. after SP			0.1	0.9
C6. Car in near collision with ped.				1

personal preferences and system goals. The benefit of each result depends on (1) the *instrumentality* of the result, i.e., the chances it has to lead to a valid outcome (normally determined by the goals of the system, or the objectives defined by the agent's owner), and (2) the personal priorities and *needs* of the agent (which depend on the own agent's goals, skills and personal preferences).

With this, weights (i.e., probabilities) can be assigned to the actions that the agent can perform, according to their Motivational Force. Such weights will be combined with the Situational Context, using the corresponding influence diagram, to define the final probability of each action.

In general, computing the Motivational Force of the actions of an agent is not easy because it depends on social, psychological and personal factors, and it may also imply non-trivial resolution of conflicting interests and forces. In this paper we propose the use of a high-level specification of such forces, based on goals and responsibilities (Marron et al. 2020). Thus, each agent will have a set of high-level goals to achieve, and a set of responsibilities to comply with. In this case, the goal of all agents of the Crosswalk system is the same: "to cross the street as soon as possible and in a safe way, in order to reach their final destinations." The responsibilities are to avoid collisions with other agents, and to obey the traffic laws — e.g., the maximum speed when approaching the crosswalk. Each action of an agent will contribute to the agent's goals and responsibilities in different ways, and therefore each agent will assign different weights to each action, depending on its contributions to the agent's current motivational forces. For example, a *reckless* pedestrian will give the maximum priority to the goal of crossing the street, ignoring the rest of the forces, even the possibility of collisions. A *distracted* driver may sometimes ignore that he has passed the stopping point,e.g., when texting while driving, but will always comply with the responsibility of avoiding collisions. An *attentive* agent will always follow the rules, even when it implies waiting for a long time until the crosswalk is free. As with humans, profiles are never absolute; it can happen that an agent is 20% reckless, 30% distracted and 50% attentive. These weights contribute to the final decision.

This means, e.g., that an attentive agent will assign the same weight to each action for each force, without imposing any personal choice, while a reckless driver will always try to accelerate, assigning a weight of 0 to the rest of the possible actions, in all agent states. Table 4 shows examples of weights assigned to the possible actions of a car, in the `Slow` state, with different motivational forces.

Table 4 `Car` Motivational Forces, in the `Slow` state.

	Car state: `Slow`			
Type of agent depending on its motivational force	`move`	`accel`	`decel`	`stop`
Reckless agent		1.0		
Distracted agent	0.2	0.6	0.1	0.1
Attentive agent	0.25	0.25	0.25	0.25

In general, obtaining these weights can be a difficult task, but it is something that falls outside the scope of our work as it depends largely on the type of application. Here we will focus on how to this information can be used to enrich the software models and their simulations.

3.6. Combining Situational Context and Motivational Force

Figure 6 shows the UML class diagram of the Crosswalk system, extended with behavioral uncertainty. It is modeled with USE (Gogolla et al. 2007) because this tool already provides an extension of the UML and OCL type system to deal with uncertain datatypes (Bertoa et al. 2020).

The new attributes `situationalContext` and `decision` of class `Agent` store, respectively, the values for its current situational context, and the probabilities associated with each action. With this, the combined probabilities of each possible action will be computed using the influence diagram shown in Fig. 4, based on the weights given to the situational context and motivational force, stored in the attributes `situationalContextWeight` and `motivationalForceWeight` of class `Behavior`. These weights (whose sum is 1) represent the relative importance the agent assigns to the corresponding factors, i.e., the confidence assigned to each one. The values of these tables can evolve, as discussed later in Sect. 4.4.

Some additional inputs are needed for the computation of the final probabilities associated with each action:

- The probabilistic decision tables defined for the states of each agent (e.g., Table 3). These are stored in class `SituationalContextTables`.
- The probabilities that the agent has to be in each of the situations defined in its Situational Context, also stored in class `SituationalContextTables`.
- The type of agent depending on its motivation, e.g., 20% reckless, 20% distracted and 60% attentive. (Attributes `recklessMood`, `distractedMood` and `attentiveMood` of class `MotivationalForce`).
- The weights assigned to the possible actions of the agent, depending on the type of agent and on its current state (e.g., Table 4), also stored in class `MotivationalForce` (not shown in Fig. 6).

According to the maximum expected utility criterion (Neumann & Morgenstern 1953), the action with the highest probability (i.e., the one with the highest value in the `decision` tuple attribute of class `Agent`) will be typically performed by the agent. In addition, agents record their decisions using traces, which provide logs that can be used after the simulation to visualize and/or analyze the system execution.

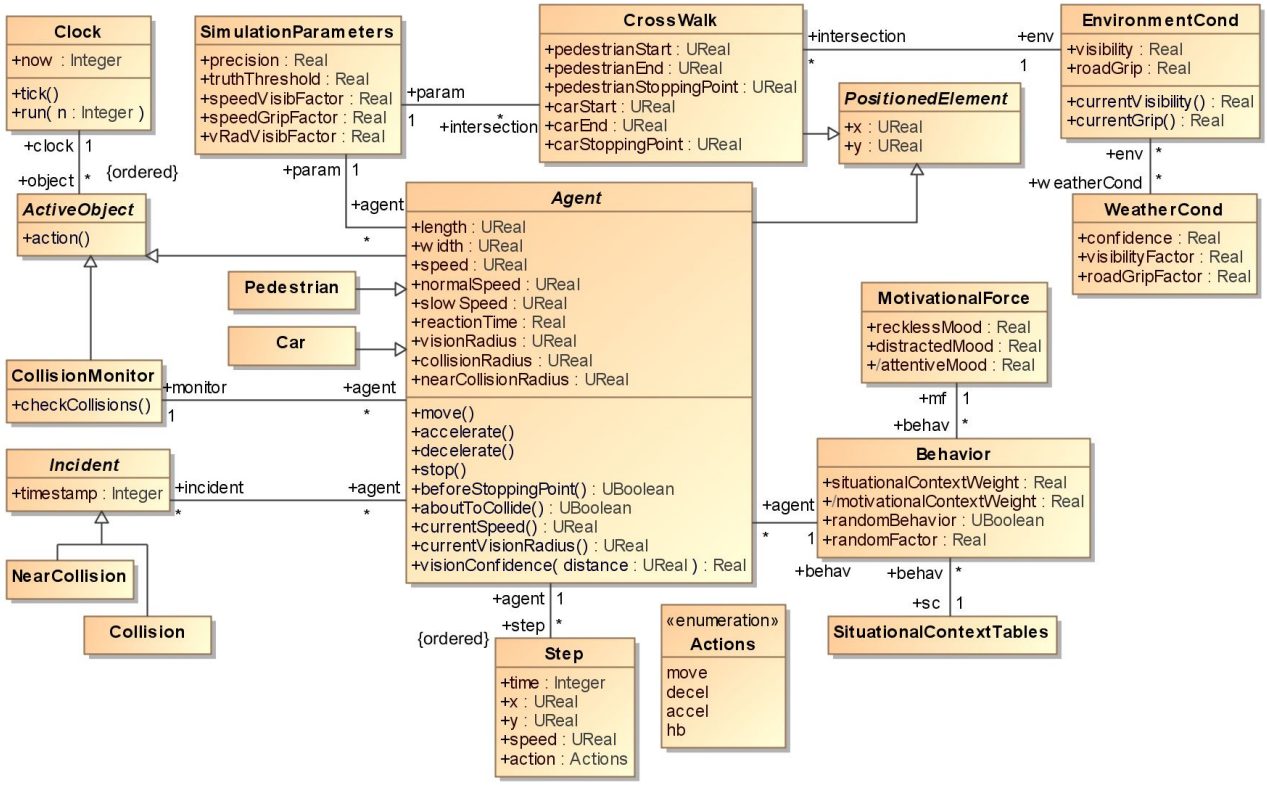

Figure 6 The Crosswalk system metamodel, extended with behavioral uncertainty.

3.7. Object attributes and simulation parameters

Once the agent has decided the action to perform during the next time interval, we need to set the parameters of such actions, e.g., the speed at which the agent will move. For the estimation of these parameters we need to consider both the uncertainty factors that affect their values, and the degree of influence of the environment conditions on the corresponding attributes (Fig. 5).

3.7.1. Uncertainty in parameter calculation Object attributes are used in models to capture their properties, such as physical dimensions (length, width, collision radius), capabilities (maximum speed, reaction time), and current state (position, speed, vision radius). These variables are normally typed using the UML and OCL primitive datatypes, which assume precise values. However, models of physical systems need to account for the inherent uncertainty of their elements and environment, and therefore the need for notations that consider such uncertainty. Here we will focus on the following types of uncertainty (Czarnecki & Salay 2018; Troya et al. 2021):

- Measurement uncertainty, due to, e.g., unreliable or imprecise measuring instruments, mechanical tolerance, or lack of visibility conditions. This leads to imprecise calculation of distances or speeds, and of the comparison of their values (Salay et al. 2020).
- Occurrence uncertainty, about the presence (or not) of objects in the system. It can be due to, e.g., lack of visibility or adverse weather conditions, which make the agent be unsure about the presence of a pedestrian or any other object in the road (Phan et al. 2019).

To handle these uncertainties we make use of an extension of UML and OCL datatypes to represent and propagate measurement uncertainty according to the international metrology standards (JCGM 100:2008 2008), and of its realization in the modeling tool USE (Bertoa et al. 2020). Type `UReal` extends type `Real` with measurement uncertainty (e.g., `UReal(3.5,0.01)` represents the uncertain `Real` 3.5 ± 0.01), while type `UBoolean` provides the Probabilistic extension of Boolean logic, where probabilities represent the likelihood (confidence) of a predicate to be true; e.g., `UBoolean(true,0.99)` represents a confidence of 99%. The extended type system provides the automatic propagation of uncertainty through the numerical, comparison and logic operators.

3.7.2. Influence from external factors To define the relationships between the different object attributes, and the effects that external factors have on them, we use influence diagrams such as the one shown in Fig. 5.

The concrete influence of some attributes on others can be expressed by OCL expressions and operations that implement the relations defined in the influence diagram. For example, the relationship between the weather conditions and the visibility in the crosswalk and the road grip can be specified as follows:

```
currentVisibility():Real = visibility *
 self.weatherCond->iterate(wc : WeatherCond;
  r: Real = 1.0 | r * wc.visibilityFactor))
currentRoadGrip(): Real = roadGrip *
 self.weatherCond->iterate(wc : WeatherCond;
  r:Real = 1.0 | r * wc.roadGripFactor))
```

These variables are real numbers in the range [0..1] where 0 means no visibility/grip and 1 means perfect visibility/grip. The expression used to derive their values simply modifies the initial road value by a factor that is obtained by multiplying the factors of all the weather condition that currently apply to the road (e.g, ice, fog, rain or darkness).

Calculating the variations of other attributes depending on the environmental conditions is a bit more complex, since they use simulation parameters. For example, the way in which the speed of the agent is modified by the visibility and road grip can be specified as follows:

```
currentSpeed() : UReal =
  let visibility : Real =
        self.int.env.currentVisibility() in
  let roadGrip : Real =
        self.int.env.currentRoadGrip() in
  let fv:Real=self.int.param.speedVisibFactor in
  let fg:Real=self.int.param.speedGripFactor in
  self.speed*(fv*visibility + fg*roadGrip +
              (1 - fv - fg))
```

Next, operation `currentVisionRadius()` adjusts the vision radius depending on the current visibility:

```
currentVisionRadius() : UReal =
 self.visionRadius*self.int.param.vRadVisibFactor
```

Finally, operation `visionConfidence()` returns the confidence, expressed as a `Real` number in the range [0..1], that we assign to an object that an agent sees at a certain `distance`. It will be 0 is the object is outside the vision range of the agent. Inside its vision range, the confidence will be 1 if it is close, and starts diminishing as the object reaches the vision range limit. This is determined by the value of parameter `vRadVisibFactor`.

```
visionConfidence(distance:UReal): Real =
 let VR :UReal = self.currentVisionRadius() in
 let th :Real = self.int.param.truthThreshold in
 let vf :Real = self.int.param.vRadVisibFactor in
if (distance>VR).confidence()>=th then 0.0
else if (distance<VR*vf).confidence()>=th then 1
     else 1.0 - (distance/R).value()
     endif
endif
```

In this expression, note the use of `truthThreshold` to decide if a condition expression is true or false. Given that our measurement and logical expressions incorporate uncertainty, thresholds are needed to make decisions.

3.8. Representing Random Behavior

Finally, another aspect that cannot be neglected is related to the unpredictable or even random behavior of agents. For example, a pedestrian who suddenly stops, or a car that accelerates with no apparent reason (Kraaier & Killat 2008; Oberkampf et al. 2002). This is modeled by a final (and optional) phase that randomly modifies the decision of the agent about its next action.

Attribute `randomBehavior` of class `Behavior` determines the likelihood of the agent to behave randomly, while the `randomFactor` attribute determines the degree of change allowed for the final decision. For example, if `randomBehavior=0.2` and `randomFactor=1`, the agent will exhibit a completely random behavior on 20% of its decisions. Likewise, if `randomBehavior=0.1` and `randomFactor=0.3`, one of the possible actions of the agent (arbitrarily chosen) will be added a 30% chance of being selected, in 10% of the agent's decisions. Of course, other alternative algorithms to implement random behavior could be applied instead.

Note that this last step attempts to capture unpredictable behaviors or unplanned circumstances such as the agent suddenly changing its mind or simply malfunctioning, which are generalized here as random actions. More careful reasoning or more detailed analysis of unexpected behaviors could be used to eliminate the randomness of the unknown, unplanned, or apparently surprising behavior of the agents when they are of an epistemic nature. Other aleatory uncertain behaviors, such as a car engine suddenly breaking or a pedestrian having a stroke while crossing, still need to be treated randomly.

3.9. Second motivating example: A Tank Warfare

The second application we developed to demonstrate and evaluate our proposal is a survival game, a *Tank Warfare*, in which two teams of tanks fight each other. The last team standing is the winner. Different types of uncertainty are involved in this application, which must be taken into account to simulate the system more realistically and faithfully.

First of all, the tank occupants' uncertain behaviour needs to be considered, as well as their profiles and motivations, which may be more or less bold or cautious. Measurement uncertainty also plays an important role, since this type of vehicle's navigation and firing systems usually have limited precision. The visibility conditions of each tank and its environment should also be taken into account. Finally, there is always a degree of randomness in the final decisions of each tank crew.

3.9.1. Game specification

At any given time, tanks can do any of the following: they can *shoot* at another tank, *move* in any direction or do nothing. As in the crosswalk simulation, time is modeled by a `Clock` object whose `tick()` operation invoke the `action()` method of all tanks.

To determine if the tank can execute an action, there are three main parameters in the game. The first is `health`; every time a tank is shot, this value is decreased. When the value reaches 0, the tank is destroyed. The second parameter is `power`, which is consumed each time a tank executes an action. Tanks can not perform actions if this value reaches 0. The third parameter is `ammo`, which indicates the number of cannonballs available. Reloading is not possible in our simulation, so once the tank fires all of its bullets, it cannot shoot any more.

3.9.2. Method application

The first step is to specify the possible actions that the tanks can perform, together with the state machines that specify their (deterministic) behavior. Given the two main actions mentioned in the previous section (`move` and `shoot`), we defined five compound actions: `escape()`, which moves in the opposite direction to the nearest enemy tank; `shoot()` at the closest enemy tank; `chaseEnemy()`, which moves towards the nearest enemy tank; `joinAlly()`, which moves towards the nearest ally tank; and `stop()` to save power.

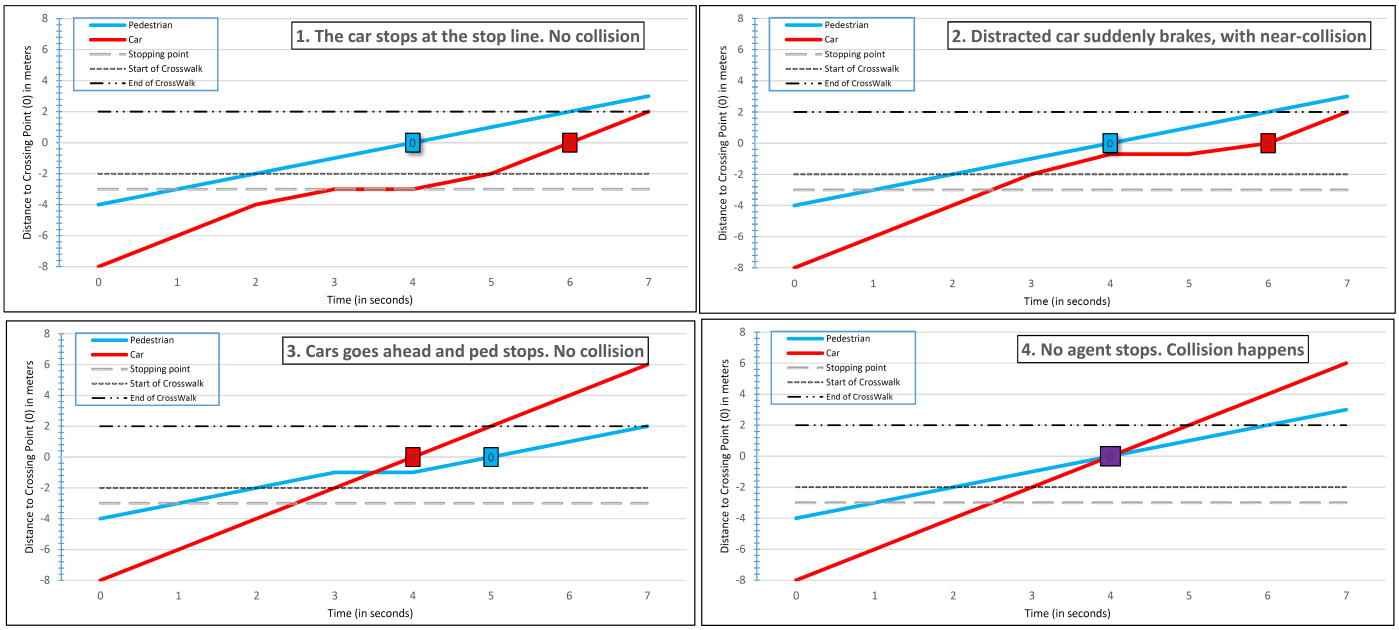

Figure 7 Examples of simulations showing different scenarios. Square dots represent the times when the agents cross the midpoint of the crosswalk. Collisions only occur if they cross at the same time.

In the second step, we define the *situational context* by specifying the possible situations in which the agents can be, depending on their subjective interpretation of the environment. In this case, we have defined 6 situations and 2 states in which a tank can be, given the number of *allies* and *enemies* surrounding it, and its vulnerability level.

A tank can be in any of these two states: *Low Vulnerability* or *High Vulnerability*. The current state depends on the value of attribute `vulnerability`, which is calculated by aggregating the values of `power`, `health`, and `ammo`. Attribute `vulnerabilityBound` defines the threshold that determines the current state. In the *High Vulnerability* state, the tank decisions will be more reckless, while in the *Low Vulnerability state* its decisions will be more conservative. For example, in the second *situational context*, a vulnerable tank will escape because there are no allies nearby to help, while a non-vulnerable agent will chase the enemy to engage in a fight.

The third step identifies the *motivational forces* that influence the agent's decisions. In our case, we consider two types of behaviour depending on the strategy chosen to play the game. The first one is the *bold* strategy, in which the tanks will prioritize the most aggressive decisions over all others. For example, they will be willing to shoot at enemy tanks even in a minority situation. In the second strategy, called *cautious*, the tanks will prioritize conservative choices to try to survive at all costs. If tanks feel that they can lose because they are in a minority, they will choose to escape.

The fourth step assigns weights to the *situational context* and *motivational force* models. In our examples, these weights were assigned by estimating the possibilities of the logical actions that a player would take, but they can come from any source such as real sampled data or machine learning simulations.

The fifth step incorporates the possible *measurement* and *belief* uncertainty to the model. In this example, we have defined some uncertain attributes regarding the mobility of the tank, the power consumption process or its vulnerability because they are all related to physical parameters. We also considered occurrence uncertainty about the presence of objects in the system when some tank is unsure of the existence of another one.

Finally, some randomness in the tanks' behaviour can also be added to emulate the unpredictability of human behaviour.

The UML and SOIL specifications of this and the previous system, as well as the results of several simulations with various configurations can be found in the paper's companion website.[1]

4. Analysis

Once we have described our proposal for modeling the behavior of agents, in this section we discuss the types of analyses that can be performed with it.

4.1. Simulation

Although in theory upfront design-time verification techniques could be used to analyze their properties, research has shown that the complexity of these systems makes static analysis practically unfeasible (Helle et al. 2016; Koopman & Wagner 2016). In contrast, run-time techniques, such as simulation or monitoring, can still provide very relevant information at a lower cost (Babikian 2020).

One of our reasons for specifying our models with the tool USE is that it provides a high-level textual action language, called SOIL (Büttner & Gogolla 2014) that enables the behavioral specification of UML models, as well as its execution. In particular, SOIL extends the OCL notation with traditional imperative constructs, including the creation of instances and

[1] http://atenea.lcc.uma.es/projects/ModelingUncertainBehaviours

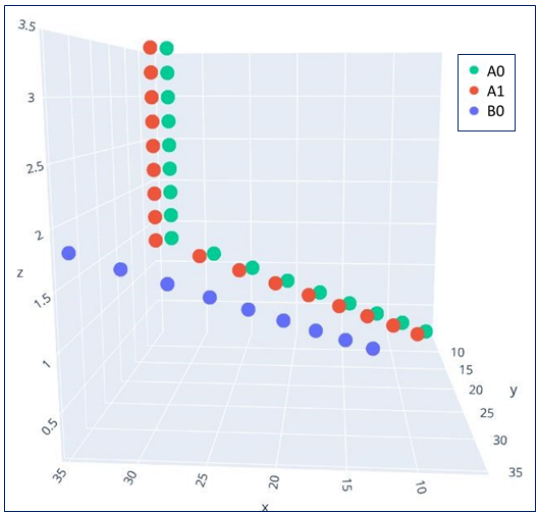

Figure 8 Visualization of a simple simulation with 3 tanks.

links, the assignment of values to attributes, and the imperative specification of bodies of operations. Probability distributions and random numbers are also supported (Vallecillo & Gogolla 2017). This is important to create agents with different behavior by simply changing the values of their decision tables and motivational forces, and to generate stochastic simulations.

With USE, our simulation framework basically consists of a simulation loop in which the `Clock` object invokes method `action()` on all active objects (cf. Sect. 3.2), which perform the actions specified in those methods by evaluating OCL and SOIL expressions.

For illustration purposes, Fig. 7 graphically displays the resulting traces of four simulations of the *Crosswalk* system, showing different scenarios where one car and one pedestrian try to cross the street. Similar graphics can be used to visualize the effect of changes in the simulation parameters, the environment conditions, etc. These charts can be developed from the agents' traces, which can be directly obtained from the USE tool and easily converted into csv files.

Similarly, we can simulate the Tank Warfare application. For example, let us consider one team with two tanks (`A0` and `A1`) and another team with one (`B0`), see Fig. 8. Initially, tanks `A0` and `A1` are in the *Low Vulnerability* state and have a *bold* personality. In contrast, `B0` is in the *High Vulnerability* state and has a *cautious* personality. Their situational contexts are different too: while tanks `A0` and `A1` are in superiority (they have allies in their vision range), tank `B0` is alone (no allies around) and has enemies close by. We can see how during the simulation, tanks `A0` and `A1` start chasing `B0`. After eight time units, `B0` is in their shooting range. Then, `A0` and `A1` shoot at it, and tank `B0` is destroyed.

4.2. Monitoring system properties of interest

Having the possibility to simulate the system with different parameters is very useful to understand how the system works, and to obtain information about the possible behavior of different types of agents (reckless, attentive or distracted; with better or worse reaction times and vision ranges, etc.), and under varying

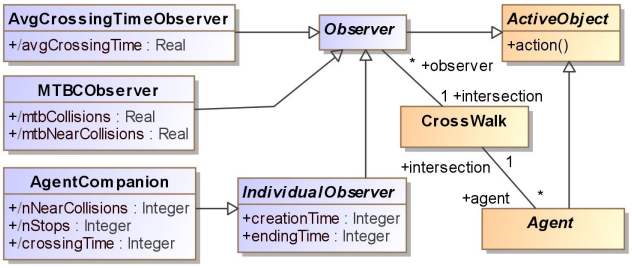

Figure 9 The Crosswalk metamodel with Observers.

situations and circumstances (e.g., diverse weather conditions, different precision of measuring instruments, etc.). The use of high-level modeling notations also enables the specification and analysis of some system properties of interest, such as the mean-time between collisions in varied weather conditions, the average crossing times for different agent profiles, or the number of collisions depending on the congestion of cars and pedestrians—in the Crosswalk example—, or the effect of more calibrated and precise guns, better trained and motivated crews (hence more confident, less cautious), or longer-lasting batteries in the Tanks example.

To do so, we make use of *observer objects* (Troya et al. 2013), whose purpose is to monitor the execution of the system and the state of other objects. Each observer is in charge of one or more properties of interest, and has a well defined behavior. Observers can also monitor other observers.

Figure 9 shows an excerpt of the Crosswalk system metamodel, extended with three observers (the newly added classes are shaded darker). Two observers monitor, respectively, the average crossing time and the mean time between collisions of the system. The individual observer `AgentCompanion` computes the number of times that its associated agents had to stop; the number of near collisions they were involved in, and their final crossing times.

The advantage of this approach is that all these properties are stored in the observers' attributes and they can be specified at the model level, using OCL expressions. For example, the following listing shows the derivation expressions for the attributes of class `AgentCompanion`, which is the individual observer defined for system agents.

```
class AgentCompanion < IndividualObserver
attributes
 nStops:Integer derive:
  self.agent.step->select(action=#hb)->size()
 nNearCollisions:Integer derive:
  self.agent.incident->select(i|
        i.oclIsTypeOf(NearCollision))->size()
 crossingTime: Integer derive:
   self.endingTime - self.creationTime
```

Being derived attributes, their values are always updated. Using the attributes of the agents' individual observers, the system observers can similarly specify theirs.

4.3. Temporal properties

Observers can also be used to monitor temporal properties of the system, enabling their dynamic analysis. For example, a typical

temporal requirement expressing a *liveness* property may state that any stopped agent waiting to cross, will eventually move on. To specify this property in a simulation environment, individual observers use their agents' traces to compute the time they have remained stopped, and a system observer checks that no agent is stopped more than the maximum acceptable delay.

```
class StopsObs < IndividualObserver
attributes
 stoppedTime : Integer derive:
   let ts : Integer = (self.agent.step->
      select(action=#hb)->last()).time in
   let tm : Integer = (self.agent.step->
      select(action=#accel)->last()).time in
   (tm - ts).max(0)
end
class WatchdogObserver < Observer
attributes
 maxAcceptableWaitingTime : Integer init: 6000
 limitExceeded : Boolean derive:
   StopsOb.allInstances->select(stoppedTime >
     self.maxAcceptableWaitingTime)->notEmpty()
end
```

Different strategies can be considered when a property is violated. For instance, if we want to stop the simulation when an agent remains in the `Stopped` state more than the specified threshold, a postcondition can be added to the observer's `action()` operation in order to check that `limitExceeded` is `false`. The USE engine stops the execution whenever a pre- or postcondition is violated.

In turn, a typical *safety* temporal property may state that no collision happens. This can be checked by a system observer that monitors that no instances of class `Collision` occur.

These examples illustrate how temporal properties that use *always* and *eventually* operators can be specified and monitored with our proposal. We expect that more complex temporal properties can be similarly specified with observers, and thus a more thorough study is planned as part of our future work.

4.4. Self-adaptive features

Observers can also be used to implement self-adaptive features, by using the values of the observers' attributes to decide changes in the system parameters. For example, we could specify an observer that monitors the average time agents remain stopped at a crosswalk, and if this value goes above a certain threshold, the observer can automatically increase the normal speed of the agents until the situation goes back to normal. Similarly, if the number of near collisions in a crosswalk increases, limiting the maximum speed of the agents can help reducing the collisions. We can use our behavioral specifications combined with observers to perform simulations able to check whether our assumptions are correct or not, and to evaluate the effect of different thresholds values on the system overall behavior.

Again, the use of UML and OCL models simplifies the high-level specification of such self-adaptive behavior, enabling the cost-effective analysis and simulation of systems, compared to other solutions that use lower-level agent platforms.

4.5. Evaluation

This section describes the evaluation we have conducted on our proposal.

Simulation. We performed a series of simulations to analyze the effect of incorporating uncertainty into our models, and whether it results in more informative simulations. We defined a set of scenarios, each one describing a specific situation.

In the Crosswalk example, we distinguished first between *normal* and *peak* hours, as representative examples of extreme situations. The former case emulates a system working in normal conditions and where all agents are attentive and incidents should not happen; the latter scenario represents a system with reckless and distracted agents where incidents might be common. More precisely, we assumed that in peak hours, the speed of the agents was 5% higher than in normal hours, and 20% of the pedestrians walked in groups of 2-4 persons forming lines. We included 0.2% of reckless agents (who cross rashly, ignoring traffic signals and the presence of other agents) and 4% of distracted ones. In contrast, reaction time was 10% faster in rush hours, since all agents tend to be more alert. Weather conditions was the second dimension that we considered, affecting the visibility and the road grip. It could be either good visibility, or bad visibility (50%, emulating a poorly illuminated road at night), or bad road grip (50%, as in an slippery road), or both.

Among other things, we wanted to understand the effects of the different parameters (speed, visibility, road grip and measurement uncertainty) on the system behavior. Our aim was to analyze the situational and the motivational context that help providing a reasonable decision. In our scenario, this could mean assessing the impact of whether installing a street light close to the crosswalk could mitigate the incidents in dark nights, or the effect of fixing the road infrastructure on the reduction of the incidents by improving the road grip by 30%.

We simulated the initial *crisp* system that does not use our proposal (Fig. 2) and then we used our approach, with different levels of Measurement uncertainty (No, Small, Med and High), depending on the precision of the measuring instruments (0, 1cm, 2.5cm, 10cm) and the degree of confidence (100%, 99%, 95% and 85%). Table 5 shows the results for every 1000 crossings. Each cell displays the number of hard brakes, near-collisions and collisions that occurred during the simulation. Hard brakes are important because they mean abrupt stops, normally to avoid near-collisions or collisions. In all cases, agents were generated with different reaction times and vision ranges, following Normal distributions $N(0.7, 0.1)$ and $N(30.0, 2.0)$, respectively (Makishitaa & Matsunaga 2008). Each simulation was run 5 times, and the table shows the average results.

The resulting figures show that the parameters used for the agents' speed and reaction times, and well as the distances defined for the intersection, ensure safe crossings in normal conditions. However, increasing the speed of the agents and introducing reckless and distracted agents start causing incidents. We can also observe the effects on the safety conditions of the crosswalk of the main simulation parameters: road grip, visibility, level of measurement uncertainty (precision and confidence), and the speed of agents. Using our models, we can check, for instance, that if we want to reduce the number of incidents during peak hours, the most effective measure would be to decrease the allowed speed limit by 15%, which would permit agents to avoid collisions.

Table 5 Simulation results for 1000 crossings (Figures indicate number of Hard brakes, Near-collisions and Collisions).

		Good W.	Bad Grip	Bad Vis.	Bad G-V.
Crisp	Normal	[– | – | –]	[– |.04| –]	[– |.01| –]	[– |.05|.01]
Model	Peak	[– |2.5| –]	[– |5.0|1.5]	[– |3.0|1.0]	[.02|5.5|2.0]
No	Normal	[– | – | –]	[– |.04| –]	[– |.01| –]	[– |.05|.01]
Unc.	Peak	[.2|.6| –]	[– |7.0|1.0]	[– |2.0|.6]	[– |9.6|1.3]
Small	Normal	[– | – | –]	[.01|.03| –]	[– |.01| –]	[.01|.07|.02]
Unc.	Peak	[– |.2|.2]	[.2|5.4|2.8]	[.2|.4|.4]	[.4|7.8|4.4]
Med	Normal	[– | – | –]	[.01|.03|.01]	[.01|.02| –]	[.01|1.2|.02]
Unc.	Peak	[.6|4.4|.6]	[.2|4.4|3.8]	[.6|2.4|1.0]	[.2|6.2|5.4]
High	Normal	[– | – | –]	[.02|.01|.06]	[.01|.06|.02]	[.04|1.5|.08]
Unc.	Peak	[1.4|3.4|5.4]	[.2|6.8|9.8]	[1.2|5.6|4.2]	[.6|11.2|11.2]

Table 6 Execution times of the Crosswalk system for different number of crossings.

No. of crossings	No. of Agents	Time (secs.)
200	400	12.61
500	1000	32.00
1000	2000	62.19
1500	3000	126.00
2000	4000	264.68

Performance. Although in this paper our main focus is the expressiveness and analysis capabilities of our proposal, we also evaluated the performance of our simulations. The goal was to check whether the use of high-level languages, such as OCL, and the employ of USE, which is a tool neither intended or optimized for running simulations, could hinder the applicability of our approach. All the tests mentioned here were carried out on a personal computer with an Intel® core™ i7-7500U @2.70GHz, 16Gb RAM and running Windows 10 Pro and USE 6.0.

In the Crosswalk example, the average execution time for running 1000 crossings (i.e., involving 2000 agents) was 62.19 seconds, with a standard deviation of 4.5 s. These figures are not exceptional, but they were acceptable for running our tests in a reasonable time. Table 6 shows the execution times for different number of crossings and agents involved.

Table 7 shows the execution times of the Tanks example. We simulated the system with several number of tanks per team (1 to 4). Every time one of the tanks was shot down or ran out of power, it was removed from the simulation and a new one was created, until the total number of tanks defined for the simulation was reached (1000 and 1500, respectively).

4.6. Further issues and limitations

This section describes here some of the limitations of our proposal as well as other issues that may require further investigation.

Model Calibration. Validation is a process of comparing the model and its behavior to that of the real system. Further, calibration is the iterative process of comparing the model with real system, revising the model if necessary by, e.g., adjusting the model parameters, until the behavior of the model faithfully emulates that of the real system (Wigan 1972). In addition to the modeling effort required to develop the models of the system, their calibration requires a significant effort that cannot be neglected (Glenn et al. 2004; Kiesel et al. 2011).

Table 7 Execution times of the Tanks Warfare system for different number of simulations.

	No. of tanks per team			
Total No. of tanks	**1**	**2**	**3**	**4**
1000	29.29	34.45	42.62	44.76
1500	39.05	46.08	57.52	63.05

In our case, for calibrating and validating the Crosswalk system agents we used the data of a research project on ITS applications, on which the example was inspired,[2] as well as the existing data available from the literature on the behavior of cars and pedestrians, as referenced in Sect. 3. The time and effort we spent in calibrating the behavior of the individual agents was similar to the effort required to develop the models, much more than we initially expected. Unfortunately, this step cannot be ignored (Wigan 1972), although some solutions could be used to mitigate the effort required. For example, the development of a library of ready-made agents of several classes that can be reused in different models could be of great help.

Related to this, unconstrained propagation of measurement uncertainty can also represent problems because of an unbounded and excessive accumulation of uncertainty. One countermeasure that we used was to restrict the life-time of the agents as much as possible, focusing just on their behavior during each simulation cycle.

Complexity. Simulating the interactions among agents with uncertain behavior is inherently complex. Even when we have proposed a step-by-step methodology to incrementally add several types of uncertainty into the behavior of the agents, this approach requires to explicitly consider the interactions among the agents too. Thus, the complexity of the specifications will grow exponentially with the number of agent types. For example, introducing a traffic light in the crosswalk example, or two different lanes, one for pedestrians and one for cyclists, would require to consider a growing combination of situations. This problem, which also applies to systems modeled with BBNs, would require further research.

Faithfulness. In general, there is an inevitable compromise between the level of detail and the complexity of the specifications. The key is to define a set of abstractions that are faithful enough to capture the relevant properties of interest, and simple enough to be usable by the system modelers and analysts (Lee 2017). In our context, the value of a model lies in how well its behavior matches that of the physical system (Lee & Sirjani 2018). Such a faithfulness is easier to achieve if properties of the modeling language itself reflect properties of the problem

[2] https://c-mobile-project.eu/

domain being modeled (Sirjani 2018), and this is why the use of the extension of OCL datatypes with uncertainty has proved to be very helpful. Likewise, the use of casual networks and influence diagrams have provided the appropriate specification mechanisms for reasoning about the agents' motivations. They have enabled us to explicitly capture the relevant aspects of the system, and understand how they influence its behavior. With this paper we have tried to show the power of modeling uncertain behavior using high-level notations, and the mechanisms available to achieve this.

Other limitations. Apart from the limitations discussed above, one of the problems of a proposal like this is obtaining real data to validate and calibrate the models. Even when our crossing system was inspired in a real example from a research project in the Netherlands, every crossing is unique and generalizing from this data would be a challenge.

Our current proposal may also not be suitable for all types of systems. For example, it cannot deal with continuous models of physical systems, but only with discrete ones (or the discrete versions of the continuous models). Furthermore, as mentioned above, the complexity of the model increases with the complexity of the problem. This implies larger models that are more difficult to develop, debug, validate and calibrate. Even when the high-level modeling notations used attempt to reduce the accidental complexity of modeling efforts, the inherent complexity of considering systems with uncertainty presents a challenge for software modelers.

Finally, another general problem is that the modeler has to anticipate the types of uncertainty the system is subject to. Therefore, unknown-knowns and unknown-unknowns are not contemplated. These types of uncertain behaviours are now all encapsulated as *random* decisions of the agents, and are added during the last step of the proposal. Further analyses for unexpected behaviours may be needed in order to reduce this uncertainty.

5. Conclusions and Future Work

In this paper we have presented a proposal for the specification of the behavior of agents under different types of uncertainty, using high-level models. We have also described how these models can be used to simulate the modeled systems and to analyze some of their properties of interest. Two main case studies have been used to demonstrate the proposal, and to highlight its main advantages and limitations.

Our work can be continued in several directions. First, further case studies should give us more feedback on the features, expressiveness and possible improvements of our approach. Second, empirical validation with realistic systems in existing environments would help us evaluate its applicability, usability and effectiveness. Finally, adding new types of analyses to our approach, such as static ones, or combining our proposal with other specification techniques, such as scenario-based testing to generate artificial scenarios as a way to support the specification of parameters and variability in the system (Babikian 2020; Gadelha et al. 2020), are also part of our future research goals.

Acknowledgments

We would like to thank the reviewers for their very valuable comments and suggestions, which helped us significantly to improve the paper. This work is partially supported by the Spanish Government under project COSCA (PGC2018-094905-B-I00) and by the European Union's Horizon 2020 Research and Innovation Programme under Grant Agreement number 723311: Accelerating C-ITS Mobility Innovation and depLoyment in Europe (C-MobILE).

References

Albrecht, S., & Stone, P. (2018). Autonomous agents modelling other agents: A comprehensive survey and open problems. *Artif. Intell.*, *258*, 66–95. doi: 10.1016/j.artint.2018.01.002

Babikian, A. A. (2020). Automated generation of test scenario models for the system-level safety assurance of autonomous vehicles. In *MODELS'20 companion proc.* (pp. 24:1–24:7). ACM. doi: 10.1145/3417990.3419484

Bertoa, M. F., Burgueño, L., Moreno, N., & Vallecillo, A. (2020). Incorporating measurement uncertainty into OCL/UML primitive datatypes. *Softw. Syst. Model.*, *19*(5), 1163–1189. doi: 10.1007/s10270-019-00741-0

Bucchiarone, A., Cabot, J., Paige, R. F., & Pierantonio, A. (2020, January). 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

Czarnecki, K., & Salay, R. (2018). Towards a framework to manage perceptual uncertainty for safe automated driving. In *Proc. of SAFECOMP'18 workshops* (Vol. 11094, pp. 439–445). Springer. doi: 10.1007/978-3-319-99229-7_37

Dajsuren, Y., & van den Brand, M. (Eds.). (2019). *Automotive Systems and Software Engineering – State of the Art and Future Trends*. Springer. doi: 10.1007/978-3-030-12157-0

Darwiche, A. (2009). *Modeling and reasoning with bayesian networks*. Cambridge University Press.

Famelis, M., Salay, R., & Chechik, M. (2012). Partial models: Towards modeling and reasoning with uncertainty. In *Proc. of ICSE'12* (pp. 573–583). IEEE Press.

Gadelha, R., Vieira, L., Barbosa, D. M., Vidal, F., & Maia, P. H. M. (2020). Scen@rist: an approach for verifying self-

adaptive systems using runtime scenarios. *Softw. Qual. J.*, *28*(3), 1303–1345. doi: 10.1007/s11219-019-09486-x

Garlan, D. (2010). Software Engineering in an Uncertain World. In *Proc. of FoSER@FSE/SDP 2010* (pp. 125–128). ACM. doi: 10.1145/1882362.1882389

Geller, B., & Bradley, T. (2012). Quantifying uncertainty in vehicle simulation studies. *SAE Int. J. Passeng. Cars - Mech. Syst.*, *5*, 381–392. doi: 10.4271/2012-01-0506

Giese, H., Bencomo, N., Pasquale, L., Ramirez, A. J., Inverardi, P., Wätzoldt, S., & Clarke, S. (2014). Living with Uncertainty in the Age of Runtime Models. In *Models@run.time* (Vol. 8378, pp. 47–100). Springer. doi: 10.1007/978-3-319-08915-7_3

Glenn, F., Neville, K., Stokes, J., & Ryder, J. (2004). Validation and calibration of human performance models to support simulation-based acquisition. In *Proc. of the winter simulation conference* (pp. 1533–1540). IEEE Computer Society.

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

Gogolla, M., Hilken, F., & Doan, K.-H. (2018, December). Achieving model quality through model validation, verification and exploration. *Computer Languages, Systems & Structures*, *54*, 474–511. doi: 10.1016/j.cl.2017.10.001

Harel, D. (1987). Statecharts: A visual formalism for complex systems. *Sci. Comput. Program.*, *8*(3), 231–274. doi: 10.1016/0167-6423(87)90035-9

Helbing, D., & Molnár, P. (1995). Social force model for pedestrian dynamics. *Phys. Rev. E*, *51*(5), 4282–4286.

Helle, P., Schamai, W., & Strobel, C. (2016). Testing of autonomous systems — challenges and current state-of-the-art. In *Incose international symposium* (p. 571—584).

Howard, R. A., & Matheson, J. E. (2005). Influence diagrams. *Decision Analysis*, *2*(3), 127–143. doi: 10.1287/deca.1050.0020

Huang, L., Wu, J., You, F., Lv, Z., & Song, H. (2017). Cyclist social force model at unsignalized intersections with heterogeneous traffic. *IEEE Trans. Industrial Informatics*, *13*(2), 782–792. doi: 10.1109/TII.2016.2597744

Jain, A., & Boehm, B. W. (2006). SimVBSE: Developing a game for value-based software engineering. In *Proc. of CSEE&T'06* (pp. 103–114). IEEE Computer Society. doi: 10.1109/CSEET.2006.31

Jay, M., Régnier, A., Dasnon, A., Brunet, K., & Pelé, M. (2020). The light is red: Uncertainty behaviours displayed by pedestrians during illegal road crossing. *Accident Analysis & Prevention*, *135*, 105369.

JCGM 100:2008. (2008). *Evaluation of measurement data—Guide to the expression of uncertainty in measurement (GUM).* Retrieved from http://bit.ly/GUM100

Jing, X., Pinel, P., Pi, L., Aranega, V., & Baron, C. (2008). Modeling uncertain and imprecise information in process modeling with UML. In *Proc. of COMAD'08* (pp. 237–240). Retrieved from http://www.cse.iitb.ac.in/%7Ecomad/2008/PDFs/11-jxioa.pdf

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

Karkhanis, P., van den Brand, M. G. J., & Rajkarnikar, S. (2018). Defining the C-ITS reference architecture. In *ICSA'18 companion* (pp. 148–151). IEEE Computer Society. doi: 10.1109/ICSA-C.2018.00044

Kiesel, R., Streubühr, M., Haubelt, C., Löhlein, O., & Teich, J. (2011). Calibration and validation of software performance models for pedestrian detection systems. In *Proc. of SAMOS'11* (pp. 182–189). IEEE. doi: 10.1109/SAMOS.2011.6045460

Kirchhof, J. C., Michael, J., Rumpe, B., Varga, S., & Wortmann, A. (2020). Model-driven digital twin construction: synthesizing the integration of cyber-physical systems with their information systems. In *Proc. of MoDELS'20* (pp. 90–101). ACM. doi: 10.1145/3365438.3410941

Kiureghian, A. D., & Ditlevsen, O. (2009). Aleatory or epistemic? does it matter? *Structural Safety*, *31*(2), 105–112.

Koopman, P., & Wagner, M. (2016). Challenges in autonomous vehicle testing and validation. *SAE International Journal of Transportation Safety*, *4*, 15–24. doi: 10.4271/2016-01-0128

Kraaier, J., & Killat, U. (2008). Mobile networks random direction or random waypoint? A comparison of mobility models for urban environments. *European Transactions on Telecommunications*, *19*(8), 879–894. doi: 10.1002/ett.1219

Kwon, W. T., Park, N. C., Jung, S. H. J., & Kim, T. G. (1996). Fuzzy-DEVS Formalism: Concepts, Realization and Applications. In *Proc. of ais'96* (p. 227-–234).

Lee, E. A. (2017). *Plato and the Nerd. The Creative Partnership of Humans and Technology*. Cambridge, MA: MIT Press.

Lee, E. A., & Sirjani, M. (2018). What good are models? In *Proc. of FACS 2018* (Vol. 11222, pp. 3–31). Springer. doi: 10.1007/978-3-030-02146-7_1

Leibowitz, H. W., Owens, D. A., & Tyrrell, R. A. (1998). The assured clear distance ahead rule: Implications for nighttime traffic safety and the law. *Accident Analysis & Prevention*, *30*(1), 93–99. doi: 10.1016/s0001-4575(97)00067-5

Liu, L., Peng, J., Zhang, X., Zhang, R., Chen, B., Gao, K., & Yang, Y. (2018). Reactive behavioral strategy for unmanned ground vehicle under liner temporal logic specifications. In *Proc. of bigcomp'18* (pp. 133–140). IEEE Computer Society. doi: 10.1109/BigComp.2018.00028

Ludewig, J., Bassler, T., Deininger, M., Schneider, K., & Schwille, J. (1992). SESAM - simulating software projects. In *Proc. of SEKE'92* (pp. 608–615). IEEE Computer Society. doi: 10.1109/SEKE.1992.227898

Makishitaa, H., & Matsunaga, K. (2008). Differences of drivers' reaction times according to age and mental workload. *Accident Analysis & Prevention*, *40*(2), 567–575. doi: 10.1016/j.aap.2007.08.012

Marron, A., Limonad, L., Pollack, S., & Harel, D. (2020). Expecting the unexpected: Developing autonomous-system design principles for reacting to unpredicted events and conditions. *CoRR*, *abs/2001.06047*. Retrieved from https://arxiv.org/abs/2001.06047

Martín-Rodilla, P., & Gonzalez-Perez, C. (2019). Conceptualization and non-relational implementation of ontological and epistemic vagueness of information in digital humanities.

Informatics, *6*(2), 20. doi: 10.3390/informatics6020020

Mosterman, P. J., & Zander, J. (2016). Industry 4.0 as a cyber-physical system study. *Software and Systems Modeling*, *15*(1), 17–29. doi: 10.1007/s10270-015-0493-x

Muñoz, P., Burgueño, L., Ortiz, V., & Vallecillo, A. (2020). Extending OCL with Subjective Logic. *Journal of Object Technology*, *19*(3), 3:1-15. (Special Issue dedicated to Martin Gogolla on his 65th Birthday) doi: 10.5381/jot.2020.19.3.a1

Nassal, A., & Tichy, M. (2016). Modeling human behavior for software engineering simulation games. In *In proc. of GAS@ICSE'16* (pp. 8–14). ACM. doi: 10.1145/2896958.2896961

Navarro, E. O., & van der Hoek, A. (2004). SIMSE: an interactive simulation game for software engineering education. In *Proc. of CATE'04* (pp. 12–17).

Neumann, J. v., & Morgenstern, O. (1953). *Theory of games and economic behavior*. Princeton University Press.

Novák, P. (2009). Probabilistic behavioural state machines. In *Proc. of ProMAS'09* (Vol. 5919, pp. 67–81). Springer. doi: 10.1007/978-3-642-14843-9_5

Oberkampf, W., DeLand, S., Rutherford, B., Diegert, K., & Alvin, K. (2002). Error and uncertainty in modeling and simulation. *Reliability Engineering & System Safety*, *75*(3), 333–357. doi: 10.1016/S0951-8320(01)00120-X

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)

Pearl, J. (1994). A probabilistic calculus of actions. In *Proc. of UAI'94* (pp. 454–462).

Pearl, J. (2000). *Causality: Models, reasoning and inference*. Cambridge University Press.

Pearl, J., & Mackenzie, D. (2018). *The book of why: The new science of cause and effect*. Basic Books.

Pedersen, T., Johansen, C., & Jøsang, A. (2018). Behavioural computer science: an agenda for combining modelling of human and system behaviours. *Hum. Cent. Comput. Inf. Sci.*, *8*, 7. doi: 10.1186/s13673-018-0130-0

Phan, B., Khan, S., Salay, R., & Czarnecki, K. (2019). Bayesian uncertainty quantification with synthetic data. In *Proc. of SAFECOMP'19 workshops* (Vol. 11699, pp. 378–390). Springer. doi: 10.1007/978-3-030-26250-1_31

Refsdal, A. (2008). *Specifying computer systems with probabilistic sequence diagrams* (Doctoral dissertation, University of Oslo, Norway). Retrieved from https://www.duo.uio.no/handle/10852/9873

Refsdal, A., & Stølen, K. (2008). Extending UML sequence diagrams to model trust-dependent behavior with the aim to support risk analysis. *Sci. Comput. Program.*, *74*(1-2), 34–42. doi: 10.1016/j.scico.2008.09.003

Ridel, D. A., Rehder, E., Lauer, M., Stiller, C., & Wolf, D. F. (2018). A literature review on the prediction of pedestrian behavior in urban scenarios. In *Proc. of ITSC'18* (pp. 3105–3112). IEEE. doi: 10.1109/ITSC.2018.8569415

Salay, R., Chechik, M., Horkoff, J., & Sandro, A. (2013). Managing requirements uncertainty with partial models. *Requirements Eng.*, *18*(2), 107–128.

Salay, R., Czarnecki, K., Elli, M. S., Alvarez, I. J., Sedwards, S., & Weast, J. (2020). PURSS: towards perceptual uncertainty aware responsibility sensitive safety with ML. In *Proc. of SafeAI@AAAI'20* (Vol. 2560, pp. 91–95). CEUR-WS.org. Retrieved from http://ceur-ws.org/Vol-2560/paper34.pdf

Sanders, R. L. (2015). Perceived traffic risk for cyclists: the impact of near miss and collision experiences. *Accident Analysis and Prevention*, *75*, 26–34. doi: 10.1016/j.aap.2014.11.004

Sharma, S., Ogunlana, K., Scribner, D., & Grynovicki, J. (2018). Modeling human behavior during emergency evacuation using intelligent agents: A multi-agent simulation approach. *Information Systems Frontiers*, *20*(4), 741–757. doi: 10.1007/s10796-017-9791-x

Shaw, K., & Dermoudy, J. R. (2005). Engendering an empathy for software engineering. In *Proc. of ACE'05* (pp. 135–144). doi: 10.5555/1082424.1082441

Shi, D., Selekwa, M. F., Jr., E. G. C., & Moore, C. A. (2005). Fuzzy behavior navigation for an unmanned helicopter in unknown environments. In *Proc. of CSMC'05* (pp. 3897–3902). IEEE. doi: 10.1109/ICSMC.2005.1571754

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

Thompson, E. L., & Smith, L. A. (2019). Escape from model-land. *Economics: The Open-Access, Open-Assessment E-Journal*, *15*(2019-40), 1–15. doi: 10.5018/economics-ejournal.ja.2019-40

Troya, J., Moreno, N., Bertoa, M. F., & Vallecillo, A. (2021). Uncertainty representation in software models: A survey. *Software and Systems Modeling*. doi: 10.1007/s10270-020-00842-1

Troya, J., Vallecillo, A., Durán, F., & Zschaler, S. (2013). Model-driven performance analysis of rule-based domain specific visual models. *Information & Software Technology*, *55*(1), 88–110. doi: 10.1016/j.infsof.2012.07.009

Vallecillo, A., & Gogolla, M. (2017). Adding random operations to OCL. In *Proc. of MoDeVVa'17* (Vol. 2019, pp. 324–328). CEUR-WS.org. Retrieved from http://ceur-ws.org/Vol-2019/modevva_5.pdf

Wang, Y., & Bai, L. (2019). Fuzzy Spatiotemporal Data Modeling Based on UML. *IEEE Access*, *7*, 45405–45416. doi: 10.1109/ACCESS.2019.2908224

Wigan, M. (1972). The fitting, calibration, and validation of simulation models. *SIMULATION: Transactions of The Society for Modeling and Simulation International*, *18*, 188–192. doi: 10.1177/003754977201800506

Wortmann, A., Barais, O., Combemale, B., & Wimmer, M.

(2020). Modeling languages in industry 4.0: an extended systematic mapping study. *Softw. Syst. Model.*, *19*(1), 67–94. doi: 10.1007/s10270-019-00757-6

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

Zhang, M., Yue, T., Ali, S., Selic, B., Okariz, O., Norgren, R., & Intxausti, K. (2018). Specifying uncertainty in use case models. *J. Syst. Softw.*, *144*, 573–603. doi: 10.1016/j.jss.2018.06.075

Zhou, B., Lu, J., Wang, Z., Zhang, Y., & Miao, Z. (2009). Formalizing Fuzzy UML Class Diagrams with Fuzzy Description Logics. In *Proc. of IITA'09* (Vol. 1, pp. 171–174). doi: 10.1109/IITA.2009.97

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.

Priyanka Karkhanis is a PhD candidate at the Eindhoven University of Technology (TU/e), The Netherlands. Her research focus on the cooperative intelligent transport systems architecture models and simulation. You can contact the author at p.d.karkhanis@tue.nl.

Mark van den Brand is a full Professor Software Engineering and Technology of the section Model Driven Software Engineering, Eindhoven University of Technology (TU/e), The Netherlands. His main research areas are model-based software engineering, and automotive software engineering. You can contact the author at m.g.j.v.d.brand@tue.nl or visit https://www.tue.nl/en/research/researchers/mark-van-den-brand.

Antonio Vallecillo is full Professor at the University of Málaga, where he leads the Atenea Research Group on Software and Systems Modeling. 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/.