Noname manuscript No.
(will be inserted by the editor)

EURECA: Epistemic Uncertainty Classification Scheme for Runtime Information Exchange in Collaborative System Groups

Constantin Hildebrandt · Torsten Bandyszak · Ana Petrovska · Nishanth Laxman · Emilia Cioroaica · Sebastian Törsleff

Received: date / Accepted: date

Abstract Collaborative Embedded Systems (CES) typically operate in highly dynamic contexts that cannot be completely predicted during design time. These systems are subject to a wide range of uncertainties occurring at runtime, which can be distinguished in aleatory or epistemic. While aleatory uncertainty refers to stochasticity that is present in natural or physical processes and systems, epistemic uncertainty refers to the knowledge that is available to the system, for example, in the form of an ontology, being insufficient for the functionalities that require certain knowledge. Even though both of these two kinds of uncertainties are relevant for CES, epistemic uncertainties are especially important, since forming Collaborative System Groups (CSGs) requires a structured exchange of information. In the autonomous driving domain for instance, the information exchange between different CES of different vehicles may be related to own or environmental behavior, goals or functionalities. By today, the systematic identification of epistemic uncertainties sourced in the information exchange is insufficiently explored, as only some specialized classifications for uncertainties in the area of self-adaptive systems exist. This paper contributes an epistemic uncertainty classification scheme for runtime information exchange (EURECA) in collaborative system groups. By using this classification scheme, it is possible to identify the relevant epistemic sources of uncertainties for a CES during Requirements Engineering.

Keywords Uncertainty, Requirements Engineering, Runtime Information Exchange, Collaboration

C. Hildebrandt
Helmut-Schmidt-University, Institute of Automation Technology, Holstenhofweg 85, 22043 Hamburg, Germany
Tel.: +49-40-6541-3789
Fax: +123-40-6541-2004
E-mail: c.hildebrandt@hsu-hh.de

T. Bandyszak
paluno – The Ruhr Institute for Software Technology, University of Duisburg-Essen, Gerlingstr. 16, 45127 Essen, Germany

A. Petrovska
Technische Universität München, Fakultät für Informatik, Boltzmannstr. 3, 85748 Garching bei München, Germany

N. Laxman
Software Engineering: Dependability, Technische Universität Kaiserslautern, Gottlieb-Daimler-Str., 67663 Kaiserslautern, Germany

E. Cioroaica
Fraunhofer IESE, Fraunhofer Platz 1, 67663 Kaiserslautern, Germany

S. Törsleff
Helmut-Schmidt-University, Institute of Automation Technology, Holstenhofweg 85, 22043 Hamburg, Germany

1 Introduction

In the past, embedded systems had predominantly static relations to their context. In the future, however, it can be expected that they will increasingly operate within collaborative networks [1]. Embedded systems that dynamically collaborate with each other are referred to as collaborative embedded systems (CESs). The groups that are being formed by CESs to achieve specific goals are called collaborative system groups (CSGs), e.g. vehicles forming a platoon to save fuel when travelling to a common destination [2]. The prerequisite for such collaboration is that CESs communicate with each other. Much research has been dedicated to uncertainties that occur at the runtime of cyber-physical systems or self-adaptive systems, and how they can be classified (see Section 2). A source of uncertainty that has not been systematically addressed yet is the communication act itself, that is a receiver being uncertain with respect to the information that a sender wants to convey. For instance, when a vehicle in a platoon proposes a new target speed of 60 by sending a message including only the number '60', it might be unclear whether the number is referring to miles per hour or kilometers per hour. Another example would be the presence of unknown concepts in a message, for example, the sender referring to the concept “speed” in a message, while the receiver is only aware of the concept “velocity”.

This contribution introduces an approach for addressing communication-induced epistemic uncertainties. On the one hand, the approach provides a classification scheme for such uncertainties. On the other hand, it provides a method for applying the scheme as a checklist to support the systematic analysis of potential uncertainties in the communication of CESs. This is based on scenarios and can be integrated into the requirements engineering of such systems in order to explicitly take uncertainty into account. The analysis of consequences, e.g. hazards, that may arise from communication-induced uncertainties is not in the scope of our approach. However, it is positioned to provide structured inputs to such scenario analyses. We employ the autonomous driving domain as a running example as well as for the exemplary application of our approach. The approach, however, is designed to be applicable for all application domains of CSGs. As such, preliminary validation has been performed for two additional domains - factory automation and energy distribution.

This paper is structured as follows: Section 2 presents the body of research regarding runtime uncertainties to put our research into context. In Section 3, we discuss uncertainties that may arise within CSGs. Subsequently, we introduce our approach in Section 4. In Section 5, we present an exemplary application of our approach within the autonomous driving domain. We conclude the paper in Section 6 with a conclusion and outlook.

2 State of the Art – Runtime Uncertainty

The term uncertainty has been broadly discussed across many disciplines and sciences. In the literature there are many different given definitions and considered perspectives when it comes to defining and classifying uncertainties. For example, Walker et al. [3] consider uncertainty from a *decision making* point of view, to support making decisions in the presence of uncertainty. Refsgaard et al. [4] discuss uncertainty in the area of natural environment modelling. Foreseeably, in the area of computer science and software engineering, uncertainty is most notably discussed in the context of self-adaptive systems [5–9] and cyber-physical systems [10]. In these systems uncertainties have a central role, which stems from the manifestation of the dynamic nature of the systems.

2.1 Uncertainty Fundamentals

Cailliau and Lamsweerde [11] differentiate two essential levels of uncertainties: *physical uncertainty* and *knowledge uncertainty. Physical uncertainty* represents all the uncertainties that may occur in the system, for example, the failure of a sensor, or in the system context. *Knowledge uncertainty* denotes uncertainty that relates to the estimation of physical uncertainty values by experts.

Furthermore, [3] and [7] also cover the distinction between physical uncertainty and knowledge uncertainty, here referred to as *aleatory* and *epistemic* uncertainty instead. Uncertainty is classified as aleatory, if there is no foreseeable possibility for reducing it. Aleatory uncertainty refers to uncertainty due to the inherent stochasticity that is present in natural, physical processes and systems [12]. On the contrary, uncertainty is classified as epistemic, if there exists a possibility the uncertainty to be reduced by gathering more data or by refining models [13].

2.2 Classifying Runtime Uncertainties

In addition to the general categorization of uncertainties, which have been summarized in the previous Section, there are few more detailed taxonomies for classifying uncertainties that particularly focus on uncertainties that can occur at runtime. These have been, most notably, proposed in the context of adaptive systems, where uncertainty plays a major role and motivates the need for self-adaptation capabilities and associated engineering challenges [14]. All these classifications put a special emphasis on the time when the uncertainties occur, which is particularly important considering the importance of the role that time has in the above-mentioned systems. Moreover, all of the classifications introduce and focus on one or more uncertainty dimensions.

The very first classification that has been proposed by Walker et al. [3] (2003), targeting uncertainties in model-based decision support, which is closely related to the decision making and actions performed by software systems at runtime. As we will see below, more recent and software-specific classifications actually build upon [3]. Walker et al. identify three dimensions of uncertainties: *nature*, *location*, and *level. Nature* refers to the distinction of aleatory and epistemic uncertainty, which has been already discussed in the previous subsection. The *location* identifies where the uncertainties manifest. The *level* of uncertainty classifies uncertainty according to lack of knowledge and lack of awareness of knowledge deficits. Namely, in our classification we map the *information exchange* to Walker's location dimension, and we refer to their level dimension as *lack of knowledge*. Additionally, we provide a schema based on representation of knowledge using ontologies.

Ramirez et al. [6] (2012) propose a taxonomy and describe potential sources of uncertainties on three different levels, according to the life cycle phase in which uncertainty can occur: the requirements level, design level, and runtime level. During the development of the system, the sources of uncertainties mainly refer to consequences of deficiencies in conducting the development activities, such as ambiguous requirements, or an inadequate design. For instance, such uncertainties may be rooted in insufficient requirements negotiation or the inherent volatility of requirements. In contrast, runtime uncertainties affect the decision making of the envisioned system during operation. The sources of uncertainties at the higher levels, subsume the sources of uncertainties on the lower levels. Therefore, if the source of uncertainty is not resolved at the lower level (for example, on the requirements level), then it propagates to the higher ones (design level, and potentially runtime level). According to [6], potential sources of runtime uncertainties are mainly related to interactions between the system and its context, including sensor noise, inaccuracy of sensor measurements, or an unpredictable system environment. We focus on the system's ability to exchange information during operation, hence, runtime uncertainties in respect to Ramirez's taxonomy.

Perez-Palacin and Mirandola [7] (2014) focus on uncertainties that can be present in (formal) models used during development and at runtime to guide self-adaptation. Their taxonomy of model uncertainty is based on [3], and they further extend the identified sources of uncertainties in [5]. Primarily, they adopt two dimensions out of the three-dimensional taxonomy proposed by Walker et al. in [3]: *nature* (aleatory and epistemic uncertainty) and *location*. The location of an uncertainty identifies the place where uncertainty is visible in a model, for example, uncertainty can be related to the information a modeling language should be able to express and its limitations, to the way the reality is represented using prescribed modeling elements, or to the properties of model elements to be used for further analyses [7]. Among all of the proposed uncertainty classifications and taxonomies, Perez-Palacin's taxonomy is the closest to our contribution. The different uncertainty locations mentioned in their work have a lot in common with the distinction of different levels in our information exchange representation.

The most recent and sophisticated classifications are given by Mahdavi-Hezavehi et al. [8] (2017), which has been elicited from a systematic literature review, and by Camara et al. [9], which can be seen as an extension of the first taxonomy by Mahdavi-Hezavehi. Both give one of the most prominent classifications, identify and categorize plenty of previous works on uncertainty.

Mahdavi-Hezavehi et al. [8] first identify an *initial uncertainty dimensions classification schema* and *initial source classification schema*, based on the previous literature, in particular the work by Perez-Palacin et al. [7], Refsgaard et al. [4], David Garlan [15], Esfahani and Malek [5], and Ramirez et al. [6]. The *initial uncertainty dimensions classification schema* comprises four uncertainty dimensions: location (where the uncertainty manifests within the complexity of the model), nature (whether the uncertainty is due to the imperfection of the knowledge, or due to the inherent variability of the event), level/spectrum (where the uncertainty manifests along the spectrum between knowledge and ignorance), and sources (refers to the variety of uncertainties sources). The *initial source classification schema* comprises three uncertainty sources: model (uncertainties originating from system models), goals (uncertainties emerging from the systems goals and their obscurities), and environment (uncertainties originating from different environmental circumstances). They extended and completed both the *dimension schema* and *source classifications schema* based on data they extracted from their primary study.

Therefore, both taxonomies distinguish *five dimensions of uncertainties*: location, source, nature, level, and emerging time. Hence, the classifications extend the initial three-dimensional concept from [3], and incorporate the source and emerging time (design-time or runtime) as fourth and fifth dimension. Additionally, Mahdavi-Hezavehi et al. extended the *initial source classification schema* to following generic classes of uncertainty sources: model uncertainty, adaptation functions uncertainty, goals uncertainty, environment uncertainty, resources uncertainty and managed system uncertainty.

In the last three [7–9] classifications, the uncertainty sources are always use-case specific and there is no alternative in using the classifications without using their model specifications (context model, resource model or goal model). However, in CES practice, there will be models defined by an external entity—industry, for example. Sometimes there might be even models that are not envisioned yet. This leaves open the question whether it is meaningful to classify uncertainty sources by using a specific kind of model. In our work, we are proposing a classification scheme that can be used to identify uncertainty sources in any kind of model.

Our approach provides a classification schema for types of epistemic uncertainties at runtime, with special emphasis on uncertainties originating from information exchange in collaborating systems, which has not been addressed by another uncertainty classification to the best of our knowledge.

3 Epistemic Uncertainties in Collaborative System Groups

3.1 Information Exchange in CSGs

In the following, we introduce some general characteristics of collaborative embedded systems (CES), which pose specific challenges to the consideration of different kinds of runtime uncertainties. Such collaborative embedded systems (CESs) operate in highly dynamic environments. Additionally, CES collaborate when they combine their individual capabilities to achieve a common goal. Achieving a common goal can be beyond the capabilities of the individual systems, and thus specifically require a collaboration. When systems collaborate to achieve a specific goal, they form a collaborative system group (CSG). A CSG can evolve dynamically, for example, it might include new CES, or even dissolve.

To illustrate CES and CSG throughout the paper, we use examples from autonomous driving, which is an interdisciplinary research area that has recently gained much attention in the scientific community. In particular, we focus on vehicle platooning as an exemplary use case relevant for autonomous driving. A platoon is a group of vehicles controlled by cooperative adaptive cruise control (CACC) systems that utilize networking and communication to enable collaboration among the partaking vehicles, and can thus be considered a CSG [2]. A common goal that can be achieved through minimizing following distances in a platoon, is reducing the fuel consumption. Applied to heavy-duty trucks, platooning enables fuel savings of up to 20% [16].

A CES, which is under initial consideration for a potential engagement in a collaborating group, is called system under consideration (SUC), and is embedded into its operational context at runtime [17]. In the example of a platoon, the individual CACC systems controlling the partaking vehicles are the CES; however, to keep things simple, we simply denote the different vehicles as CESs under consideration. In general, the context of a system consists of all objects that are relevant to the system, but cannot be influenced by developers and is perceived as given [18]. Context objects (COs) can be divided into two different types [19]: Collaborative context objects and non-collaborative context objects. A collaborative context object is able to share information about itself in a reactive or proactive manner with the SUC; for example, the SUC calls a self-description service, or a collaborative context object pro-actively sends a self-description to the SUC or other CESs in a CSG, while accessing a network. In order for their collaboration to be initiated, CESs have to analyze each others' information, for instance, their specifications of functional and quality properties. In the platooning example, different collaborating vehicles exchange such information to communicate their destinations of travel [20]. Hence, from the point of view of one vehicle, the other vehicles of a platoon are collaborative context objects that interact with it. In contrast, non-collaborative context objects are unable to provide a machine-readable description of information that is required to collaborate with the SUC. Nevertheless, they can be perceived by the CESs, or they may even interact with CESs in some non-collaborative, passive manner. For instance, a radar sensor measuring the distance to a preceding vehicle can be considered non-collaborative.

3.2 Epistemic Uncertainties in Vehicle Platooning

As mentioned in Section 3.1, we focus on potential uncertainties that may occur in a platoon. There are many control paradigms available for realizing a platoon [21], which we do not discuss in detail. A platoon is typically led by a platoon leader that plays a special role, and a set of vehicles following the platoon leader [22]. Typically the vehicle with the highest safety characteristics is determined as the leader, which is negotiated among the vehicles [23]. One particular form for realizing a platoon is through joining a smart ecosystem as presented in [24]. Smart ecosystems consists of actors with different goals [25] that influence the dynamics within an ecosystem and bring along a set of uncertainties regarding system's goals. Moreover, ecosystems are not formed from scratch, already existing systems can be enhanced with software updates that make them collaborative. An example of a platoon formation as a smart ecosystem is through download of software updates that enables sharing of context information. At an entry point on a highway autonomous vehicles can get software updates that enable them to form a platoon through sharing of information regarding the perception of the environment.

In order to illustrate what kinds of epistemic uncertainties can occur in such a vehicle platoon, we consider one specific example scenario, i.e., the joining maneuver where a vehicle joins the platoon in the middle. The following explanations are based on the detailed description provided in [22]. A vehicle detects a platoon with roughly the same destination, and requests to join it. The platoon leader then coordinates the formation of a gap where the new vehicle can join the platoon. To this end, it communicates the gap information (i.e., the join position) to the new vehicle, which then adjusts its speed to approach the required position. The gap is opened by establishing a temporal second platoon (led by the vehicle in front of which the new vehicle is supposed to join) reducing its speed. Eventually, the joining vehicle is notified about the opened gap, and changes the lane to join in.

The coordination of the joining maneuver requires the exchange of a significant amount of information between vehicles. For example, the joining vehicle needs to check if its goals are compatible with the goals of the platoon leader. Exchanged goal specifications may include, for example, desired speed of travel (cf. [26]), or targeted fuel or energy savings. There are other scenarios in which information exchange is necessary to negotiate between different vehicles. Consider a scenario wherein an ambulance needs to arrive at a particular destination fast, and therefore its goal is to overtake it. To this end, the ambulance communicates its priority goal and thereby may request the platoon to change the lane to enable overtaking (cf. [27]).

In the scenarios sketched above, a wide range of potential uncertainties can occur. In addition to aleatory uncertainties originating from the physical surroundings, which are out of scope of this paper, major issues arise from epistemic uncertainties related to the information that is exchanged between vehicles. In general, different representations and different underlying ontologies of the versatile kinds of information exchanged within a platoon are potential sources for uncertainties that can occur at runtime. This might be due to the fact that vehicles built by different Original Equipment Manufacturers (OEMs) interact with each other, which is an important challenge in autonomous driving [28,29].

In a platoon, information about the context, including other vehicles or pedestrians, is exchanged, for example, through a smart ecosystem as introduced above. The perception of every vehicle differs. For example, consider the case where a human-driven vehicle enters the gap that has been formed for some other vehicle to join the platoon [22]. This situation may be detected by the joining vehicle through its on-board sensors, but not yet realized by the platoon leader (e.g. due to communication delays within the platoon, cf. [30]). As a consequence, the perception of the gap in the middle of a platoon by the platoon leader and the joining vehicle differs, which leads to ambiguous information about the context when respective information is exchanged [31].

Epistemic uncertainties could also be related to the goals of individual vehicles, which need to be negotiated in a platoon (for example, for joining). Being part of a CSG formed as a smart ecosystem, a CES interacts with other systems that have collaborative and competitive goals [25]. For instance, a platooning vehicle may leave its current platoon after detecting another one with more closely related goals. Uncertainties related to misunderstanding goals of different vehicles can have serious, hazardous effects. Due to different ontologies for expressing goal information used by different OEMs, a collaborating system could have a goal that is not properly understood by other vehicles in the platoon or it is not properly declared. This creates miss-understandings that can lead to safety-critical situations. For example, an individual goal of a vehicle to move forward in order to leave the platoon is communicated to the preceding vehicle that understands it as a goal in the platoon. As a consequence the preceding vehicle, in order to drive closely to the vehicle in front, increases its speed and causes a crash. Hence, solutions to assure safety in case of goal uncertainties need to be in place. Of course, in an ideal situation a single standardized ontology would be in place (for example, to exchange safety-related knowledge about accidents [32,33]), preventing such misunderstandings. However, there are still open research challenges regarding the interoperability of heterogeneous collaborating vehicular systems using different knowledge representations [34,35]. Furthermore, in order to achieve competitive advantage, an ecosystem, in our case the vehicle platoon, needs to effectively engage with external partners in order to maximize the benefits [36]. Hence, the classification scheme proposed in the next section aims at supporting the systematic identification of epistemic uncertainties that may occur during information exchange.

4 An Uncertainty Classification Scheme for Collaborative Embedded Systems

This chapter introduces a classification scheme that can be applied during Requirements Engineering for identifying and classifying epistemic uncertainties that result from information exchange between CESs. For this purpose, we introduce a simplified knowledge modelling procedure for capturing human knowledge in an ontology in Section 4.1. Two main classes of epistemic uncertainties in information exchange are introduced. Based on these definitions, Sections 4.2 and 4.3 include definitions of specific sub-classes for each main class, before Section 4.4 introduces the structure of the epistemic uncertainty classification scheme for runtime information exchange (EURECA) and where it is applied in Requirements Engineering.

4.1 Knowledge Modelling Procedure

As a special branch of knowledge engineering, ontological engineering is concerned with the formalization of

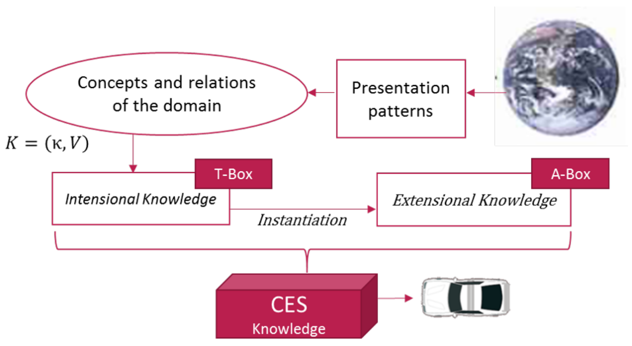


Fig. 1 Knowledge Modelling procedure

human knowledge in a machine readable way, i.e. an ontology [37]. To obtain such an ontology, one has to perform certain modelling steps, which are subsequently described by adapting a simplified procedure of [38]. The procedure is also summarized in Figure 1 while the red and gray color emphasize the application of the procedure by different development teams (constructing the SUC and, from the point of view of the SUC, the CO, respectively).

When formalizing human knowledge, the SUC's developers capture their real-world perception in presentation patterns first, e.g.: “vehicle must brake in case of an obstacle in the driving lane”. Afterwards, these informal presentation patterns are transformed into a conceptualization κ_{SUC} of the SUCs domain (e.g. autonomous driving) $\kappa_{SUC} = (D, \Re)$ with D being relevant concepts of the SUCs domain and $\Re$ being relevant relations between concepts of the SUCs domain.

Describing the real-world context of a vehicle that should be communicated during runtime for instance, requires a conceptualization κ_{SUC} of the real-world concepts $D = \{tree, vehicle, distance, ...\}$, including relations between these concepts $\Re = [\{tree \times hasDistance \times distance\}, \{...\}]$. After having defined the developer's real-world perception in the conceptualization κ_{SUC}, one can choose, depending on the application's needs, a language L that should represent the conceptualization in a machine readable format. Common languages in the field of Ontological Engineering are, for instance, Ontology Web Language Description Logics (OWL-DL) and First Order Logics [37]. Each language has its own vocabulary V (e.g. atomic concepts and atomic roles in Description Logics) denoting possible elements for representing parts of the conceptualization κ_{SUC}.

The transformation of the conceptualization κ_{SUC} into the vocabulary V is called ontological commitment $K_{SUC} = (\kappa_{SUC}, V)$, where each part of the conceptualization is mapped to an element of the vocabulary. For the extent of this paper, we introduce a simplified formal definition of a ontological language with vocabulary $V_S = (C, R, A)$, where C is a set of concepts, R is a set of relationship types among concepts, and A is a set of attributes. C, R and A are defined as tuples where:

- $c \in C; c = (t_c, d_c)$
- $r \in R; r = (t_r, d_r)$
- $a \in A; a = (t_a, d_a)$

While the term $t \in T$ only relates to the former defined D and $\Re$ (e.g. ”Tree”), the formal definition $d \in D$ uses the vocabulary of the language V_S in order to describe the term with that language (e.g. describing ”Tree” in description logic).

The result of the ontological commitment is the ontology O_{SUC} that can be used as a machine-readable artifact at runtime for representing intensional, terminological knowledge (T-Box or type level of knowledge, e.g. terms and logical definitions of ”tree”) and extensional, assertional knowledge (A-Box or instance level of knowledge, e.g. recognized instances of passed by ”trees” and their distance to the SUC) [39]. This knowledge modelling procedure is applied by the developers of the SUC, but also by developers of the COs, resulting in possibly different knowledge models.

When sending messages, we assume that the CES (i.e., SUC or a CO) transform only an excerpt of their extensional knowledge due to performance reasons into a message instead of sending their complete extensional knowledge. Therefore, the message only contains information which must be processed to extensional knowledge by the receiving device again.

For the instantiation of extensional knowledge in a message, the following definitions are necessary. A message m sent by the SUC that uses the ontology O_{SUC} for the specification of the message content, contains a set of information items I. Each information item $i \in I$ is a tuple $i = (v, rel)$, where v is defined as the value of the information item (e.g. value of an attribute a or target and source of a relation r, or an ID as an instance of c), and $rel \in (C \cup R \cup A)$ is defined as the mapping of each information item to the ontology, specifying the semantics of each value using the defined formal semantic definition of $O_{SUC} = \{C, R, A\}$. We can now motivate two super-classes of epistemic uncertainties in information exchange through the following example. A CO and an SUC communicate by exchanging messages, while using individual ontologies O_{CO} and O_{SUC} respectively, for the semantic specification of the information items within messages. An exemplary message could contain information for triggering an emergency brake maneuver when a CO identifies an obstacle and informs the SUC. In order to convey the emergency brake information, the CO sends an excerpt of its own extensional knowledge within a message m, which has been created based on its intensional knowledge. The SUC receives the message and processes the contained information in order to transform it into extensional knowledge according to its own intensional knowledge. There is **no** epistemic uncertainty in information exchange, if

1. All used ontological elements of the message m are known and consistent to both, the SUC and CO, defined as true for the following statement: $\forall i_j : (rel_j \in C_{O_{CO}} \wedge rel_j \in C_{O_{SUC}}) \vee (rel_j \in R_{O_{CO}} \wedge rel_j \in R_{O_{SUC}}) \vee (rel_j \in A_{O_{CO}} \wedge rel_j \in A_{O_{SUC}})$
2. The information items within the message m: $i_j = (v_j, rel_j)$ are not missing or violating (semantical inconsistency) a specification according to the definitions of O_{SUC}, and are not incomplete or inconsistent with respect to the actual situation of the SUC

If these two conditions hold, epistemic uncertainties sourced in the information exchange of the CO and the SUC are inexistent since both systems share the needed intensional knowledge to understand the information of message m. Moreover, the message m is well formed so that the contained information can be processed to extensional knowledge. The cases in which condition one is violated are presented in Section 4.2. Section 4.3 presents the cases violating the second condition. Section 5 introduces an exemplary application of each type and instance level uncertainty. The subsequent Sections 4.2 and 4.3 therefore only hold the necessary definitions due to the space restrictions.

4.2 Uncertainty Sources on the Type Level

The type level uncertainty cases, which are defined in this Section, are based on a T-Box mismatch at runtime. This may occur if CO and SUC are using different T-Boxes for specifying and interpreting the messages to be exchanged at runtime. Based on the simplified modelling procedure in Section 4.1, three reasons for a T-Box mismatch may occur:

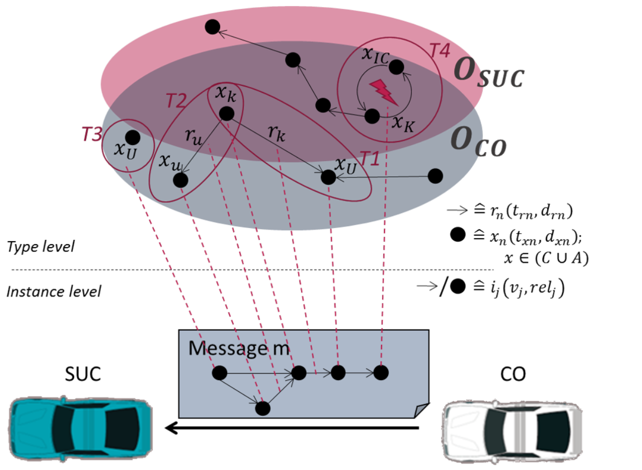
– The ontologies O_{SUC} and O_{CO} do differ, because the conceptualization κ_{SUC} and κ_{CO} already contained different elements (see subsequent T1, T2, T3).
– The ontologies O_{SUC} and O_{CO} do differ, because the ontological commitment was done differently by the individual developers, resulting in parts of κ_{CO} being semantically inconsistent to parts of O_{SUC} and vice versa (see subsequent T4).
– The ontologies O_{SUC} and O_{CO} do differ, because the developers used different languages for the ontological commitment. However, this case is not considered here.

In order to simplify the subsequent formal definitions, and to ease readability, we use subscript notation to denote relevant characteristics that are required to explain the four different type level uncertainties: From the SUC's point of view, a message m can contain unknown T-Box elements x_u or known T-Box elements x_k, which are described by a term known to the SUC ($t_{xO_{SUC}}$) and a logical definition of that term ($d_{xO_{SUC}}$). Each T-Box element can either be a concept, a relation or an attribut ($C \cup R \cup A$) within the SUCs or COs ontology:

$$x_u(t_{xO_{CO}}, d_{xO_{CO}}); x_u \in (C_{O_{CO}} \cup R_{O_{CO}} \cup A_{O_{CO}})$$

$$x_k(t_{xO_{SUC}}, d_{xO_{SUC}}); x_k \in (C_{O_{SUC}} \cup R_{O_{SUC}} \cup A_{O_{SUC}})$$

Figure 2 emphasizes the four different type level uncertainties by example, which are subsequently defined.



$T4$
x_{IC}
O_{SUC}
$T2$
x_k
x_K
$T3$
r_u
r_k
O_{CO}
x_U
x_u
x_U
$T1$
$\rightarrow \hat{=} r_n(t_{rn}, d_{rn})$
$● \hat{=} x_n(t_{xn}, d_{xn}); x \in (C \cup A)$
Type level
Instance level
$\rightarrow / ● \hat{=} i_j(v_j, rel_j)$
Message m
SUC
CO

Fig. 2 Type level uncertainty example

T1, known difference in scope: At least one ontological element of the message m is not known to the SUC, and the unknown element x_u has a known relation r_k to a known element x_k. From the logical description of $d_{KDS} = (d_{xO_{CO}}, d_{rO_{SUC}}, d_{xO_{SUC}})$ it can be inferred that x_u is a more abstract/detailed (e.g. "subclassing") element than x_k or from the same granularity (e.g. "same-as"). Here, the message m is not well understood, because of intensional knowledge unknown to the SUC. Due to the known relation to the unknown element, the SUC can at least use the semantics of $d_{rO_{SUC}}$ for further reasoning or mitigation.

T2, unknown difference in scope: At least one ontological element of the message m is not known to the SUC, and the unknown element x_u has an unknown relation r_u to a known element x_k. From the logical descriptions $d_{UDS} = (d_{xO_{CO}}, d_{rO_{CO}}, d_{xO_{SUC}})$ it can be inferred that x_u is somehow related to the SUC's intensional knowledge. The message m is not well understood, because of unknown but related intensional knowledge. Due to the fact, that there is no known relation to the unknown element, the SUC can only use the semantics of $d_{xO_{SUC}}$ for further reasoning or mitigation.

T3, distinct scope: At least one ontological element of the message m is not known to the SUC, and the unknown element x_u has no known relation r_k or unknown relation r_u to a known element x_k, so that none of the aforementioned cases holds. From the logical description $d_{xO_{CO}}$ it can be inferred that x_u does not have any relation to the SUC's intensional knowledge. The message m is not well understood, due to unknown intensional knowledge required for processing the information of message m. Due to the fact that there is no known or unknown relation to the unknown element, the SUC can only use the semantics of $d_{xO_{CO}}$ for further reasoning or mitigation.

T4, inconsistent ontological commitments: At least one ontological element of the message m: $x_{IOC}(t_{xO_{CO}}, d_{xO_{CO}})$; $x_u \in (C_{O_{CO}} \cup R_{O_{CO}} \cup A_{O_{CO}})$ violates known formal semantic definitions of $x_k(t_{xO_{SUC}}, d_{xO_{SUC}})$; $x_k \in (C_{O_{SUC}} \cup R_{O_{SUC}} \cup A_{O_{SUC}})$. The element x_{IOC} has an inconsistent formal semantic definition compared to x_k in the sense of: $(t_{xO_{SUC}} = t_{xO_{CO}}) \wedge (d_{xO_{SUC}} \neq d_{xO_{CO}})$. The information items associated with the ontological element x_{IOC} can therefore not be used, since this would result in inconsistent intensional knowledge of the SUC. The message m is not well understood, due to inconsistent intensional knowledge used for the specification of the message.

4.3 Uncertainty Sources on the Instance Level

The instance level uncertainty cases, which are defined in this Section, are based on the actual information specified in the message. Instance-level uncertainties may occur if the specified information cannot be processed to extensional knowledge (A-Box) of the SUC. Based on the simplified modelling procedure introduced in Section 4.1, two main reasons may occur:

- Non-situation related: Information items contained in the message m violate the SUC's intensional knowledge (I1), or the message m is missing formal semantic specifications of contained information items (I4).
- Situation related: Based on the actual situation of the SUC, in which it processes the received message, the contained information items in message m are either situationally inconsistent (I2) or situationally incomplete (I3).

For simplifying the definition of the four different instance level uncertainties, we facilitate the subsequent definitions: From the SUC's point of view, a message m contains a set of information items I_m, containing atomic i_j (see eq. (1)). The information items that are required by the SUC in an actual situation for executing own functionalities are denoted by i_{rj} (see eq. (3)). Each information item i_j or i_{rj} includes a defined reference to a type (i.e. the T-Box element), see eq. (2) and (4) respectively:

$$i_j \in I_m = (v_j, rel_j);\ \text{with } 0 \leq j \leq J \tag{1}$$

$$rel_j \mapsto x_k(t_{xO_{SUC}}, d_{xO_{SUC}}) \tag{2}$$

$$i_{rj} \in I_{req} = (v_{rj}, rel_{rj});\ \text{with } 0 \leq rj \leq RJ \tag{3}$$

$$rel_{rj} \mapsto x_k(t_{xO_{SUC}}, d_{xO_{SUC}}) \tag{4}$$

Figure 3 summarizes the subsequently defined four cases of instance level uncertainties.

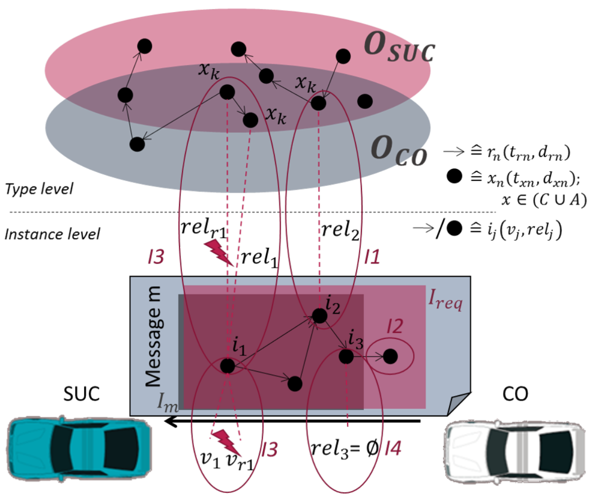


Fig. 3 Instance level uncertainty example

I1, semantically inconsistent information: The message m contains at least one information item $i_j = (v_j, rel_j)$, where the value v_j violates the known formal semantic definition of $rel_j \mapsto x_k$. This is independent of the actual situation that the SUC is embedded in. In this case, the message m is not well understood, due to information specified inconsistently with respect to to the intensional knowledge of the SUC.

I2, situationally incomplete information: The message m contains a set of information items I_m, and the SUC requires a set of information items I_{req} to be contained in m. The number of information items only partially meet the required information of the SUC, i.e. $I_m \subset I_{req}$, while the required set I_{req} depends on the actual situation that the SUC is embedded in. The message

m is well understood but misses necessary information expected in a certain situation.

I3, situationally inconsistent information: The message m contains a set of information items I_m, and the SUC requires a set of information items to be contained in the message I_{req}. The information items of the set I_m and their specification are situationally inconsistent to the set of required information items I_{req}, so that for at least one information item $i_j \in I_m$ the following statement holds true: $(v_j \neq v_{rj}) \wedge (rel_j \neq rel_{rj})$. Since the required set I_{req} depends on the actual situation that the SUC is embedded in, this instance level uncertainty is situation dependent. The message m is not well understood, due to situational inconsistency.

I4, missing type membership: The message m contains a set of information items I_m, and the SUC is not able to process at least one information item $i_j \in I_m$, because the information item has no type relation: $i_j = (v_j, rel_j = \emptyset)$. While this is independent of actual situation in which it occurs, the message m is not well understood, due to incomplete specification of the information item's type relation.

4.4 Epistemic Uncertainty Classification Scheme for Runtime Information Exchange

Table 1 shows the structure of the **e**pistemic **u**ncertainty classification scheme for **r**untime information **exc**h**a**nge (EURECA), which is subsequently described. EURECA is a two-dimensional schema. The first column contains all ontologies O_k that are relevant for information exchange with context objects during runtime of the SUC. The second column contains all concepts c_l, relations r_m and attributes a_n of each O_k, which are subject to epistemic uncertainties at runtime. The next columns contain the type and instance level uncertainties defined in Sections 4.2 and 4.3. Each cell crossing a type or instance level uncertainty with concepts c_l, relations r_m and attributes a_n of O_k can be used to document an epistemic uncertainty that may occur during runtime and that has to be mitigated during the engineering. To apply EURECA, the following prerequisites have to hold:

– The ontologies O_k (e.g. goal or context ontologies) that are used for runtime information exchange are available and known.
– The use case scenarios that specify behavioral requirements of the SUC are available and known in order to elicit relevant crossings of type and instance level uncertainties with concepts c_l, relations r_m and attributes a_n of O_k.

Table 1 Epistemic uncertainty classification scheme for runtime information exchange

O_k	c_l, r_m, a_n	Type level uncertainty				Instance level uncertainty			
		T1	T2	T3	T4	I1	I2	I3	I4
Ontology O_k	c_l	Identified type-level uncertainty sources per scenario				Identified instance-level uncertainty sources per scenario			
	r_m								
	a_n								

With respect to the above mentioned prerequisites, EURECA is intended to be applied during Requirements Engineering, as soon as use case scenarios or other kinds of behavioral requirements have been specified, which indicate the required information to be exchanged during runtime by SUC and COs. Please refer to [40] for a comprehensive description on behavioral requirements and their documentation with message sequence charts. Furthermore, please refer to [41] for a description on how to develop the ontologies mentioned in the first prerequisite.

5 Classified (Epistemic) Uncertainties in Exemplary Use Case Scenarios

This Section provides an exemplary application of the classification scheme (EURECA) which has been introduced in Section 4.4. For this, different scenarios from vehicle platooning use case are considered (Section 3.2).

In our example, we illustrate a vehicle platoon driving on a highway. As our focus is on epistemic uncertainties, the information exchange between the vehicles is considered. In particular, we focus on the joining maneuver (Section 3.2): A vehicle on the highway, not part of a platoon yet, would like to join the platoon under consideration and approaches it. As most of the important decisions are managed by a platoon leader, it is considered as SUC in all successive scenarios, and the joining vehicle is considered to be the CO for the SUC. The SUC is considered to be powered by internal combustion (IC) engine and the (potentially) joining vehicle being powered by electric motor. This gives rise to a wide range of potential uncertainties, as the information exchanged for joining a platoon contain vehicle specifications, which may significantly differ for each engine type.

The common outset to all subsequent scenarios is that the CO has to request the platoon leader to join the platoon. Hereby, SUC and CO exchange messages to establish a CSG. Despite the fact that there will be various information being exchanged, our focus for the subsequent scenarios is limited to the information ex-

Table 2 EURECA scheme for the application scenario of vehicle platooning. TLU = type level uncertainty, ILU = instance level uncertainty

O	c_l, r_m, a_n	TLU				ILU			
		T1	T2	T3	T4	I1	I2	I3	I4
SUC goal ontology	c_1	S1							
	c_4		S2						
	c_5			S3					
	r_2					S5	S5		
	r_4								
	a_1				S4				
	a_2							S6	
	a_6								
	a_7								S7
	a_8						S5		

change pertaining to goal specification. To negotiate the goals, and coordinate the joining procedure, the SUC demands the CO to share some relevant goal specifications, like its "desired fuel consumption". As our focus is on information exchange (i.e. the payload of a protocol), we avoid technical details of the specific communication protocol, which are related to, e.g. how messages are structured, or how required fields and type descriptions of a message can be identified by individual vehicles.

Table 2 gives an overview of the different concepts, relations, and attributes relevant to the platooning scenario used for illustration. The table also illustrates the use of EURECA for guiding uncertainty identification during Requirements Engineering: Each cell referencing a scenario (S1-S7) indicates a relevant uncertainty identified during requirements elicitation, which somehow manifests in a certain concept, relation, or attribute considered in the development of a software-intensive system that processes information exchanged in a platoon (such as a CACC, see Section 3.2). For each filled cell we provide a description of an exemplary scenario (S1-S7), in which the respective uncertainty may occur, in the following subsections. We expect the modelling of each scenario to be done, for instance, in a message sequence chart (MSC) indicating the information exchange between SUC and CO. Due to space restrictions, we do not show the MSCs depicting each scenario subsequently.

5.1 Scenarios Illustrating Type Level Uncertainties

T1, known difference in scope: In this scenario (S1), the platoon leader (SUC) receives a message m specified using a known concept c_1 with a known relation r_4 to an unknown concept c_u.

$$x_k = c_1 = (t_{c_1} = \text{“MinimizeEnergyConsumption”},$$
$$d_{c_1} = s(t_{c_1}))$$
$$r_k = r_4 = (t_{r_4} = \text{“SupportingGoal”}, d_{r_4} = [c_1] \times [c_u])$$
$$c_u = (t_{c_u} = \text{“IncreaseDownhillRecuperation”},$$
$$d_{c_u} = s(t_{c_u}))$$

Where the definitions d_{x_z} of concepts are defined by a function s that assigns certain semantics to the term t_{x_z} respectively. We do not elaborate in detail on the function s as it depends on the language (e.g. based on temporal logics) chosen to formally represent goals.

SUC and CO have a shared goal (expressed by the concept c_1, i.e. reducing their energy consumption), but the CO has an additional goal c_u related to c_1 by r_4. The term t_{c_u} is specific for the electric vehicle as it describes its goal concept to increase downhill recuperation through its electrical engine. Hence, the SUC cannot interpret message m due to a difference in scope, which is, however, known because there exists a known relation r_4 shared by SUC and CO.

T2, unknown difference in scope: In the case of an unknown difference in scope, let us consider the scenario S2, where the SUC knows the goal c_4 to arrive at the destination of travel in an optimal state so that further trips are possible:

$$x_k = c_4 = (t_{c_4} = \text{“OptimalVehicleStateAtDestination”},$$
$$d_{c_4} = s(t_{c_4}))$$

Both SUC and CO prefer having some energy left in the vehicle (either fuel or electric power) after reaching the destination. However, the CO provides some further information item containing a specification of a desired state of charge (a_u) with some relation r_u, according to the following definitions that are unknown to the SUC:

$$r_u = (t_{r_u} = \text{“HasStateOfCharge”}, d_{r_u} = [c_4] \times [a_u])$$
$$a_u = (t_{a_u} = \text{“StateOfCharge”}, d_{a_u} = [0, 1])$$

As a consequence, messages that contain information items using these definitions cannot be interpreted because the IC engine powered vehicles ontology only refers to a certain amount of fuel left instead of a battery charge level.

T3, distinct scope: To illustrate a completely distinct scope of two ontologies, let us assume a scenario S3 in which the SUC has the goal to reach the next fuel station c_5 along the planned route that has to be negotiated with the CO. From the CO, however, the SUC receives a message specified according to an unknown concept

c_3 with an unknown relation r_3 and unknown attribute a_3, which are specific for electric vehicles:

$c_u = (t_{c_u} =$ "ReachNextChargingStation"$, d_{c_u} = s(t_{c_u}))$
$r_u = (t_{r_u} = "HasRechargePlugType'', d_{r_u} = [c_u] \times [a_u])$
$a_u = (t_{a_u} =$ "PlugType"$, d_{a_u} = \{$"IEC 62196-2 Type 1", "IEC 62196-2 Type 2", "IEC 62196-2 Type 3"$\})$

Both, SUC and CO want to reach their next refuel / recharge station. However, each of them has different and distinctly defined concepts, relations and associated attributes, resulting in not understanding each other. In particular, the CO specifies the goal to reach the next charging station, with different relevant attributes such as the required charging plug type.

T4, Inconsistent ontological commitments: To illustrate the case T4, we consider a scenario S4, which is concerned with exchange of messages containing information items that the SUC and the CO can understand according to their ontologies. The SUC receives a message including an information item referencing c_1 (see above) with a known relation r_2 and known attribute a_1, which can be mapped to the SUCs ontology O_{SUC}:

$r_k = r_2 = (t_{r_2} =$ "HasMaxConsumption",
$d_{r_2} = [c_1] \times [a_1])$
$x_k = a_1 = (t_{a_1} =$ "MaxEnergyConsumption",
$d_{a_1} =$ float: $[liters/$100km; max. 100, min.2$])$

However, the ontology of the CO differs in the sense that it includes another, differing definition of the attribute a_1 due to the fact that electric vehicles define their energy consumption in a different unit (kWh/km):

$x_{IOC} = a_1 = (t_{a_1} =$ "MaxEnergyConsumption",
$d_{a_1} =$ float: $[kWh/$km; max. 70, min. 5$])$

Both attribute definitions involve numeric real values in certain intervals, (using "float" as data type), but their semantics and logical implication is different, resulting in misinterpretation and thereby uncertainty.

5.2 Scenarios Depicting Instance Level Uncertainties

I1, semantically inconsistent information: In order to illustrate the case that uncertainties may occur due to semantically inconsistent information, we consider a scenario S5 in which the SUC demands a refinement of the goal c_1 by means of an attribute a_1 containing a concrete floating point number value of the vehicle's maximum desired fuel consumption:

$c_1 = (t_{c_1} =$ "MinimizeEnergyConsumption"$, d_{c_1} = s(t_{c_1}))$
$r_2 = (t_{r_2} =$ "hasMaxConsumption"$, d_{r_2} = [c_1] \times [a_1])$
$a_8 = (t_{a_8} =$ "MaxFuelConsumption",
$d_{a_8} =$ float: $[liters/$100km$])$

However, the CO is only is able to specify the maximum power consumption a_6, which is also known to the SUC but used to specify different concepts than c_1, e.g. related to the vehicle's battery or power outlet. Nevertheless, the CO sends a message m containing an information item i_1 that specifies a_6 as a refinement of c_1 to the platoon.

$a_6 = (t_{a_6} =$ "MaxPowerConsumption",
$d_{a_6} =$ float: $[kW])$
$i_1 = (v_1 = (c_1, a_6), rel_1 = r_2)$

The message i_1 sent by the CO contains a reference to the relation r_2 (using rel_1, see Section 4.1), which, however, is inconsistent w.r.t. the semantic definition of r_1 according to the ontology O_{SUC} of the platoon leader, since the target of the relation r_2 is defined to be the attribute a_1 only.

I2, situationally incomplete information: The case I2 can be illustrated using the scenario S5 (see above) as well. In this case however, instead of using a relation inconsistent to its type definition, the CO simply does not provide the information item that is needed for the SUC to evaluate whether joining is granted or not. This may be due to the reason, that the CO is unable to specify a maximum desired fuel consumption because of its different engine properties. Thus, an attribute that is required to be conveyed in the message m, i.e., $i_2 = (v_2, rel_2 = a_8) \in I_{req}$, is not received by the SUC, and neither is the relation r_2.

I3, situationally inconsistent information: Considering a scenario S6 in which the CO has successfully joined the platoon, the SUC may periodically (e.g. once every 60 seconds) request each following vehicle to report their status, including, e.g. their current destination of travel, or the estimated remaining driving range, in order to monitor the platoon stability, and to identify potential refueling stations, if necessary.

The CO's first report m_1 contained a remaining driving distance of 120 km, as specified by:

$i_3 = (v_3 = 120, rel_3 = a_2)$
$a_2 = (t_{a_2} =$ "DistanceRemaining"$, d_{a_2} =$ float: $[km])$

Afterwards, the platoon has to brake several times. As electric vehicles recharge their battery while braking, the CO is able to actually increase the remaining driving distance, while the IC engine vehicle had a decreasing remaining driving distance. Hence, the CO's next report m_2 contains a slightly increased driving range left $i_4 = (v_4 = 123, rel_j = a_2)$. The SUC, however, uses an ontology O_{SUC} for IC vehicles, and thus requires each vehicle to report a decreased driving range as time proceeds. Hence, based on m_1 the SUC expected a value $v_4 \leq 123$ to be contained in m_2.

I4, missing type membership: In the last scenario S7, coordination messages are exchanged between the SUC and the following vehicles during operation. Due to communication failure or bad logic on the CO's side in general, messages might be corrupted during information exchange. One such corrupted message m could contain an information item i_5 that does not contain a relation to a type description rel_5. For instance, consider the following information item received by the SUC:

$i_5 = (v_5 =$ 5231'33.7"N 1318'50.9"E, $rel_5 = \emptyset)$

Because rel_5 is missing, the SUC is not able to interpret the value v_5 correctly. The SUC could infer from the string format that it is a geographical coordinate, but the semantics to interpret the value is not known. It could be the destination of travel, but also, e.g. the coordinates of the closest refueling or recharging station on the route.

6 Conclusion and Outlook

Collaborative Embedded Systems (CES) collaborate in order to achieve common goals during runtime, where they form groups of Collaborative Systems (CSG). These kinds of interactions require an exchange of information at runtime, since, for instance, commonalities on goals have to be negotiated. Due to fact, that either knowledge or situations can be very heterogeneous for these kinds of systems, several epistemic uncertainties can occur during runtime. In order to consider these uncertainties that can occur during runtime, we defined general sources of uncertainties located in information exchange and introduced a two dimensional classification schema (EURECA), which can be applied during Requirements Engineering for identifying such uncertainties. We used examples from the autonomous driving domain to illustrate the schema. The authors intention is, that EURECA is not limited to this domain only due to its general structure that only requires the ontologies used for the specification of information exchange and a specification of behavioral requirements of the SUC as a prerequisite. It could for instance be used in safety analysis to identify possible failures at an early stage and corresponding counter measures can be incorporated to avoid safety critical scenarios. However, the actual application of the scheme to other domains and the detailed evaluation of this application is up to future work of the authors, which is twofold.

First, we will be concerned with building industry applicable ontologies (e.g. goal ontologies) for the specification of exchanged information in different domains (i.e. autonomous driving, distributed energy resources, adaptable and flexible factories) and apply the scheme for these ontologies to further evaluate it. Based on this, we will extend the intended application of EURECA from the requirements phase to the design and runtime phase (e.g. models at runtime). Hereby, a developer should be enabled to implement uncertainty awareness or even (semi) automatic uncertainty mitigation to CESs.

Acknowledgements The contribution presented in this paper was funded by the German Federal Ministry of Education and Research under grant number 01IS16043 Collaborative Embedded Systems (CrESt).

References

1. Manfred Broy. Engineering cyber-physical systems - challenges and foundations. In Complex Systems Design and Management, pages 1–13. Springer, 2013.
2. Dongyao Jia, Kejie Lu, Jianping Wang, Xiang Zhang, and Xuemin Shen. A survey on platoon-based vehicular cyber-physical systems. IEEE Communications Surveys Tutorials, 18(1):263–284, 2016.
3. Warren E. Walker, Poul Harremoës, Jan Rotmans, Jeroen P. van der Sluijs, Marjolein B.A. van Asselt, Peter Janssen, and Martin P. Krayer von Krauss. Defining uncertainty: a conceptual basis for uncertainty management in model-based decision support. Integrated assessment, 4(1):5–17, 2003.
4. Jens Christian Refsgaard, Jeroen P. van der Sluijs, Anker Lajer Højberg, and Peter A. Vanrolleghem. Uncertainty in the environmental modelling process–a framework and guidance. Environmental modelling & software, 22(11):1543–1556, 2007.
5. Naeem Esfahani and Sam Malek. Uncertainty in self-adaptive software systems. In Software Engineering for Self-Adaptive Systems II, pages 214–238. Springer, 2013.
6. Andres J. Ramirez, Adam C. Jensen, and Betty H.C. Cheng. A taxonomy of uncertainty for dynamically adaptive systems. In Proceedings of the 7th International Symposium on Software Engineering for Adaptive and Self-Managing Systems, pages 99–108. IEEE Press, 2012.
7. Diego Perez-Palacin and Raffaela Mirandola. Uncertainties in the modeling of self-adaptive systems: a taxonomy and an example of availability evaluation. In Proceedings of the 5th ACM/SPEC international conference on Performance engineering, pages 3–14. ACM, 2014.
8. Sara Mahdavi-Hezavehi, Paris Avgeriou, and Danny Weyns. A classification framework of uncertainty in architecture-based self-adaptive systems with multiple quality requirements. In Managing Trade-Offs in Adaptable Software Architectures, pages 45–77. Elsevier, 2017.

9. Javier Cámara, David Garlan, Won Kang Gu, Peng Wenxin, and Bradley Schmerl. Uncertainty in self-adaptive systems: Categories, management, and perspectives. 2018.
10. Man Zhang, Bran Selic, Shaukat Ali, Tao Yue, Oscar Okariz, and Roland Norgren. Understanding uncertainty in cyber-physical systems: A conceptual model. In European Conference on Modelling Foundations and Applications, pages 247–264. Springer, 2016.
11. Antoine Cailliau and Axel van Lamsweerde. Handling knowledge uncertainty in risk-based requirements engineering. In Requirements Engineering Conference (RE), 2015 IEEE 23rd International, pages 106–115. IEEE, 2015.
12. Anna Maria Lombardi. The epistemic and aleatory uncertainties of the etas-type models: an application to the central italy seismicity. Scientific reports, 7(1):11812, 2017.
13. Armen Der Kiureghian and Ove Ditlevsen. Aleatory or epistemic? does it matter? Structural Safety, 31(2):105–112, 2009.
14. Kristopher Welsh and Pete Sawyer. Understanding the scope of uncertainty in dynamically adaptive systems. In International Working Conference on Requirements Engineering: Foundation for Software Quality, pages 2–16. Springer, 2010.
15. David Garlan. Software engineering in an uncertain world. In Proceedings of the FSE/SDP workshop on Future of software engineering research, pages 125–128. ACM, 2010.
16. Kuo-Yun Liang, Jonas Mårtensson, and Karl H. Johansson. Heavy-duty vehicle platoon formation for fuel efficiency. IEEE Transactions on Intelligent Transportation Systems, 17(4):1051–1061, April 2016.
17. Marian Daun, Jennifer Brings, Thorsten Weyer, and Bastian Tenbergen. Fostering concurrent engineering of cyber-physical systems a proposal for an ontological context framework. In 2016 3rd International Workshop on Emerging Ideas and Trends in Engineering of Cyber-Physical Systems (EITEC), pages 5–10. IEEE, 2016.
18. Ana Petrovska and Florian Grigoleit. Towards context modeling for dynamic collaborative embedded systems in open context.
19. Jennifer Brings, Marian Daun, Constantin Hildebrandt, and Sebastian Törsleff. An ontological context modeling framework for coping with the dynamic contexts of cyber-physical systems. In MODELSWARD, pages 396–403, 2018.
20. Ann Hsu, Sonia Sachs, Farokh Eskafi, and Pravin Varaiya. The design of platoon maneuvers for ivhs. In 1991 American Control Conference, pages 2545–2550, June 1991.
21. Kakan C. Dey, Li Yan, Xujie Wang, Yue Wang, Haiying Shen, Mashrur Chowdhury, Lei Yu, Chenxi Qiu, and Vivekgautham Soundararaj. A review of communication, driver characteristics, and controls aspects of cooperative adaptive cruise control (CACC). 17(2):491–509.
22. Michele Segata, Bastian Bloessl, Stefan Joerer, Falko Dressler, and Renato L. Cigno. Supporting platooning maneuvers through ivc: An initial protocol analysis for the join maneuver. In 2014 11th Annual Conference on Wireless On-demand Network Systems and Services (WONS), pages 130–137, April 2014.
23. Piergiuseppe Mallozzi, Massimo Sciancalepore, and Patrizio Pelliccione. Formal verification of the on-the-fly vehicle platooning protocol. In Ivica Crnkovic and Elena Troubitsyna, editors, Software Engineering for Resilient Systems, Lecture Notes in Computer Science, pages 62–75. Springer International Publishing, 2016.
24. Emilia Cioroaica, Thomas Kuhn, and Thomas Bauer. Prototyping automotive smart ecosystems. In 2018 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W). IEEE, 2018.
25. Karl Michael Popp. Goals of software vendors for partner ecosystems–a practitioner s view. In International Conference of Software Business, pages 181–186. Springer, 2010.
26. Majid A. Khan and Ladislau Boloni. Convoy driving through ad-hoc coalition formation. In 11th IEEE Real Time and Embedded Technology and Applications Symposium, pages 98–105, March 2005.
27. Charmaine Toy, Kevin Leung, Luis Alvarez, and Roberto Horowitz. Emergency vehicle maneuvers and control laws for automated highway systems. IEEE Transactions on Intelligent Transportation Systems, 3(2):109–119, June 2002.
28. Maytheewat Aramrattana, Tony Larsson, Jonas Jansson, and Cristofer Englund. Dimensions of cooperative driving, its and automation. In 2015 IEEE Intelligent Vehicles Symposium (IV), pages 144–149, June 2015.
29. Sebastian Ebers, Horst Hellbck, Dennis Pfisterer, and Stefan Fischer. Short paper: Collaboration between vanet applications based on open standards. In 2013 IEEE Vehicular Networking Conference, pages 174–177, December 2013.
30. Xiangheng Liu, Andrea Goldsmith, Sunider S. Mahal, and J. Karl Hedrick. Effects of communication delay on string stability in vehicle platoons. In 2001 IEEE Intelligent Transportation Systems Proceedings, pages 625–630, August 2001.
31. Torsten Bandyszak, Patrick Kuhs, Jasmin Kleinblotekamp, and Marian Daun. On the use of orthogonal context uncertainty models in the engineering of collaborative embedded systems. In Ina Schaefer, Loek Cleophas, and Michael Felderer, editors, Joint Proceedings of the Workshops at Modellierung 2018 co-located with Modellierung 2018, volume 2060 of CEUR Workshop Proceedings, pages 121–130. CEUR-WS.org, 2018.
32. Javier Barrachina, Piedad Garrido, Manuel Fogue, Francisco J. Martinez, Juan-Carlos Cano, Carlos T. Calafate, and Pietro Manzoni. Caova: A Car Accident Ontology for VANETs. In 2012 IEEE Wireless Communications and Networking Conference (WCNC), pages 1864–1869, April 2012.
33. Javier Barrachina, Piedad Garrido, Manuel Fogue, Francisco J. Martinez, Juan-Carlos Cano, Carlos T. Calafate, and Pietro Manzoni. VEACON: A Vehicular Accident Ontology designed to improve safety on the roads. Journal of Network and Computer Applications, 35(6):1891–1900, November 2012.
34. Michele Ruta, Floriano Scioscia, Filippo Gramegna, Saverio Ieva, Eugenio Di Sciascio, and Raffaello P. De Vera. A Knowledge Fusion Approach for Context Awareness in Vehicular Networks. IEEE Internet of Things Journal, 5(4):2407–2419, August 2018.
35. Derlis Gregor, Sergio L. Toral, Teresa Ariza, Federico Barrero, Raúl Gregor, Jorge Rodas, and Mario Arzamendia. A methodology for structured ontology construction applied to intelligent transportation systems. Computer Standards & Interfaces, 47:108–119, August 2016.

36. Jan Bosch and Helena Holmström Olsson. Ecosystem traps and where to find them. Journal of Software: Evolution and Process, 2018.
37. Asunción Gómez-Pérez, Mariano Fernández-López, and Oscar Corcho. Ontological Engineering: with examples from the areas of Knowledge. 2004.
38. Steffen Staab and Rudi Studer. Handbook on ontologies. Springer Science & Business Media, 2010.
39. Franz Baader, Diego Calvanese, Deborah McGuinness, Peter Patel-Schneider, and Daniele Nardi. The description logic handbook: Theory, implementation and applications. Cambridge university press, 2003.
40. Marian Daun, Thorsten Weyer, and Klaus Pohl. Validating the functional design of embedded systems against stakeholder intentions. In Model-Driven Engineering and Software Development (MODELSWARD), 2014 2nd International Conference on, pages 333–339. IEEE, 2014.
41. Constantin Hildebrandt, Sebastian Törsleff, Birte Caesar, and Alexander Fay. Ontology building for cyber-physical systems: A domain expert-centric approach. In 2018 14th IEEE Conference on Automation Science and Engineering (CASE 2018), 2018.