Context modeling for cyber-physical systems

Marian Daun1 | Bastian Tenbergen2

1paluno – The Ruhr Institute for Software Technology, University of Duisburg-Essen, Essen, Germany
2Department of Computer Science, State University of New York at Oswego, Oswego, New York, USA

Correspondence
Marian Daun, paluno – The Ruhr Institute for Software Technology, University of Duisburg-Essen, Essen, Germany.
Email: marian.daun@paluno.uni-due.de

Funding information
Projekt DEAL; German Federal Ministry of Education and Research, Grant/Award Number: 01IS12005C

Abstract
When developing cyber-physical systems (CPS), the context is of vital importance. CPS interact with the world not only through sensing the environment and acting upon it (like embedded systems) but also by communicating with other CPS (like systems in the Internet of Things [IoT]). This means that the context interactions CPS must deal with are much greater than regular embedded or IoT systems: On the one hand, external systems and human users constrain the specific interaction among them. On the other hand, properties of these external systems, human users, and laws, regulations, or standards constrain the way the CPS must be developed. In this paper, we propose a comprehensive, ontologically grounded context modeling framework to systematically explore the problem space in which a CPS under development will operate. This allows for the systematic elicitation of requirements for the CPS, early validation and verification of its properties, and safety assessment of its context interactions at runtime.

KEYWORDS
collaborative system networks, context, context analysis, context modeling, dynamic context, cyber-physical systems, model-based engineering, requirements engineering, validation, verification

1 INTRODUCTION
To develop modern systems that are suitable, safe, and secure for a variety of application fields, proper context consideration is critical and essential. Especially in requirements engineering, the necessity to explicitly document implicit knowledge about the problem domain is of paramount importance. For example, insufficient context consideration can result in key stakeholders being neglected during requirements elicitation, leading to incomplete and conflicting requirements. Moreover, as outlined by Micskei et al., insufficient context consideration can lead to omitting key test cases to establish safe execution at runtime of safety-critical systems. Or as becomes obvious in Cyber Security Body of Knowledge, omitting context concerns may lead to attackers and attack vectors being overlooked, thereby exposing internal assets of security-sensitive systems to malicious external behavior.

Therefore, the field of requirements engineering in particular has suggested approaches for information systems and for self-adaptive systems to elicit and document context information. Yet, with few exceptions, these approaches do not explicitly and specifically focus on the elicitation and documentation of context information but rather on the development of individual systems in general. However, for cyber-physical systems (CPS), proper context consideration is even more important and challenging. This is because CPS are typically closely integrated in their contexts. CPS monitor context measurements by means of sensors, compute necessary control commands, and act upon their contexts by means of actuators, much like embedded systems, but also exchange data with one another like Information Systems, often in physically heterogenous Internet-of-Things (IoT)-type environments. Although IoT systems and CPS are similar in terms of their...
functional collaboration with one another,17–19 by making use of powerful communication networks,20,21 what sets CPS apart is that they are increasingly often entrusted with safety-critical22 or security-sensitive23 functionality.

Traditional implicit and single-system context consideration is hence no longer sufficient to design CPS that will seamlessly, safely, and securely integrate into an existing runtime environment.24 This is because traditional systems development approaches assume a “closed world”25; that is, presuppose that the context the system will be deployed in can be fully understood (and documented) at design time. In CPS development, however, the closed world assumption no longer holds because collaboration between multiple CPS gives rise to functionality within the CPS network that is not restricted to individual systems but can also emerge from their interplay.26–28

To assist in the development of functionally collaborative CPS, this paper proposes a comprehensive, ontologically grounded context modeling framework. The context modeling framework allows developers to systematically explore the problem space in which a CPS under development will operate. We suggest that explicitly documenting facts and assumption about the system’s context (which, in the following, we call “context information”) allows engineers to anticipate what happens when changes occur in the context at runtime. When explicitly documenting this context information alongside the regular engineering artifacts, permissible context configurations can be captured, which are known to retain adequate, safe, and secure functionality.24,29 This can, for example, aid safety analysis and runtime adaptation,30 as we will show in Section 6.

This paper contributes a context modeling framework for CPS that takes the peculiarities of collaborating CPS into account. The paper at hand describes the final iteration of continuously improved context modeling techniques pertaining to certain aspects of CPS development (e.g., structure or function) and is the first work to holistically describe context modeling in a detailed framework. The specific benefit of our approach is as follows:

1. The context modeling framework supports in elicitation and documentation of context information in requirements engineering. Therefore, we particularly distinguish between important knowledge sources and operational context entities with which the CPS will interact.
2. The explicit consideration of different system network configurations (i.e., a CPS possibly partaking in different collaborations) allows improved validation and verification of functional collaboration of networked CPS at design time and runtime. We do not only take changing context into account but also take changing system networks that impact the context and vice versa.
3. Thereby, the context modeling framework is designed to assist during safety analyses and security hardening by systematically investigating the system’s effect onto and inputs from entities in its context.

In this paper, we propose a context modeling framework consisting of detailed, but extendable ontologies as well as notational elements; show the application of ontologies and notations by means of an illustrative running example and to discuss alternative documentation approaches; and demonstrate means of tailoring the context modeling framework to enable analysis through explicit context information documentation.

This paper is structured as follows: Section 2 discusses the background and related work on context theory and context consideration in software engineering. Section 3 introduces a running example which we will use to illustrate our proposed framework. Section 4 proposes ontologies for different types of context information. Section 5 gives examples of the types of context information that can be documented, suggests notational elements, gives example of context models, and describes tool support. Section 6 discusses analytic methods that are enabled through our context modeling framework and how our approach can be tailored for different uses. Section 7 concludes this work.

2 | BACKGROUND AND RELATED WORK

In this section, we discuss the underlying context theory as well as the state of the art in context consideration in software engineering. To this end, we review the related work in Section 2.1 and outline underlying context theory principles in Section 2.2.

2.1 | Context in software engineering

Context consideration is a well-established fundamental approach in software and requirements engineering literature. Already in 1978, Tom DeMarco proposed the structured analysis approach building upon a thorough understanding and modeling of the system’s context.31 Similarly, Jackson’s problem frames approach32 proposes a systematic divide and conquer approach to break down the vision of the system under development into smaller, manageable scopes, with information not pertaining to the current scope of investigation to be part of that scope’s context. From both examples, it can also be seen that in many cases, context consideration is implicit; that is, approaches rarely make explicit difference between system and context. This also holds for some central aspects of mission-critical software development. For example, functional hazard analyses33 used during safety assessment or misuse cases34,35 and used during security engineering consider the system’s effect onto the context (i.e., the way humans can be hurt in case of safety) or the effect of the context onto the system (i.e., how a malicious user gains unauthorized access in case of security).
Literature from the field of requirements engineering makes more explicit efforts to consider context.\textsuperscript{1,36} For example, in goal-oriented approaches,\textsuperscript{37,38} top-level goals are refined into specific requirements with respect to context information. This makes the fulfillment of the goal dependent on the system itself, external systems, or human users. Ontology-based context modeling approaches have also been proposed in the past.\textsuperscript{12,39,40}

However, a prerequisite for explicit documentation in the first place is a proper understanding of the problem domain. Several authors have proposed categorization schemes for context information. For example, Kruchten provided a contextual framework that allows specifying the organizational, regulatory, and developmental domain governing agile systems development.\textsuperscript{41} This framework contextualizes the development process to guide adaptation necessary for novel or safety-critical systems. A system-centric context categorization framework is proposed by Pohl.\textsuperscript{2} Pohl suggests four “context facets”: the subject facet, the usage facet, the IT facet, and the development facet. Furthermore, he distinguishes between three types of “context aspects”: requirement sources, context objects, and properties and relationships of the context object. A particular context aspect can be in more than one facet. A different categorization is suggested by Robertson and Robertson,\textsuperscript{42} who use stakeholder maps to show the stakeholders’ relation to the product. Around the product, which is in the center of a visual diagram they call the context “map,” are three areas. The innermost area is called operational work area and contains the stakeholders who directly interact with the context subject. Around the operational area is the area that of containing business. This area contains those stakeholders who do not interact with the context subject directly yet benefit from it. The outermost area is called the wider environment. It consists of the stakeholders who have an impact on or an interest in the product. The area containing the development team, which they call core team members stretches across all three other areas of the stakeholder map.

Both Pohl and Robertson and Robertson also consider the interaction between the system and elements within its context. Pohl\textsuperscript{2} describes the “usage facet” as those aspects of the system’s context that are in direct interaction with it at runtime, for example, human users or external systems. Robertson and Robertson\textsuperscript{42} visualize the flow of information between the product and “adjacent systems” using a context diagram. This term was also suggested by DeMarco\textsuperscript{31} to describe something quite similar, including human users that receive information from or sends information to the product.

Interactions between system under development and the context are also discussed in terms of Sutcliffe’s domain theory.\textsuperscript{43} Sutcliffe uses generic models to describe meta-domains, grounded domains, and generic tasks. Similarly, McMenamin and Palmer\textsuperscript{44} define planned response systems as systems that react to certain events in their context in a predictable manner. A more behavior-oriented view is adopted by Parnas and Madey.\textsuperscript{45} The authors recommend the use of mathematical relations to specify how a system affects its environment as a response to a particular state change in its environment, an approach that was later extended by Gunter et al.\textsuperscript{46} as well as Weyer.\textsuperscript{51} Weyer describes a semi-automated approach for discovering false assumptions about the context in requirements specifications. He defines three perspectives on the operational context: structural, behavioral, and functional, in accordance with Davis’\textsuperscript{47} suggestion.

### 2.2 Context of cyber-physical systems

CPS have been around for several years now.\textsuperscript{20} Yet, defining the term “cyber-physical system” is exceedingly difficult,\textsuperscript{48} due to different goals and approaches undertaken in various industries. For example, some researchers focus on CPS’ ability to sense their environment and transmit measurements to other systems, like IoT devices,\textsuperscript{14} and apply them to smart manufacturing systems.\textsuperscript{15,16} Others are interested in functional collaboration in system assemblies.\textsuperscript{17} Others again study the interaction of communicating CPS and the effect they have onto their environment, for example, in regulation networks or other types of actuators,\textsuperscript{13} and apply it to smart grid energy systems.\textsuperscript{49} But what all of these different understandings have in common is that CPS use sensors to read values from their environment, use actuators to somehow influence it, and use powerful communication networks to coordinate their efforts.\textsuperscript{20,21}

In this sense, CPS combine aspects of both embedded systems and IoT systems.\textsuperscript{50} Moreover, CPS are increasingly often entrusted with safety-critical and security-sensitive, mission-critical functionality.\textsuperscript{23,51} This poses significant new challenges for the development process of CPS. This is because determining the impact of the CPS (individually or as a network) on the safety of its environment no longer depends on the individual system. Instead, it depends on the functional interplay\textsuperscript{51} that emerges at runtime within the network of communicating and hence collaborating CPS. Similarly, close network interaction gives rise to additional security threat vectors, thus making it more difficult to ensure that the CPS are sufficiently hardened against malicious or erroneous inputs from the environment. This means that functionality does not only take place within one automobile, aircraft, industrial plant, etc., but between several of them.

Safety hazards and security breaches can often be attributed to insufficient context consideration (cf., e.g., McDermid and Lawrence and Gallagher\textsuperscript{52,53}). Objects in the context often pose threats to the system or interacting with them causes some unforeseen malicious behavior. Therefore, it is important to understand, specify, and analyze the context of a CPS to identify safety and security defects as early as possible. However, the CPS context combines the context of traditional embedded systems and of IoT systems, hence making this task difficult. Furthermore, even more challenging, the forming of CPS networks that closely collaborate leads to constant changes in what is part of the system and what needs to be considered as context.
This means that the understanding of what constitutes the context of a CPS becomes much wider than that of traditional embedded systems. The context no longer comprises the specific interactions the CPS will have at runtime with other components within the same system, but also comprises external systems, runtime interactions, physical measurements. Furthermore, context comprises laws, regulations, and other information sources constraining the development process. In consequence, an approach to enable systematic context engineering is necessary.

Yet, as contemporary engineering approaches focus on the development of individual systems or on the definition of inputs and outputs exchanged with the system’s environment, functional collaboration of CPS within dynamic contexts is not explicitly considered. Recently, many research roadmaps have investigated challenges for CPS or self-adaptive systems, but emphasis on systematic context engineering has thus far not been a focus.

### 2.3 Towards a context theory for cyber-physical systems

It can be concluded from Section 2.1 that context plays a major role in systems development. However, as we have seen in Section 2.2, the collaborative and functionally dependent nature of CPS makes this a daunting task. Table 1 summarizes the different notions of “context” proposed in the literature and juxtaposes the relationship to our work. In the following, we derive guiding principles by relating these central notions of “context” to observations from the CPS literature. These guiding principles serve as the basis for the context modeling framework which we will describe in Section 3.

### Table 1 Summary of the term “context” in software engineering literature and relevance to this work

<table>
<thead>
<tr>
<th>Reference</th>
<th>Use of “context”</th>
<th>Relevance to this work</th>
</tr>
</thead>
<tbody>
<tr>
<td>DeMarco(^{31})</td>
<td>The context is the source of data flows. The sources of information are outside the system.</td>
<td>The differentiation between system and context is key to understand the role of an individual CPS within its collaborative network. We adopt this idea as the central notion in our framework.</td>
</tr>
<tr>
<td>Pohl, DeMarco, Weyer, and Nature Team(^{2,11,31,55})</td>
<td>A boundary separates the context from the system under development.</td>
<td></td>
</tr>
<tr>
<td>Jackson(^{32})</td>
<td>Anything not part of the development process must be part of the context.</td>
<td></td>
</tr>
<tr>
<td>Ericson and Sindre(^{33,34})</td>
<td>Safety assessment determines impact of system on context.</td>
<td>The safety of the CPS and the CPS network depends not only on individual systems but also on their interplay. Similarly, security of the network means the interactions must be sufficiently hardened. CPS context consideration must foster assessment of safety and security.</td>
</tr>
<tr>
<td>Sindre(^{35})</td>
<td>Security assessment determines impact of the context onto context.</td>
<td></td>
</tr>
<tr>
<td>Zave and Jackson, Gause, and Krutch(^{1,36,41})</td>
<td>Context retains information about the system to be developed.</td>
<td>“Forces” can be external stakeholders, knowledge sources, as well as interacting organic or inorganic agents, which assert for the CPS network to achieve a common goal. These goals may be organizational or operational but could also restrict the functionality of the CPS network. It is hence essential for a CPS context ontology to uncover all “forces” and their relative constraints on the system, regarding the “domains” or “facets” in which they can occur.</td>
</tr>
<tr>
<td>Strang et al., Bergh and Connix, and Dhaussy et al.(^{12,39,46})</td>
<td>Describe a generic context to foster interoperability of the system in similarly structured contexts.</td>
<td></td>
</tr>
<tr>
<td>Pohl(^{2})</td>
<td>Four context “facets” structure the information about the system relevant for development.</td>
<td></td>
</tr>
<tr>
<td>Robertson and Robertson(^{42})</td>
<td>Context maps define relevant information sources to be considered during development.</td>
<td></td>
</tr>
<tr>
<td>Harper and Zheng(^{56})</td>
<td>A “forces viewpoint” describes decision drivers for software architectures. These forces are aspects within the context that influences the system.</td>
<td></td>
</tr>
<tr>
<td>van Lamsweerde and Yu(^{37,38})</td>
<td>Goals from the context are to be fulfilled by the system under development, either operational (i.e., regarding function) or organizational (e.g., regarding regulations).</td>
<td></td>
</tr>
<tr>
<td>Sutcliffe(^{43})</td>
<td>Generic models can be used to describe the context the system will live in as well as interactions between context and the system.</td>
<td>In addition to the functional interplay between each CPS and the context, the functional interplay within the CPS network must be systematically described. This functional interplay is highly dynamic and overlapping, as the runtime interactions expand and contract. We therefore adopt a differentiation between types of operational interactions as a central idea in our CPS context ontology.</td>
</tr>
<tr>
<td>McMenamin and Palmer, Parnas and Madey, Gunter et al., and Weyer(^{11,44–46})</td>
<td>Describe the context and its states and functions with dedicated models and formalisms to determine a system’s suitability to function in an assumed context.</td>
<td></td>
</tr>
</tbody>
</table>

Abbreviation: CPS, cyber-physical systems.
2.3.1 | Principle 1: Separation between system and context

In software engineering, a dividing line is commonly drawn between what is and what is not within the scope of development.56 The “scope” in this sense means system under development, which may be arbitrarily changed during development. Beyond the scope lies what cannot (or rather: will not) be changed during development. As outlined above, we call this the “context” of the system, and it comprises all objects that are of relevance to the system or its development. To this end, Pohl2 and Weyer11 differentiate the system under development from its context by means of what they call a “system boundary” that harbors the interfaces between them. They also define a “context boundary” that differentiates parts of the context that are relevant from those aspects of the environment that are irrelevant with respect to the scope of inquiry.

2.3.2 | Principle 2: Consideration of different context subjects and their overlapping contexts

To cope with the complexity of modern systems, systems engineering frameworks utilize abstraction layers that allow for decomposing systems into subsystems. Depending on the scope of development, the subject the context acts upon may change, thereby shifting the context boundary.31 This can be done across abstraction layers58 or across viewpoints.59 In consequence, the distinction between system and context depends on the development subject and, hence, must be variable. In context theory, the system, subsystem, function, software, or whatever the context is defined for is typically referred to as the context subject.

2.3.3 | Principle 3: Differentiation between systems, system networks, and functional collaboration

The key characteristic that differentiates CPS traditional systems is that CPS form functionally collaborative networks to achieve a common goal. Systems collaborating in a system network emergently create overall functionality to achieve super goals that the individual systems cannot achieve on their own.60 In the ACC running example from Section 3, only a network of ACC-equipped vehicles can work together to resolve a traffic jam. An important characteristic of system networks that consist of CPS is that they are usually not designed top-down (i.e., by developing the collaborative network) but instead individual systems. This means that the development processes are disjoint and do not explicitly define all possible system networks within which the system under development can be.71,50

As can be seen for CPS networks,26 it can be differentiated between the CPS and its context and the system network and its context. Table 2 summarizes the two different manifestations of context subjects and context objects that are relevant to CPS.

Treating collaborative systems in the system network as context objects, however, ignores the fact that unlike context objects in the conventional sense, these other systems often are not predefined at design time. In fact, many system networks will consist only in part of systems existing at design time. All objects in the collaborative context of a system under development (i.e., the context subject) are able to actively communicate certain information (e.g., states, properties, and parameters) about themselves that are necessary for evaluating how to achieve this goal.61 On the other hand, non-collaborative context objects participate only passively in achieving the system network’s goal.62

2.3.4 | Principle 4: Differentiation between context views for collaborating CPS

Systems collaborating in a system network form direct and indirect associations not only with local context objects (e.g., within the same car61) but also with remote systems, thereby massively increasing the complexity of system interactions.21,50 To alleviate this complexity, the automotive industry has coined the term “safety elements out of context” (SEooC63), which essentially and, for the purpose of this discussion, somewhat misleadingly describes the context subject without a specific context in mind. SEooC specifications of course do not ignore the system’s interfaces with the remainder of the vehicle, but rather specific context assumptions such that the system can be integrated into contexts where the assumptions hold, ignoring aspects that are of no concern for the functional interaction64 or system network composition.65

| TABLE 2 | Different definitions of context subject and context objects for system networks |
|--------:|---------------------------------:|---------------------------------:|
| System network perspective | CPS perspective |
| Context subject | System network | CPS within a system network |
| Context objects | Relevant objects outside the system network | Other CPS in the system network and relevant objects outside the system network |

Abbreviation: CPS, cyber-physical systems.
Particularly for CPS, functional collaboration can be understood as a “common force” that binds functional concerns together (see Harper and Zheng, p. 124). A common way to handle common concerns is the adoption of views onto the interacting systems.

2.3.5 Principle 5: Dynamicity of CPS networks

CPS networks change at runtime. Therefore, CPS that are part of such a network must cope with a dynamic context. In principle, whether a given context is dynamic depends on the timespan considered, respectively, the observation horizon. If the observation horizon is chosen to be infinitely narrow, there will be no change in the collaborative context as well as in the non-collaborative context of another system. If instead the observation horizon is chosen to be wider, several changes may become possible for the context subject.

3 RUNNING EXAMPLE

Throughout the remainder of this paper, we will make use of a cyber-physical adaptive cruise control (ACC) system, like the one described in Mao and Chen, to illustrate our approach. Regular ACCs maintain a driver-set speed and safety distance to the vehicle ahead. This is done by sensing forward obstacles and requesting appropriate engine torque and/or brake force from the engine control unit and brake controller, respectively. A cyber-physical ACC may provide key functionality for an autopilot functionality, effectively enabling the car to drive (more or less) autonomously. In this sense, the ACC is more than just a reactive embedded system influencing the vehicle’s speed based on some sensor input; it also receives control commands from the autopilot. The ACC hence forms a functional collaborative network with other systems inside the car. Further information on such advanced ACC systems with autonomous driving behavior can be found in Mao and Chen and Caramihai and Dumitrache.

In a more advanced scenario, the cyber-physical ACC could be used to resolve traffic jams on highways. To do so, multiple ACC systems of surrounding vehicles interact with one another to determine traffic jams and negotiate a common “convoy” velocity. This behavior results in a higher overall velocity of all involved vehicles in the traffic jam, helps to avoid rear-end collisions, and decreases environmental pollution.

Figure 1 depicts a situation in which this additional function is applied. The figure shows a motorway with two distinct directions. On the bottom direction, two lanes, a traffic jam has formed, while on the upper direction, traffic is flowing normally. Each depicted vehicle possesses an ACC system that provides the function “autonomous driving.” The driver’s desired vehicle velocity \( v \) is set by the driver herself. Due to the established traffic jam, a decreased current velocity \( v' \) is adapted either by the driver or by the ACC’s “follow to stop” or “stop and go” automatic (i.e., decelerating the car till stopped and accelerating again to follow a preceding car). The collaborative nature of the ACC in this example causes \( v' \) to be permanently transmitted to the adjacent vehicles, so that every vehicle is aware of its neighbors’ current velocity. When \( v' \) falls below a certain threshold (i.e., if a traffic jam is approaching and the driver is decelerating), the vehicles’ ACC systems negotiate an approximate average velocity \( v'' \) that every vehicle maintains to build a steady traffic convoy.

For example, the darker car’s regular velocity has been set on 120 km/h by its driver. Due to the traffic jam, the driver was forced to decelerate to 10 km/h. Once its velocity had been fallen below the threshold of 20 km/h, the ACC has begun negotiating a convoy velocity with the other vehicles.
ACC of adjacent vehicles. The ACCs collaboratively adjust all vehicles' velocities to 13 km/h, that is, the heuristically optimal convoy velocity in this situation.

What in requirements engineering is considered, the “context” of a system, that is, the relevant parts of the environment that may constrain its development, becomes more complicated for CPS. In this example, the context of the ACC not only consists of the driver, the engine control unit, the brake controller, and some obstacle sensor, along with laws and regulations governing its operation (as would be the case with traditional embedded systems), but also consists of the autopilot system, as well as surrounding cars that may or may not be equipped with cyber-physical ACCs or autonomous driving capabilities. Moreover, in contrast to embedded systems, where the context is determined during development and the system will always operate in contexts of this prescribed configuration, the context configuration for the cyber-physical ACC can change dynamically at any given moment during operation. This is the case, for example, when another car drives out of communication range, hence terminating the functional dependency between two or more cyber-physical ACCs. In Shanaa et al., we identified the following types of functional dependencies:

- **homogeneous and static**: Every CPS is of the same make and model as every other CPS it interacts with, and the number of interacting CPS does not change. This is the case in, for example, distributed energy prosumer architectures.
- **Homogenous and dynamic**: Every CPS is the of the same make and model as every other CPS, but the number of interacting CPS can change dynamically at runtime. For example, this is the case in automated traffic regulation or smart city infrastructures.
- **Heterogeneous and static**: Every CPS interacts with CPS of different makes, models, or revisions and possibly with other non-CPS, but the number of interacting systems does not change. This is typical for IoT-based systems, for example, in the automation domain, where IoT sensors assist in regulating motors, pumps, etc.
- **Heterogenous and dynamic**: Every CPS interacts with CPS of different makes, models, revisions, or non-CPS and the number of involved interacting system changes at runtime. Examples include the ACC example above.

### 4 | ONTOLOGICAL FOUNDATIONS

In this section, we propose an ontology-centric framework to (i) foster the consistent engineering of multiple concurrently developed interacting systems and (ii) to build the foundation for the use of validation, verification, safety, and security assessment techniques during systems integration. In the following, we illustrate the framework’s underlying ontologies that implement the principles from Section 2.2. We also explain the different types of context information that can be documented using the framework.

#### 4.1 | Context ontology

Principle 1 of context theory (see Section 2.2) requires a distinction to be made between the system that is subject of development and the objects within its context. Specifically, the framework ontology allows distinguishing between parts of the system that are under development and parts of the context that influence the system and must be under consideration but cannot be changed by the developer (see Jackson). Figure 2 shows the core ontological basis to delineate the system under consideration from objects influencing it. This ontology is based on our prior work on context formalization for embedded systems.

In accordance with Principle 1, the system under development is called the “context subject.” It is separated from its context objects therein by means of the subject boundary. The subject boundary is similar to the system boundary proposed by Pohl and Weyer and is

---

**Figure 2** Context ontology from Daun et al.
meant to scope the context subject in the sense that the context subject and everything “within” the subject boundary is the focus of development. Yet, in contrast to Pohl and Weyer, we do not use the term “system” boundary because for CPS development, the context is not bound to an individual system (see Section 1). Instead, functionally collaborating CPS dynamically forms new networks with other CPS, through which new functionality emerges. Therefore, context consideration is important on both levels (system level and CPS network level), which is why we do not limit our understanding of “context” to one system alone, but extend it to allow consideration of all subjects relevant from the perspective of emerging functionality.

Interfacing between the context subject and context objects exist “on” the subject boundary and must either be offered by the context objects or be designed into the context subject. The context is that part of the environment that is relevant to the context subject or its development. It is separated from the remainder of the environment by the context boundary.

In the example from Section 3, if the ACC is the focus of development, it becomes the context subject. In that case, all systems within the same car, for example, the engine control unit, the car’s brake controller, the infotainment system, etc., are part of the ACC’s context. Yet, not the engine control unit, the car’s brake controller, the autopilot feature (whether it is implemented as a separate hardware system or a software component) but also cyber-physical ACCs of other cars belong to the ACC’s relevant context and are thus considered context objects. Depending on the scope of the ACC development and whether a distance sensor (e.g., RADAR or a front-facing camera) is already available in the car (e.g., due to a lane keeping assist system being installed in the same vehicle), the distance sensor becomes part of the context subject. It must hence be developed as part of the ACC development as well. Alternatively, it may be considered a context object the ACC may interface with. In this sense, our context ontology satisfies Principle 2 from Section 2.2.

In addition to external systems or software, context objects may also comprise people (or other organic agents) or abstract information sources. Context objects either interact with the context subject once it has been deployed (e.g., users or other systems) or provide knowledge pertaining to the context subject or its development (e.g., physical constrains or laws). Therefore, the ontology differentiates between two different core types of context: The context may be differentiated the system’s context of knowledge or as the system’s operational context. On the one hand, the context of knowledge comprises all sources that possess information that constrain the system or its development. On the other hand, the operational context focuses on the interaction between the context subject and context objects.

In this sense, our context ontology improves upon the notion of different types of context information proposed by Pohl and Robertson. Although our “context of knowledge” comprises Pohl’s “subject facet,” “IT system facet,” and “development facet” and therefore goes beyond Robertson and Robertson’s “containing business” surrounding the “operational area.” Their “operational area” is comparable with our operational context; however, like Pohl’s “usage facet,” it also includes systems and software, not just human users. Yet, in contrast to Pohl, the requirements engineering process is not considered part of the system's context. This, however, poses the dilemma that information relevant to the system under development uncovered during requirements engineering cannot be adequately represented. For example, legacy systems undoubtedly constrain or inform development of a new system, yet neither interact with the system under development. Because this information is uncovered during requirements engineering, it cannot be subsumed under the system context, unlike the case of Pohl’s context understanding. Our notion of knowledge sources in the context of knowledge explicitly allows differentiating between any information constraining development and information about the system's future interaction at runtime.

In accordance with Principles 2 and 4 from Section 2.2, the context of knowledge and the operational context are not disjoint: A context object interacting with the context subject may require a certain type of interface, as outlined above. Although the context objects and their interaction with the context subject are part of the operational context, the interface specification enabling this interaction is part of the context of knowledge, as it constrains how the context subject must be built to be compatible with the context object. In the following sections, we explain the subontologies detailing the makeup of the context of knowledge (see Section 4.2) and the operational context (see Section 4.3).

### 4.2 Context of knowledge

As outlined above, the context of knowledge comprises context objects that possess information pertaining to the context subject or its development and must hence be adhered to by the context subject. Such information is contained in a variety of documents such as laws, organizational rules, or standards, but also stakeholders and other types of constraining information (e.g., physical dimensions or natural laws), which in the following, we refer to as “knowledge sources.” Knowledge sources may also comprise information about systems the context subject interacts with (i.e., are in the “operational context,” see next section), as outlined above. In our example from Section 3, this comprises systems the ACC communicates with the engine control unit using a specific bus (e.g., FlexRay) or with remote cars through protocols (e.g., Zigbee). Such context information can typically be found in the engineering artifacts of systems in the context subject’s operational context, making these engineering artifacts “knowledge sources” for the context subjects. For example, the interface specifications of the ACC and the engine control unit must be consistent regarding their interconnectivity. An ontology based on Daun et al. that structures different types of knowledge sources is shown in Figure 3.
The relations between several kinds of knowledge sources and the context subject are depicted. Types of knowledge sources include but are not limited to documents (e.g., laws, organizational regulations, engineering standards, etc.), systems in the operational context (e.g., external systems interacting with the context subject), or stakeholders. Stakeholders may be customers, users, members of the development team (e.g., the requirements engineer, the tester, the architect, the business analyst, etc.), and other experts on the application domain that pertains to the context subject.

In the ACC example, not only relevant experts could be automotive engineers but also the driver. The context subject or its development may further be constrained by government laws, organizational regulations, or standards, making these examples of knowledge sources as well. In the ACC example, not only relevant documents could be the aforementioned laws pertaining to road traffic regulations in the country for which the system is developed but also relevant standards such as ISO 26262.

Furthermore, because the context subject will eventually be physically manifested, physical laws will affect its operation and are therefore knowledge sources as well. For example, this could be the laws of thermodynamics that must be considered during development of the ACC, for example, to not overheat the brakes.

4.3 Operational context

The operational context subsumes all those context objects that interact with the context subject once it is deployed. On the one hand, this may comprise not only organic agents such as human users but also synthetic agents, such as external systems. Moreover, for collaborative CPS, in accordance with Principle 3 from Section 2.2, this may also include remote systems of a similar type that are functionally collaborating with the context subject. Due to the dynamic nature of runtime configurations of the operational context, precise knowledge about the runtime context can only by estimated at design time. Figure 4 shows the ontological basis for the operational context, which we previously proposed in Daun et al., and allows documenting design time assumptions about the runtime configuration of the operational context.

In accordance with Principle 4 from Section 2.2, the operational context differentiates between different concerns within the context to allow engineers to adopt different perspectives during system development. Our operational context ontology therefore consists of three perspectives: structural, functional, and behavioral.

In the structural perspective, the focus is on physical entities that interact with the context subject, their dependencies with the context subject, as well as their dependencies among one another. Context objects in this perspective are called “context entities” and include but are not limited to any type of static-structural information, such as different types of hardware (e.g., ECUs, actuators, sensors, or bus transceivers), agents (e.g., subsystems, external systems, or human users), as well as dependencies between them. In the example from Section 3, the brake controller and the own vehicle are examples of context entities from the perspective of the ACC as the context subject. Dependencies between the ACC and the context entities include, for example, the actuation of brakes and negotiated convoy speeds. However, generalization, aggregation, and composition dependencies can also be depicted. For example, ACC and brake controller can be aggregated to an abstract class of “driver assistance system.”

The functional perspective abstracts from physical entities and focuses on their respective services. Because it is common practice in function-centered engineering to develop abstract logical functions before they are partitioned into technical architectures, the functional perspective allows the documentation of dependencies between the functions of the context subject with services in its context that can be
accessed. In the functional operational context, context objects hence are called “context functions.” The functional perspective can also be used to document data flows and control flows between the context subject and these context functions. In our ACC example, the ACC function “maintain distance” makes use of the function “brake.” Because “brake” is provided by the brake controller and not the ACC, it is considered a context function from ACC’s point of view. In this sense, by abstracting from the physical embodiment of the context entities that provide a functionality, the context subject can rely on the function “negotiate convoy speed” not being physically attached to a single device within the context, but an emergent functionality of the entire CPS network, regardless of its physical composition (in accordance with Principle 5 in Section 2.2).

The operational context in the behavioral perspective depicts externally observable states of all context entities and all context functions. In this perspective, context objects are therefore called “context states.” The behavioral operational context can be used to enrich the structural perspective and the functional perspective by allowing developers to document the specific anticipated behavior of context entities and/or context functions, respectively, agnostic of the involved context objects. The behavioral operational context is hence a black box view on the behavior of other subsystems of the CPS network and, as appropriate, the corresponding context subject states (see Principles 3–5 in Section 2.2). In the ACC example from Section 3, the “maintain convoy speed” behavior allows for slowing down some cars while speeding up others. The ACCs within the collaborative network of the ACC provide this behavior to the context subject ACC. Furthermore, the network behavior also allows, for example, for determining road hazards or incompatible functional incompatibilities (e.g., when a car without cyber-physical ACC is amidst the convoy), thereby allowing developers to identify potentially unsafe states of the functional network and define suitable degradation modes that maintain safety of the intended functionality.80

Figure 4  Subontology of the operational context from Daun et al.73

5 | NOTATION AND EXAMPLES

In the previous section, we introduced the ontological basis for our context modeling framework. We named several examples of context information that may be documented therein. That list is meant neither to be exhaustive nor definitive. Instead, the ontologies suggest a categorization
framework that is meant to assist the elicitation process of context information and enables its documentation in a way that satisfies the principles underlying context theory derived from the related work in Section 2. Documentation itself may take many different forms. In its simplest form, a simple bullet list for each context perspective (i.e., behavioral, functional, structural, or context of knowledge) may suffice. Alternatively, we can implicitly document context information as constraints in logic formulae. However, to leverage the inherent ability of model-based development to overcome challenges due to the inherent complexity of CPS, we propose a documentation format employing graphical models in the following. For each context perspective, we propose an example list of model elements motivated by our explanations from Section 3 along with a brief description and visual notational element. Afterwards, we give example models and related them to the running example from Section 3. Furthermore, in Section 5.5, we discuss tool support to assist developers in creating context models from all perspectives using the below notation in a way that we hope can easily be integrated in existing industrial tool chains.

5.1 Documenting the context of knowledge

The context of knowledge is like the “context map” proposed by Robertson and Robertson and mainly contains dependencies between knowledge sources. In this sense, the context of knowledge contains static (in contrast to dynamic) information. In consequence, any type of diagram depicting static-structural information can be used to depict the relationship between the context subject and knowledge sources, as well as between knowledge sources (e.g., ER, UML class, or SysML block definition diagrams).

Table 3 shows examples for modeling elements that can be used in context of knowledge diagrams. The table lists the element name, states a brief description of the knowledge source, and suggests a notation.

Table 4 summarizes the dependency types for the context of knowledge. It lists the element’s name, shows the notation, and states a brief description. Those modeling elements added to the table, which are not part of Table 3 or Table 4, are unchanged basic UML elements and therefore simply added in the toolbox definition to appear in the modeling environment.

An example of a context of knowledge diagram applied to the ACC running example from Section 3 is shown in Figure 5. The figure shows an excerpt of the context of knowledge, specifically regarding knowledge sources pertaining to the identification of unreachable, safety-critical state. For example, this context of knowledge diagram shows that ISO 26262 is a relevant standard for automotive development and mandates that the state space of a system be verified during development. However, the standard does not strictly mandate how this needs to be done. Therefore, the developer conducts a Markov analysis, the results of which are summarized in another engineering artifact.

Although, as outlined above, other modeling languages (e.g., UML class diagrams) could be used to depict the same information, using the notation suggested in Figure 5 assists in distinguishing between context information and the context subjects. Especially, because context of knowledge diagrams tends to accumulate a large number of associations between the context subject and the knowledge sources, it may be useful to generate views which hide less relevant information while keeping the most important information for a particular purpose. This allows generating a “landscape” of knowledge sources, as we will outline in Section 6.

The context of knowledge has been successfully applied to several case examples, including an airborne collision avoidance system.

5.2 Documenting the structural operational context

As outlined in Section 4.3, the structural operational context describes static information about context objects, similarly to static diagrams used during systems engineering. In this sense, just like context of knowledge diagrams, structural operational context diagrams ontologically describe relationships between context entities that interact with the context subject at runtime. To visually depict this information, any type of structural diagram can be used.

For this purpose, we recommend using UML class diagrams or SysML block definition diagrams. Because both implement the meta object facility (MOF), both should be extended with the stereotypes described in Tables 5 and 6. In contrast to context of knowledge models, structural operational context models should therefore make use of the standard modeling elements in class or block diagrams (e.g., generalization, associations, association classes, etc.). Yet, to adhere to Principle 1 in Section 2.2, we recommend a more specific differentiation of the MOF metatype “class” into “context entity” and “context subject,” as outlined in Figure 4 and as shown in Table 5. This is necessary to visually differentiate the two concepts.

Similarly, we recommend introducing two subtypes for the MOF meta-class “association,” namely “context entity association” and “context subject association.” The former may be used to make visually obvious the relationships between context entities interacting within the operational context in a way that indirectly impacts the context subject. The latter can be used to depict context entities and how they directly affect the context subject, as shown in Table 6.

Figure 6 shows an excerpt of the structural operational context of the cyber-physical ACC from Section 3. As can be seen, typical UML associations are used to depict the interaction relationships between the context subject and the context entities. These are called “context subject
associations” and may bear an access direction in addition to the UML default reading direction near the association label. In contrast to typical system models, the structural operational context also depicts context entity associations between two or more context entities (i.e., by using UML’s n-ary associations). Such relationships depict exchange of control or data flows between context entities that are relevant for the functionality of the context subjects. In the figure below, this is the case between the CPS transceiver, which is some type of radio interface (e.g., wifi or Zigbee communication) to specify that this transceiver is used for wireless communication with other cyber-physical remote cars. The ACC context subject then makes use of this transceiver to negotiate convoy maneuvers with remote cars.

### Table 3: Modeling elements of context of knowledge diagrams

<table>
<thead>
<tr>
<th>Modeling element</th>
<th>Description</th>
<th>Sample notation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Context subject</td>
<td>This modeling element names the CPS under development that is constrained by knowledge sources.</td>
<td><img src="image" alt="Context Subject1" /></td>
</tr>
<tr>
<td>Development process</td>
<td>This modeling element denotes aspects of the development process that constrain the context subject, such as regulation that mandates certain activities to be carried out.</td>
<td><img src="image" alt="Development Process1" /></td>
</tr>
<tr>
<td>Document</td>
<td>This modeling element denotes a relevant document in the context of a subject. This includes any type of artifact with relevant information, such as laws, organizational regulations, or even post-it notes written by developers.</td>
<td><img src="image" alt="Document1" /></td>
</tr>
<tr>
<td>Engineering artifact</td>
<td>This modeling element denotes a relevant engineering artifact of external systems relevant to the context subject. This includes requirements specification, architecture specifications, interface specifications, test cases, code, analysis results, etc.</td>
<td><img src="image" alt="Engineering Artifact1" /></td>
</tr>
<tr>
<td>Physical Law</td>
<td>This modeling element denotes a relevant physical law governing the functionality of the context subject (e.g., gravity) or limiting it (e.g., ambient heat).</td>
<td><img src="image" alt="Physical Law1" /></td>
</tr>
<tr>
<td>Stakeholder</td>
<td>This modeling element denotes a relevant stakeholder in the context of a subject, such as developers, users, customers, testers, etc.</td>
<td><img src="image" alt="Stakeholder1" /></td>
</tr>
</tbody>
</table>

*Abbreviation: CPS, cyber-physical systems.*

### Table 4: Dependency types of context of knowledge diagrams

<table>
<thead>
<tr>
<th>Modeling element</th>
<th>Description</th>
<th>Sample notation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Development constraint</td>
<td>Development constraints impact how the CPS can be developed, e.g., through the application of specific methods or processes, or by imposing invariants or legal limitations.</td>
<td><img src="image" alt="Development Constraint" /></td>
</tr>
<tr>
<td>Solution constraint</td>
<td>Solution constraints impact the implementation of the CPS directly, e.g., by requiring certain input values or output commands.</td>
<td><img src="image" alt="Solution Constraint" /></td>
</tr>
</tbody>
</table>

*Abbreviation: CPS, cyber-physical systems.*
Structural operational context models have been used successfully in embedded systems-type applications, but also industrial plants, and for a cyber-physical collision avoidance system.

### 5.3 Documenting the functional operational context

The functional operational context depicts the functional collaboration of context functions, that is, services offered within the context that the context subject may either consume inputs from or produce outputs to. Dynamic, function-centered diagram types such as data flow diagrams...
used in DeMarco’s structured analysis approach or UML/SysML activity diagrams are suitable to document the functional interplay between the system and its context. Like for structural operational context diagrams (see Section 5.2), we recommend subtyping the MOF meta-class “activity” and “opaque action,” respectively, to differentiate between context functions and functions of the context subject. Note that like in structural operational context diagrams, we use dashed line to depict context objects. Yet, in contrast to structural operational context or context of knowledge diagrams, the context subject bears no black bars. This is because the subject of interest in functional operational context diagrams is the functions offered to the context by the context subject, not the context subject as a static-structural entity. In consequence, it may be desirable, depending on the CPS under development, to depict more than one context subject function in the functional operational context. Table 7 shows the specialized notational elements suggested to be used in activity diagrams to depict functional operational context elements.

Functional operational context diagrams place special emphasis on dependencies between the CPS and collaborative systems in the context, in accordance with Principle 3 in Section 2.2. Yet, in contrast to structural operational context diagrams, these dependencies are not ontological, but depict control or data flows. This could mean that, for example, execution of a context subject function is prevented at the request of a context object. To depict such dependencies, we introduce the modeling element “dependency” and “dependency edge,” which specialize the MOF meta-classes “class” and “control flow,” respectively, and as shown in Tables 7 and 8. Like data objects in activity diagrams that are static-structural information derived from, for example, class diagrams, data objects are depicted between two control flows to indicate that this data

---

TABLE 7  Modeling elements of functional operational context diagrams

<table>
<thead>
<tr>
<th>Modeling element</th>
<th>Description</th>
<th>Sample notation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Context function</td>
<td>This modeling element represents a function in the context of the context subject. This function is either available to the context subject as a service or makes use of a function offered by it.</td>
<td><img src="context_function_example.png" alt="Context Function Example" /></td>
</tr>
<tr>
<td>Context subject function</td>
<td>This modeling element represents context subject functionality, i.e., a function that interacts with other functions within the context, either by receiving inputs from or providing outputs to the context.</td>
<td><img src="context_subject_function_example.png" alt="Context Subject Function Example" /></td>
</tr>
<tr>
<td>Data object</td>
<td>This modeling element is the typically known data object from UML. It is used to denote that a control flow transmits non-trivial information, hence turning the control flow into a data flow.</td>
<td><img src="data_object_example.png" alt="Data Object Example" /></td>
</tr>
<tr>
<td>Dependency</td>
<td>This modeling element notes a dependency between functions. This is the case when one function influences the execution of another function, e.g., if one function prevents the other.</td>
<td><img src="dependency_example.png" alt="Dependency Example" /></td>
</tr>
</tbody>
</table>
object is flowing between two activities, two dependency edges connect the context function with a dependency type and the dependency type with the context subject function (or the other way around). Associated labels can be used to describe the nature of the dependency.

An example depicting a functional operational context diagram is shown in Figure 7. All context functions are represented agnostic of which specific context entity implements it. Yet, only those context functions are shown directly interact with some function of the running example ACC. Conceptually, these can be understood as “services” available in the context the context subject may use, or “services” the context subject offers to the context. Some of these may have direct counterparts within the context (e.g., “negotiate convoy speed” in Figure 7), although others may be interface functions (e.g., “getter” or “setter” functions, like “set desired speed” in Figure 7). Again, others may not be directly available within the context nor within the context subject but represent functionality that emerges from the collaborative interplay between the context subject with context functions. An example of such a phenomenon is the “make room” maneuver. This emergence is depicted as an unlabeled dependency edge.

We have successfully applied functional operational context diagrams to depict collaborative networks for CPS. Specifically, we have shown an involved runtime interaction between smart ACCs resolving a traffic jam in Tenbergen et al.59 We have also successfully applied functional operational context diagrams to validate the control logic of desalination plants against process requirements in Brandstetter et al.86

### 5.4 Documenting the behavioral operational context

In contrast to the structural and functional operational context, the behavioral operational context describes externally observable context states. “Externally observable” in this case means from the perspective of the context subject. The behavioral operational context diagrams describe the

<table>
<thead>
<tr>
<th>Modeling element</th>
<th>Description</th>
<th>Sample notation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dependency edge</td>
<td>This modeling element links a dependency to a function available within the context.</td>
<td><img src="image" alt="Dependency edge" /></td>
</tr>
<tr>
<td>Object or control flow</td>
<td>This modeling element is the typically known object flow from UML to exchange data objects that are shared between the context subject and the context functions.</td>
<td><img src="image" alt="Object or control flow" /></td>
</tr>
</tbody>
</table>

**Figure 7** Functional operational context of the adaptive cruise control (ACC)
state space of a context entity (from the structural operational context) or a context function (from the functional operational context). In consequence, the context subject is not usually depicted in behavioral operational context diagrams. However, those states that the context subject assumes in response to stimuli from the context may be depicted. Specifically, these context subject states must be externally observable from the perspective of an interacting context object, yet internal states of the system are not depicted. Moreover, behavioral operational context diagrams can be used to depict the state space of the collaborative CPS network, agnostic of its physical manifestation (see also Principles 3 and 5 in Section 2.3). Therefore, these diagrams are quite close to regular stateful representations, such as finite state automata, Harel Statecharts, or UML/SysML state machine diagrams. Nevertheless, as suggested in Tables 9 and 10, we propose using a visual differentiation of dashed contour lines for context states and transitions between them, which is in keeping with Principle 1 from Section 2.3, similarly to other operational context models.

The state space of the cyber-physical ACC convoy is depicted in Figure 8. More precisely, Figure 8 depicts the overlap of externally observable context states that given an event from the context causes a state transition in the context subject. In contrast to functional and structural operational context diagrams, behavioral operational context diagrams are notationally similar to regular state machine diagrams depicting the system’s internal states. We make use of the same dashed-border notation that we use in functional and structural operational context diagrams to emphasize that certain state transitions are triggered by external events or states. In the figure below, this resulted in a state machine diagram consisting of one context subject state, “convoy formed,” that is subdivided into three concurrent substates, one for node management, one for speed negotiation, and one to make room for intra-convoy overtaking. In “make room maneuver,” context states depicting the entire convoy is shown, as nodes may request that some car makes room for another vehicle to, for example, prepare to leave the convoy near a highway exit ramp. If the context subject is the request receiver and a lane boundary was detected, the ACC will steer the car closer to the lane boundary and temporarily decrease clearance between cars, as to allow another car to merge into the emerging gap. As nodes constantly join or leave the CPS network, requiring node IDs to be propagated through the network such that each node has the most updated list of participating nodes available, which is shown, for example, in “node management.”

State-based representation of context information similar to the behavioral operational context is the foundation of coherence checks between system context and system specification.11,72 Most recently, we have applied behavioral operational context information to identify and resolve off-nominal behaviors of collaborative systems in an extension or the SACC approach.87

### Table 9: Modeling elements of behavioral operational context diagrams

<table>
<thead>
<tr>
<th>Modeling element</th>
<th>Description</th>
<th>Sample notation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Context state</td>
<td>This modeling element represents an externally observable state of an entity or function within the context, or the CPS network at large. Internal context entity or context function states are abstracted into their external behavior.</td>
<td><img src="image" alt="Context State1" /></td>
</tr>
<tr>
<td>Context subject state</td>
<td>This modeling element represents a state of the context subject that is externally observable from the perspective of some context object.</td>
<td><img src="image" alt="Context Subject State1" /></td>
</tr>
</tbody>
</table>

Abbreviation: CPS, cyber-physical systems.

### Table 10: Dependency types of behavioral operational context diagrams

<table>
<thead>
<tr>
<th>Modeling element</th>
<th>Description</th>
<th>Sample notation</th>
</tr>
</thead>
<tbody>
<tr>
<td>System transition</td>
<td>This modeling element links context subject states, i.e., transitions between internal states of the context subject.</td>
<td><img src="image" alt="System Transition" /></td>
</tr>
<tr>
<td>Context transition</td>
<td>This modeling element depicts events that cause the context subject to carry out a state transition.</td>
<td><img src="image" alt="Context Transition" /></td>
</tr>
</tbody>
</table>

5.5 | Tool support

As outlined above, the ontologies proposed in Section 3 must be understood as a categorical schema, not a definitive list. Similarly, the notations and examples discussed in this section merely denote proposed ways to document the ontological information from Section 3. Nevertheless, model-based engineering is considered a particularly adequate solution for the challenges due to the inherent complexity of distributed and
complex systems, especially in early phases of development. Because adequate tool support for modeling approaches is a quasi-prerequisite for industrial adoption, we propose a prototypical implementation of our context modeling approach in the following. We implemented the context diagrams suggested above as an “MDG Technology” for SparxSystems Enterprise Architect, as closely resembles that of UML profile definition.

We followed Laplante et al.’s lightweight extension approach in doing so. The extension mechanism in Enterprise Architect requires the creation of some technical profiles in addition to the conceptual profiles dedicated to each diagram type. These technical profiles comprise the modeling elements for each of the four context types and a toolbox profile for each diagram type. Figure 9 gives an overview over the technical and conceptual profiles.

5.5.1 | Modeling elements

For each context type, there is a dedicated profile implemented for bundling a set of stereotyped extensions from the UML base elements. These stereotypes are derived from the described context types respectively and modified regarding their appearance by creating shapes according to the ideas in Sections 5.1–5.4. In Figure 9, the four modeling elements profiles are “SysML context of knowledge modeling profile,” “SysML structural operational modeling profile,” “SysML functional operational modeling profile,” and “SysML behavioral operational modeling profile.” An
5.5.2 | Diagram types

Each context type from Section 3 is assigned a separate diagram type as an extension to the known base diagram “Diagram_Logical.” Due to a nuance of Enterprise Architect's MDG system, this is also the case for non-logical dynamic diagrams such as functional and behavioral operational context models. This allows using an adjusted namespace for each context type in the diagram header in the top left corner and simplifies the recognition of the diagrams content. The selection of each diagram type is anchored to the user’s selection of modeling a specific context type, as shown in Figure 11.
For each diagram type, a dedicated toolbox is implemented as a technical profile to provide the user with user interface components comprising the set of modeling elements suitable for the specific context diagram. The modeling elements for a certain toolbox are a combination of the derived modeling elements from Section 5.1–5.4 and a set of suitable UML elements. These UML elements are directly added to the toolbox. The four toolbox profiles are “SysML context of knowledge modeling,” “SysML structural operational context modeling,” “SysML functional operational context modeling,” and “SysML behavioral operational context modeling.” The latter is shown as an example for all toolbox profiles in Figure 12.

6 | EVALUATION

We evaluated the context modeling framework using different industrial case examples. Therefore, we were either provided with industrial sample specifications or applied the context modeling framework in close collaboration with industry partners to their current development project. As our context modeling frameworks has mainly evolved over several publicly funded projects, so has its evaluation. Application examples have been provided by industry partners involved in the projects. Results of the application of the context modeling frameworks were typically intensively discussed with industry partners. As suggested by design science research methods (cf., e.g., Wieringa90), continuous evaluation commonly resulted in changes to the framework ending in the current final state of the context modeling framework as presented throughout this paper.
More information about the general evaluation approach and the provided industrial case systems can be found in Jedlitschka and Löwen. In the following, we will show the evaluation of the context modeling frameworks using two concrete application examples.

### 6.1 | Application example 1: Context views for CPS collaboration

The context ontologies from Section 3 and notation from Section 5 are meant to be used during the development process of a single CPS that is meant to join a collaborative network of CPS at runtime. In doing so, their benefit lies in eliciting context information and documenting them in a structured form. However, a drawback may be that this essentially means context diagrams constitute another engineering artifact to be produced for the context subject. Doing so allows creating certain “views” on the context subject that was otherwise not possible, which is based on Principle 4 from Section 2.2. In this paper, we discuss building views for the context of knowledge. For another example of context views, we introduced views on the operational context to detect emergent functional interplay in Tenbergen et al. Functional operational context models can be used to develop functional “views” on the CPS network, the functions offered therein, and the collaborative behavior that emerges through the interplay. In this sense, the context for each function available within the CPS network consists of the overlap of the context of individual CPS. This allows constructing a functional “view” (cf. ISO/IEC/IEEE 42010) to depict functional dependencies between the CPS participating in the collaboration and the functions they make accessible to the network. This view then explicitly illustrates necessary preconditions that must be met for the CPS’ functionality to be permissible, for example, when a context function prevents the execution of a context subject function. As a simple example, a CPS network consisting of cars equipped with cyber-physical ACC can be considered. The cars and the CPS network they participate in attempt to form a platoon driving with a negotiated speed. However, the network may consist of a vehicle towing a trailer. Because the convoy exists in a jurisdiction where cars with trailers may not exceed a velocity of 80 km/h (as is the case throughout most of Europe, an information item that is most appropriately documented in a context of knowledge diagram), this limits the context function “accelerate convoy” to an upper limit of 80 km/h throughout the convoy.
6.1.1 | Structuring context of knowledge landscapes

Generating views is not limited to the functional operational context as shown in Section 5.3 but can in principle be applied to all types of contexts. One of the core advantages of model-based documentation of context information is being able to create views onto the model of a system, or in our case, views onto the model of the context, particularly when models become too complex. Because the context of knowledge is prone to accumulate a magnitude of associations between the context subject and knowledge sources, it may be feasible to generate views that hide less relevant information while keeping the most important information for a particular purpose. We have discussed this mechanism at length in Daun et al. In the following, we apply this principle to the adaptive cruise control example from Section 3.

Recall the context of knowledge diagram for the adaptive cruise control system shown in Figure 5. The following Figure 13 shows a similar context of knowledge diagram for the autopilot feature mentioned in Section 3. As can be seen, the autopilot feature is not a “system” per se, but a functionality is realized by reusing existing components and their control commands. These components may be entirely software based, without a dedicated physical manifestation. The diagram shows that some of the knowledge sources are, therefore, shared with the ACC, for example, the “accuracy” of the lateral sensors and radar that constrain the solution in the sense that the autopilot feature, because the autopilot needs to consider their accuracy to calculate actuator commands, but possibly in a different way than the ACC.
Moreover, the ACC and the autopilot are within each other’s context and show context dependencies between shared engineering artifacts. This means that the context of knowledge of both systems is shared in some respect. This allows adopting a context-centric (rather than context subject-centric view) on the context of knowledge, as shown in Figure 14. Doing so provides a more detailed view on the relationship between the context subjects and a single system in the context, along with the shared knowledge sources impacting them both. Figure 14 contains two
context subjects (i.e., the ACC and the autopilot), but only shows knowledge sources that are shared between them. These knowledge sources must be identical but can have different constrains on the context subjects. For example, the knowledge source “radar accuracy” depicts the accuracy of the same radar unit, but its impacts on the context subjects differ. This is also the reason why the name-identical knowledge source “Markov Analysis Results” is not depicted in Figure 14, as the analysis results from the ACC and autopilot feature are two different engineering artifacts.

Representing a system in the context by its various engineering artifacts is especially useful if the system in the context is being developed at the same time because the system in the context and its engineering artifacts are likely to change frequently during this period.

6.1.2 | Benefits of context views

Documenting context information in a sense adopts a system-centric view over the world surrounding the system under development. In other words, by using the above context ontologies, the world surrounding the context subject can be understood as a system itself, upon which we can generate certain views to the benefit of the development process for the context subject. These are as follows:

**Delineate the scope of development**

Documenting knowledge sources, their relationships to one another, and shared knowledge sources impacting multiple concurrent development processes improves the ability to clearly define the scope of development. By making knowledge sources comprehensively persistent, the decision-making process regarding the scope of each system under development can be supported, and conflicts can be resolved. Furthermore, using context information, the risk of inconsistencies of subsystems that are developed in parallel on multiple layers of abstraction can be substantially reduced, as each subsystem can be understood as being part of the context of each other subsystem.

**Compatibility checking of a system within the network**

View-centric context modeling is the use for compatibility checking of the single CPS with the other partaking CPS and with the CPS network. On one hand, compatibility checking between the system and its context is supported as it is clear what the non-collaborative context is, and how the collaborative context differs from it. For the ACC case example, this means that it must be checked whether the ACC is compatible with other, remote CPS and locally connected systems (e.g., connected through CANBUS or FlexRay).

**Definition and validation of CPS network safety goals**

Through safety analyses, such as the failure modes and effects analysis (FMEA), potential failures are identified, and safety goals and safety requirements are defined as mitigations to prevent the occurrence of these failures. This is traditionally done for individual systems. However, generating context views in the manner outlined above considers the different aspects of functionality belonging to the CPS as well as to the CPS network, hazards from the collaboration can be detected during safety analyses by considering the overlap of system functions. Safety claims can be made for system network functions. In Section 6.2, we shall outline an example of an FMEA in this sense.

**Detection of emergent behavior**

In the interplay of multiple CPS, emergent behavior describes behavior, which does not result from one single system but emerges from the interplay of multiple systems. Obviously, this is a common situation in CPS networks, where the partaking systems shall contribute to an overall CPS network goal. However, it is often the case that behavior emerges in the interplay, which was not intended during requirements engineering and system specification. Detected emergent behavior might either hint at defects in the specification or at additional requirements that have been undetected so far.

6.2 | Application example 2: Context-driven safety assessment

In Section 4.2, we have shown how the context of knowledge comprises knowledge sources that constrain or inform the development of the CPS. To a large extent, these knowledge sources may be engineering artifacts of the context subject or of other systems. Doing so, in a way, transforms the context of knowledge into a library of engineering artifacts created for the context subject; see Section 6.1.1. In Figures 3 and 4, we limited our ontologies to artifacts that directly describe the context subject (and context objects) requirements. However, engineering artifacts are not limited to those types of documents but could entail others as well, such as test cases, market analysis documents, DevOps configurations, or safety cases. In this section, we show by means of the activities conducted and artifacts produced during safety assessment how the context modeling framework can be extended to account for non-functional context information.
6.2.1 | Ontology extensions

Safety assessment involves the systematic identification of possible hazards of a system to its environment (e.g., through hazard analyses) and the also collection of evidence that hazards are sufficiently unlikely to occur during operation. The goal of these activities is to produce a defensible argument in support of the claim that the system is safe. This argument is often referred to as a “safety case” and, much like other engineering artifacts or context information, can be represented visually, textually, or even formally. While the representation plays a subordinate role, the conceptual meaning of these artifacts requires a safety case to ontologically subsume at least “hazards” and “evidence.” Yet, because repeatability plays a major role in assessing the risk the context subject poses to its environment, different stakeholders may evaluate risk and hazard likelihood differently. Therefore, to incorporate safety analyses and safety argumentation into context diagrams, the context of knowledge and operational context can be extended as shown in Figure 15.

The extensions made for the context of knowledge subontology are depicted. The knowledge sources “safety engineer” and “certification authority” were added as types of stakeholders as it is not only important who constructs the safety case (i.e., the safety engineer) but also who certifies it (i.e., the authority). Moreover, knowledge sources pertaining to the “safety case” as a type of engineering artifact (i.e., “hazard” and “evidence”) were added as well. Note that these extensions are mere examples—other extensions are possible, depending on the needs of the development project at hand. Table 11 shows possible notational elements for the newly introduced extensions.

It must be noted that there is a close relationship that exists between the safety case (or rather: evidence and hazards contained therein) as well as the context subject’s structure, functionality, and effect on the context. Specifically, the context subject’s functions give rise to hazard-inducing requirements, which are then rectified with newly defined hazard-mitigating requirements. In consequence, safety impacts the operational context as well. Yet, there is no direct, ontologically new information that require special treatment in the context models. Instead, these are documented in engineering artifacts, as we will see in Section 6.2.3. In the following section, we first take a look at context information in safety engineering artifacts.

6.2.2 | Context information in safety assessment artifacts

As we have outlined above, ascertaining the safety of a system is a matter of ascertaining the impact of the system on the context. In case of CPS, this problem becomes twofold, as the safety of a system may also be dependent on the impact of the context on the system itself. For example, when the networked convoy requests the ACC to increase speed above 80 km/h, this request may not be safe in circumstance where the vehicle is towing a trailer. In this case, the safety of the car also depends on interactions from the context, not just the other way around. In consequence, safety analyses such as FMEA naturally contain a considerable amount of context information.

In the following Table 13, we show an FMEA of a personal assistant robot “Care-O-Bot 3.” This analysis was conducted by a safety expert based on a freely available requirements specification of the robot focusing on the “move” and “perceive” functions. Albeit the requirements specification was not developed with cyber-physicality in mind, the robot participates in a cyber-physical infrastructure, collaborating with smart
home health devices, roaming the physical environment of the home of an elderly person, and of course interacting with the person. The FMEA was part of an experiment reported in Madala et al.\textsuperscript{87} and therefore did not specifically focus on the identification of context information. Nevertheless, it contains a considerable amount of context information, relative to the robot’s subsystem (i.e., “move” or “perceive” as denoted in the column “Op Mode”). In Table 13, we highlighted the context objects as follows:

- Knowledge sources (from the context of knowledge) are depicted using text \textit{in italics}.
- Context entities (from the structural operational context) are depicted using \textbf{bold text}.
- Context functions (from the functional operational context) are depicted using \underline{underlined bold text}.
- Context states (from the behavioral operational context) are depicted using \textbf{bold text \textit{in italics}}.

Table 12 summarizes the findings from Table 13, joining synonyms into one cell. In total, we found 31 disjoint context objects in this FMEA. In Madala et al.\textsuperscript{87} we reported on a total of five FMEA analyses, each of which considered context objects to some degree, averaging of ca. 20 context objects per analysis.
**TABLE 13** Failure mode and effects analysis of the Care-o-Bot99 prepared for experimental results reported in Madala et al.87

<table>
<thead>
<tr>
<th>ID</th>
<th>Function</th>
<th>Op. Mode</th>
<th>System nomenclature</th>
<th>Failure modes and causes</th>
<th>Severity class</th>
<th>Potential causes</th>
<th>Safety goals</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>PER_F1 Recognition of Known Objects</td>
<td>Perceive</td>
<td>Camera</td>
<td>Camera fails</td>
<td>Catastrophic</td>
<td>1. Hardware fatigue</td>
<td>1. Surround cables with protection</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2. Cable chafed</td>
<td>2. Add redundant camera system</td>
</tr>
<tr>
<td>2</td>
<td>PER_F1 Recognition of Known Objects</td>
<td>Perceive</td>
<td>Object recognition</td>
<td>Recognition too late</td>
<td>Critical</td>
<td>1. Dust on camera</td>
<td>1. Request RO assistance in limiting recognition</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2. Poor lighting conditions</td>
<td>2. Increase maintenance intervals for dust removal</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>3. Object in atypical configuration (e.g., tipped over)</td>
<td>3. Increase object library</td>
</tr>
<tr>
<td>3</td>
<td>PER_F1 Recognition of Known Objects</td>
<td>Perceive</td>
<td>Object recognition</td>
<td>Fails to recognize</td>
<td>Critical</td>
<td>1. Dust on camera</td>
<td>1. Request RO assistance in limiting recognition</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2. Poor lighting conditions</td>
<td>2. Increase maintenance intervals for dust removal</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>3. Object in atypical configuration (e.g., tipped over)</td>
<td>3. Increase object library</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>4. Object library unavailable</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>PER_F2 Sensor fusion</td>
<td>Perceive</td>
<td>Sensor fusion</td>
<td>Sensor fails</td>
<td>Critical</td>
<td>1. Hardware fatigue</td>
<td>1. Surround cables with protection</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2. Cable chafing</td>
<td>2. Add redundant camera system</td>
</tr>
<tr>
<td>5</td>
<td>PER_F3 Env Perception for nav and manipulation</td>
<td>Perceive</td>
<td>Environment learning</td>
<td>Map inaccurate</td>
<td>Catastrophic</td>
<td>1. Sensor fusion fails</td>
<td>1. Revert to default mode</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2. Camera fails</td>
<td>2. Revert to limited functionality mode</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>3. Sensor fails</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>4. Object recognition inaccurate or too late</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>PER_F4 Learning new objects</td>
<td>Perceive</td>
<td>Object learning</td>
<td>Learning fails</td>
<td>Negligible</td>
<td>1. Camera fails</td>
<td>1. Retry learning upon next attempt</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2. Object recognition fails or too late</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>PER_F5 Human Motion Analysis</td>
<td>Perceive</td>
<td>Human motion analysis</td>
<td>Recognition fails</td>
<td>Critical</td>
<td>1. Camera fails</td>
<td>1. Stop own motion</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2. Object recognition fails or too late</td>
<td>2. Attempt to re-analyze motion</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>3. Request RO assistance</td>
<td>3. Request RO assistance</td>
</tr>
<tr>
<td>8</td>
<td>NAV_MAN1 Navigation/ manipulation by given targets</td>
<td>Move</td>
<td>Motion</td>
<td>Robot fails to move</td>
<td>Critical</td>
<td>1. Motor failure</td>
<td>1. Request RO/LO assistance</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2. Slippery surface conditions</td>
<td>2. Revert to alert mode</td>
</tr>
<tr>
<td>ID</td>
<td>Function</td>
<td>Op. Mode</td>
<td>System nomenclature</td>
<td>Failure modes and causes</td>
<td>Severity class</td>
<td>Potential causes</td>
<td>Safety goals</td>
</tr>
<tr>
<td>----</td>
<td>----------</td>
<td>----------</td>
<td>---------------------</td>
<td>-------------------------</td>
<td>---------------</td>
<td>-----------------</td>
<td>-------------</td>
</tr>
<tr>
<td>10</td>
<td>PER_F4 &amp; NAV_MAN1 &amp; NAV_MAN2</td>
<td>Move</td>
<td>Collision</td>
<td>Robot collides with object in environment</td>
<td>Catastrophic</td>
<td>An object was moved by a human into the robot's path</td>
<td>Monitor path during movement execution, emergency stop</td>
</tr>
<tr>
<td>11</td>
<td>PER_F4 &amp; NAV_MAN1 &amp; NAV_MAN2</td>
<td>Move</td>
<td>Collision</td>
<td>Robot needlessly circumvents object that previously was in its path</td>
<td>Negligible</td>
<td>An object was removed by a human from the robot's path</td>
<td>Monitor path during movement execution, recalculate route</td>
</tr>
<tr>
<td>12</td>
<td>PER_F5 &amp; NAV_MAN1 &amp; NAV_MAN2</td>
<td>Move</td>
<td>Human motion analysis</td>
<td>Robot collides with human in path</td>
<td>Catastrophic</td>
<td>Human steps into robot's path</td>
<td>Monitor path during movement execution, emergency stop</td>
</tr>
<tr>
<td>13</td>
<td>PER_F3 &amp; PER_F5</td>
<td>Perceive</td>
<td>Object handling</td>
<td>Robot drops object in unsafe location</td>
<td>Critical</td>
<td>1. Previously learned target location is obstructed 2. Human placed other object in target location 3. Target location was removed</td>
<td>Re-learn environment</td>
</tr>
<tr>
<td>14</td>
<td>NAV_MAN1 &amp; PER_F1</td>
<td>Perceive/move</td>
<td>Remote operator unavailable</td>
<td>Robot requires ro intervention, but ro is unavailable</td>
<td>Critical</td>
<td>1. Remote connection interrupted 2. RO distracted 3. RO absent</td>
<td>Stop robot behavior and wait for RO to be available again</td>
</tr>
</tbody>
</table>
6.2.3 | Explicit Hazard modeling

In tangential work,\textsuperscript{98,100,101} we have proposed the use of hazard relation diagrams to aid the manual validation of functional safety requirements. The key idea of hazard relation diagrams is to unify safety-critical context information (i.e., information from the hazard analysis results) with the hazard-mitigating requirements of the context subject. In this sense, hazard relation diagrams can be understood as a type of context model that structure the information necessary to conduct validation by visualizing trace concepts between knowledge sources from the context of knowledge (i.e., hazard analysis results) and engineering artifacts of the context subject (i.e., activity diagrams showing hazard-mitigating requirements).

The core modeling concept is hazard relation that associates one hazard to, among other things, its set of trigger conditions and safety goal. The “hazard” directly relates to the ontological extension to context of knowledge diagrams proposed in Table 11, while “safety goal” relates to the “safety case” from the same table. It must be noted that in goal-oriented requirements engineering, safety goals are more likely considered to be engineering artifacts of the context subject; see Pohl.\textsuperscript{2} Nevertheless, including them here makes obvious contextual information about the hazard to aid validation. The “trigger conditions” on the other hand, represent context states in the sense of Figure 4, and represent triggering conditions form the operational context that during operation give rise to the hazardous operational condition (see Tenbergen et al.\textsuperscript{100} for a detailed discussion). In this sense, hazard relation diagrams bridge the gap between engineering artifacts and context artifacts, thereby interrelating context of knowledge information (i.e., hazards), operational context information (i.e., trigger conditions), and engineering artifacts (i.e., safety goal and hazard-mitigating requirements depicted as UML activities).

Figure 16 shows an example of a Hazard Relation Diagram of the Care-o-Bot example from Section 6.2.2. In contrast to our ACC running example, we modeled the Care-o-Bot in this case to show that context information as well as explicit hazard modeling is possible for arbitrary systems using our context modeling approach (rather than an opportunistically selected running example). Nevertheless, the hazard relation diagram for our ACC running example is available.\textsuperscript{100} The diagram in Figure 16 shows an excerpt of the Care-o-Bot’s functional requirements, specifically the Op Mode “move,” depicting the functional requirements NAV\_MAN1 and NAV\_MAN2 from Table 13. Hazards are depicted using the symbol suggested in Table 3; trigger conditions are depicted as context states (see Table 9). The safety goal is depicted as a UML class with the appropriate stereotype, while functional requirements are depicted as UML activities. The hazard relation is an n-ary association depicted as a bold circle, representing the interrelation of the aforementioned engineering artifacts and context artifacts. In this diagram, the failure mode H10 “robot collides with object in environment” is shown as a hazard. This hazard has one context state as the trigger condition (“potential cause” in the FMEA vernacular from Table 13), that is, “object moved into robot’s path.” To implement the safety goal “monitor path during movement execution,
emergency stop," three hazard-mitigating requirements have been added, which monitor obstacles as well as human motion through additional input ports and, if an obstacle is detected, initiate an emergency stop by acting upon both motors of the robot.

During development, hazard relation diagrams can be automatically generated from hazard analyses\textsuperscript{101} such as the FMEA shown above and statistically significantly improve the manual validation of the system's functional, hazard-mitigating requirements.\textsuperscript{100}

7 | CONCLUSION

In this article, we outlined the importance of explicit documentation of context models in the engineering of CPS. We proposed a framework for context modeling consisting of ontologies and notational elements that allow the explicit documentation of different context information in four distinct but related perspectives:

- The context of knowledge subsumes information sources constraining the development of the system under development.
- The structural operational context subsumes relationships between interacting context entities as well as the context entities and the system under development.
- The functional operational context subsumes context functions, which represent conceptual "services" functions that can be accessed by the system under development during functional collaboration at runtime, for example, with other CPS.
- The behavioral operational context subsumes externally observable states of context entities and context functions.

Although our framework is meant to aid the elicitation of explicit and implicit context information, we propose a visual notation based on established diagram types along with tool support. We demonstrated the application of our visual notation by means of a detailed running example and showed in two application examples how context information elicited and documented using our framework yields benefits to CPS development that would otherwise remain covert in traditional engineering artifacts. We also showed how out context ontologies can be extended for engineering activities such as safety assessment.

Our context modeling framework has been successfully applied in several industry collaborations on different case examples, including a salt water desalination plant,\textsuperscript{85,86} an automotive light controller,\textsuperscript{102,103} and an avionic collision avoidance system.\textsuperscript{75} During these collaborations, we subjectively experienced that although some practitioners seamlessly adopt our proposed framework and visual notation, others struggle with the differentiation of information pertaining to context objects as opposed to the system under development. Future work will therefore investigate the cognitive pitfalls and avenues to improve the adoption of our context modeling framework in an effort to provide method guidance to accompany our theoretical framework.

ACKNOWLEDGMENTS

This research was partly funded by the German Federal Ministry of Education and Research under grant number 01IS12005C. We would like to thank our colleagues André Heuer, Philipp Bohn, Jennifer Brings, Andrea Salmon, and Thorsten Weyer for their collaboration in research projects and for influencing our work as well as industry partners from the valuable feedback from practical application that led to iterative improvement of the context modeling framework. Open access funding enabled and organized by Projekt DEAL.

DATA AVAILABILITY STATEMENT

The data that support the findings of this study are openly available in figshare (https://doi.org/10.6084/m9.figshare.14582676).

ORCID

Marian Daun https://orcid.org/0000-0002-9156-9731
Bastian Tenbergen https://orcid.org/0000-0002-0145-4800

ENDNOTE

* The model-based implementation and Enterprise Architect MDG plugin are freely available online (https://doi.org/10.6084/m9.figshare.14582676).

REFERENCES


How to cite this article: Daun M, Tenbergen B. Context modeling for cyber-physical systems. J Softw Evol Proc. 2022:e2451. doi:10.1002/smr.2451