Context Flow Graphs: Situation Modeling for Rule-Based Proactive Context-Aware Systems

SANDRO RODRIGUEZ GARZON, (Member, IEEE), AND BERND LOUIS
Service-centric Networking, Telekom Innovation Laboratories, Technische Universität Berlin, 10587 Berlin, Germany

Corresponding author: Sandro Rodriguez Garzon (sandro.rodriguezgarzon@tu-berlin.de)

This work was supported in part by the German Research Foundation and in part by the Open Access Publication Fund of TU Berlin.

ABSTRACT
A proactive context-aware system automatically adapts its user interface to the user’s situational needs. This is achieved by continuously capturing the environmental properties, reasoning upon the context, and detecting situations where unsolicited adjustments are helpful or notifications informative. If the characteristics of those situations are well known in advance, their occurrence can be detected at runtime by the rule-based processing of raw sensor data. However, rule-based context reasoning methods determine the user’s situation mostly based on present sensor signals instead of considering the situation to be likewise the product of the past context. This article introduces a graph-based situation modeling formalism for the specification of system-relevant environmental circumstances as context flow graphs. A directed cyclic graph represents thereby the distinct contextual characteristics a user’s situation is made of and the temporal order in which these appear and disappear during the evolution of the situation. Complex situations for rule-based proactive context-aware systems can then be expressed at a high level of abstraction and without the need to understand the underlying sensor-related signal processing mechanisms. The technical feasibility is demonstrated by a prototypical distributed proactive context-aware middleware that, in addition, comes up with a web-based user interface for the interactive graphical and logical modeling of situations as context flow graphs.

INDEX TERMS
Context-aware Computing, context modeling, proactive context-aware services, rule-based context reasoning, situation modeling, temporal context reasoning.

I. INTRODUCTION
With his seminal thoughts about a future full of ubiquitous computing devices and services [1], Mark Weiser contributed to the success of an application design paradigm that is today known as context-aware computing [2]. Rather than relying solely on domain-specific input for decision making, context-aware systems (CAS), in addition, pay attention to the multifaceted and dynamically-changing execution context. Well-selected environmental properties such as a mobile device’s geographical position - denoted as the application’s context - that are considered to be relevant for the execution of business functions, are reasoned upon in order to act with respect to the present situation [3].

A reactive context-aware system provides immediate context-dependent results in response to a user-initiated request, e.g. as during the search for a point of interest with a smartphone. If it acts autonomously by delivering context-dependent information or adapting the user interface based on the context, without explicit request by the user, it is referred to as a proactive context-aware system. Since the user’s activity gets interrupted by a proactive context-aware system, it is of paramount importance to sense, understand and interpret the situation in its full complexity so as to act in a non-disturbing and useful manner. The continuous derivation and detection of system-relevant situations on the basis of the interpretation of raw sensor signals is done through context-reasoning [4]. In a proactive context-aware system, this inference process can be set up either by hard-coding it, by allowing a user to customize it, or by letting the system learn the correlations between raw sensor signals and relevant high-level situations based on the usage history. If these correlations are known in advance and if the inference process is universally applicable, it will preferably be hard-coded. Smartphone-based activity detection [5] frameworks are based on this principle because accelerometer, gyroscope, and position sensors show universally recognizable patterns for each activity a person is participating in. But if a proactive
context-aware system has to detect highly individual situations, e.g. with the aim to automate personal tasks, the context reasoning process must be flexible. In a context-aware system that learns from experiences in a (semi)-autonomous way, as applied e.g. for the automatic contextual personalization [6], context reasoning happens implicitly by learning which situations are relevant and how they look like with respect to the raw sensor signals (e.g. through case-based reasoning [7] or machine learning classification [8]). But if the relevant high-level situations and their composition with respect to raw sensor signals are known in advance, e.g. within location-based reminder applications, their occurrence can be detected by a rule-based proactive context-aware system (in the following abbreviated as RPCAS) through the application of predefined context abstraction rules.

The benefits of rule-based context reasoning lie in the simplicity of the rules [9], their efficient evaluation and execution [10], the traceability of high-level situations, the applicability without a bootstrap phase, and the reliability of the inference process in terms of determinism. It is therefore preferentially applied in user-centric application areas such as home automation systems [11] and context-aware mobile applications [12]. In purely rule-based context reasoning, the environmental properties relevant for an RPCAS are well-selected by a human being according to a given domain model. To a certain degree, a human also specifies the high-level situations, how they are inferred from sensor signals, and the application-specific tasks the system should execute in case a situation of relevance occurs. Depending on the rule modeler’s expert level, the complexity of the application’s context and desired level of freedom, the context abstraction rules and their associated actions are defined through the use of sophisticated graphical or textual rule editors [13], with varying simplicity and power. Despite known challenges such as the detection of contradicting rules and the need to formalize the domain at design time [14], rule-based context reasoning became one of the major non-hard-wired context-aware computing techniques for ambient intelligence environments, with a multitude of approaches and applications introduced over the past decades [9].

Although RPCAS’s differ in one way or another regarding the applied rule formalism, their behavior can be generalized by ECA-rules (event, condition, action). During runtime, a rule’s condition is evaluated when the associated event occurs. This may happen - beside other reasons - periodically, triggered by the execution of another rule’s action or a sensor’s signal change. If the condition is satisfied at the moment the event occurs, the action is executed immediately. A situation of relevance can then be represented either by the event, by the condition, or a combination of both. The ECA-rule’s condition is mostly being specified as logic predicates over sensors signals such as within the context-aware middleware Gaia [15], [16] or as the situation abstraction presented in [17] and it is evaluated at a certain point in time. In other words, the high-level state of the environment is inferred by looking onto a snapshot of the sensor signals and by instantly applying abstraction rules. The context is thereby evaluated independently from a previous evaluation at an earlier point in time. Yet, an entity’s situation evolves and follows a process rather than being instantaneous and independent [18]. Out of the CAS’s perspective, an entity’s current state of the context “cannot be understood in isolation from the previous states” [19]. Hence, an RPCAS needs to become stateful in order to detect and react to situations comprising “sequences of contextual states” [20] or resulting from “a finite sequence of actions” [21]. RPCAS’s based on this principle can already be found in various domains and applications from natural language processing [22], over failure management [23] or dangerous situation detection at smart homes [24], to the application of direct marketing with location-based services (LBS) [25].

However, the process to specify complex temporal situations inevitably requires a higher cognitive effort since human modelers are responsible to transform the abstract notion or idea of a real-world situation into a proper computer-readable format. A seamless situation modeling process requires therefore an easy and intuitive modeling formalism to be in place which is expressive enough for the specification of complex situations and at the same time, precise, complete, and sound to prevent inconsistency and ambiguity. Since RPCAS’s are not only customized by technical but also marketing or logistic experts, the modeling formalism needs to allow even users with little technical experience to clearly, precisely, and unambiguously describe situations for the parametrization of RPCAS’s. The envisioned formalism must, in addition, be designed in such a way that its application is practically feasible. Not only with respect to the modeler’s user experience but in particular regarding the number of resources required to detect the occurrence of a complex situation during runtime.

The article introduces a new situation modeling formalism for proactive context-aware systems named SPICE. It’s intended purpose is to give domain experts a flexible and powerful way to encode domain-specific complex environmental circumstances - for the configuration of RPCAS’s - at a high level of abstraction and without the need to understand the underlying detection logic. This approach to situation modeling is based on the fundamental assumption that situations may take time to evolve before they finally occur. A situation is interpreted to manifest itself only in case the environment, monitored by means of sensors, follows a predefined temporal behavior, as formalized with SPICE. The need for SPICE stems from the fact that time is yet not considered to be a first-class citizen of former approaches to situation modeling or is only rudimentary covered. SPICE, on the contrary, describes a situation as temporal sequences of contextual states because it naturally allows to reason upon temporal properties such as the order in which contextual states occur, the duration an environment shows the characteristics of a contextual state, and the time an environment evolves from a contextual state to the subsequent one.
The article starts in Section II with an overview of how to represent situations using time-extended ECA-rules and alternative situation modeling approaches, such as Petri nets and process models. The fundamental concepts of SPICE are introduced in Section III together with the proper formalization and the corresponding execution semantics. The reference implementation of a SPICE-empowered rule-based proactive context-aware middleware is described in Section IV. In Section V, SPICE is compared to related situation and context modeling formalisms. The article concludes with a short summary in Section VI and an overview about possible and planned extensions in Section VI.

II. RELATED WORK
Since the early appearance of RPCAS’s, time is considered an essential element of the context, whether in form of the time of the day, the season of the year [26] or date [27]. Rule formalisms were equipped with the ability to incorporate time constraints, narrowing down a situation of relevance to happen at a specific time or time interval. This can be done by specifying the moment in time an ECA-rule should be evaluated (event specification), by allowing logic predicates to contain assertions over time (condition specification) e.g. “on Monday”, “between 8:00 and 10:00” or “within 2019”, or a combination of both. In addition, temporal aspects of a situation can also be considered on a meta level across different ECA-rules.

Following the meta level approach, Schmidt et al. [28] add the notion of duration to ECA-rules by postponing the execution of an action or by repeating it according to a specifiable period. Similarly, Costa et al. associate lifetimes to ECA-rules [29]. An ECA-rule becomes thereby “active” respectively ready to be evaluated only once, within a lifespan, starting at a point in time, multiple times, or frequently. Qiao et al. simplify ECA-rule editing by introducing a visual editor [30] with temporal specification capabilities. A dashed directed line pointing from an ECA-rules’s action to an event of another ECA-rule states that the latter rule is only evaluated eventually after the former rule fired. This temporal relationship on a meta level is annotated with a concrete duration after which the latter rule is evaluated. Nevertheless, executing ECA-rules, even sequentially in case one rule triggers another, takes no time per definition, and adding artificial delay constructs on a meta level breaks their conciseness and readability. They are better suited to describe the causality between situation and proactive action. So rather than adapting well-approved ECA-rule formalisms to incorporate time constraints for context reasoning on a meta level, researchers began to investigate how to integrate temporal aspects into the event or condition specifications using logic formalisms and languages.

First introduced by Yau et al. within the proposed Context-Aware Interface Definition Language (CA-IDL) [31], [32] and partly integrated into the situation abstraction formalism presented by Henricksen et al. [33], situations became specifiable as the logic predicates [31] - similar to temporal logic [34] - or sequences of situation-change events [33]. Based on the context modeling language (CML), Henricksen et al. define situations by a simplified form of predicate calculus. ECA-rules are then triggered in case of the occurrence of a single or multiple temporally ordered situation-change events. Hinze follows the same principle by allowing to specify a temporal disjunction, conjunction, or sequence between events and the negation of a single event occurrence [35]. Hinze’s complex event specification formalism is later used in a slim variant by Beer et al. to develop a context-aware push notification system [36]. Yau et al. go a step further and allow arbitrary logic predicates to be specified as a trigger condition, including the use of temporal operators to combine predicates temporally, e.g. condition B has to hold within a certain time interval after condition A was satisfied. Although these approaches are promising and able to represent and detect a wide range of situations, the situation descriptions are rather abstract, complex and specifying multiple variations of the same situation leads to either highly overloaded expressions or multiple situation descriptions for each variation [33]. Augusto et al. propose a smart home solution based on active databases to define and detect normal and abnormal situations that are encoded as ECA-rules [37]. For that, they introduce a new ECA-rule specification language that makes use of temporal operators, as formalized by Galton [38], to put instantaneous or durable events at different times into relation to each other in order to specify situations comprising complex event sequences. Later, belief distributions were introduced to specify the magnitudes of uncertainty of contextual information [39]. Tan et al. follow a similar approach and interpret the context for Smart Spaces respectively intelligent environments as complex sequences of events [40]. An event ontology describes how events can be related to each other using logical operators including a sequential order operator. The ECA-rules of the accompanying prototype are then used to describe either the action that needs to be taken in case a complex event occurs or to abstract high-level context.

Although ECA-rules are a convenient way to integrate a situation description and the corresponding action to be taken into a coherent syntax, a situation can also be specified independently of a concrete ECA-rule formalism. Magherini et al. introduce real-time past linear temporal logic (R-pTL), which is based on temporal logic concepts, to specify activities of daily living (ADL) as linear temporal logic formulas [41]. The authors propose a 3-step methodology to translate the natural language representation of human behaviors to R-pTL formulas in order to reduce the complexity and to improve the usability of R-pTL. Coronato et al., instead, use the Situation Calculus [42] to describe ADLs for an ambient assisted living environment with cognitive impaired people [21]. The Situation Calculus is a well established logical formalism to reason about action and change. Coronato et al. describe situations as linear sequences of actions, similar to the approach taken by Henricksen et al.. A sequence of situations may then lead...
to a state of the world in which actions need to be taken proactively, e.g. to raise an alert.

The Event Calculus is a logical language for representing and reasoning upon events and their effects over time [43]. It is well suited to describe contextual phenomena as fluents in event-based systems. A fluent can be interpreted as a global binary state variable that changes its state from “hold” to “not hold” or vice versa as a result of event occurrences. Efstratiou et al. make use of a simplified variant of the Event Calculus [44] to define ECA-like policies that represent adaptation rules for mobile systems [45]. A policy comprises definitions of events that change the status of fluents, a set of fluents, a condition as logical expressions over event occurrences and fluents, and a set of actions that are triggered in case the condition is fulfilled. A similar approach is undertaken by Chen et al. which formalizes complex actions in a smart home environment as sequential and/or parallel declarations of primitive actions [46]. Primitive actions, on the other hand, are Event Calculus axioms that are glued together with extra control structures such as the non-deterministic choice of action ($\alpha|\beta$) or concurrency operator ($\alpha||\beta$). An Event Calculus-based decision support system for smart cities is presented by Hu et al. [47]. Complex urban situations such as a collision, a citizen rescue incident, or traffic conditions are described with the Event Calculus. Rules are used to link urban situation change events with the evaluation of urban conditions and the execution of actions. The main objective of monitoring urban situations is to quantify (action of a rule) the relevance of urban contextual information for citizens or local authorities. Similar to [40], Patkos et al. introduced an event ontology in their event-based context reasoning framework to semantically describe compound events for ECA rules by means of the Event Calculus [48]. Recently, business processes, e.g. representing a manufacturing process, were made context-aware by enabling business process engines to act based on contextual information [49]. Industrial scenarios of relevance are encoded as Event Calculus axioms because they take time to evolve. A stream of events originating from machines is continuously monitored with respect to potential scenarios. If a scenario occurs, in other words, if a logical conjunction of Event Calculus axioms is fulfilled, the attributes of involved entities change, e.g. an alarm is triggered. RTEC, a dialect of the Event Calculus with windowing mechanisms [50] for stream reasoning, is recently used as a situation specification formalism for fleet management [51] or maritime monitoring [52]. Due to the complexity of “axiomitizing” system dynamics with RTEC, Vlassopoulos et al. try to simplify the modeling procedure by introducing the simplified Event Calculus (SimplEC) [53]. With SimplEC, it is possible to specify situations at a higher abstraction level using more natural language-like constructs and it abstracts away technical details.

However, despite attempts to simplify the situation modeling process, e.g. by presenting a 3-step methodology to translate human concepts into R-pTL rules [41] or SimplyEC [53], the use and application of logical languages such as the Situation Calculus, Linear Temporal Logic or Event Calculus to model situations require a high level of abstract reasoning and a deep understanding of logical concepts. Some domain-specific approaches to situation modeling take a different path and put their focus on simplicity and usability rather than on expressiveness and general applicability. At a very high abstraction level, Li et al. [20] introduced a storyboard editor to represent the application’s user interface behavior as a directed graph consisting of user interface state descriptions and condition-guarded transitions between them. Since this method is specifically targeted to be used by developers of LBS, the transitions can only be guarded by location events, such as “Bob is in Library”. Similarly, Welbourne et al. [54] presented a prototype to graphically model a sequence of location-related events with a scene editor in order to generate a computer-readable sequential event pattern for the proper configuration of an event-based situation detection subsystem. Targeting the same application area and level of abstraction, but properly formalized and more powerful, the scenario editor in [25] allows non-technical experts to graphically design temporal location patterns for the configuration of proactive LBS. Given the spatiotemporal pattern and its event-based counterpart, a complex event processing engine is then able to detect the situation of relevance, similar to the aforementioned approach [54]. Trucking mobile objects and their state - with respect to specifiable constraints - using a state transition model is proposed in [55]. Movement plans are thereby described as state transition models and non-normal movement events are detected by continuously monitoring the object-associated constraints. Domain-based approaches are well-suited to specify situations with a well-known and concise set of domain-specific context features such as location and time for LBS, but they fall short if applied to other domains, in which the range of context feature is wider and the constraints arbitrarily complex. With iCAP, Dey et al. try to ease the development of context-aware applications in general by allowing an even non-technical developer to describe situations using a graphical editor [56]. The temporal aspect is considered by being able to specify events that have to occur in a specific order or time interval in order for an action to take place. However, the selection of context features is limited to location, time, activity and personal relationships.

Petri net is a graph-based and domain-independent graphical formalism to model concurrent systems. While they are known to be useful for modeling and verifying system dynamics, including the behavior of RPCAS’s, they can also be applied to reason upon the context. Lee et al. [57] describe high-level context as event sequences, modeled as fuzzy colored Petri nets with time constraints. In [58], situations are represented as a state diagram, whereby context events trigger the transition from one state into another. The state diagrams are manually translated into a synchronized Petri net which - in turn - is used for code generation. Modeling the behavior of a virtual environment using colored Petri nets is proposed by Silva et al. [59]. During the interactive test
of context-aware designs within a virtual 3D environment, the distinct Petri nets for each component of the virtual environment are used to control the simulation at runtime. In [60], highly individual Petri nets of a smart home environment - representing the personal preferences - are used to verify a smart home application at the simulation level. According to the authors and similar to the approach presented in [58], it’s possible to generate code based on the models. But despite being well-studied and its broad tool support, Petri net modeling remains within the domain of technical experts, applied to an RPCAS mainly for verification purposes [60]–[62] rather than situation inference during operation.

Some approaches combine different situation modeling techniques in order to benefit from the advantages of different worlds in one methodology. Liu et al. [63] uses a Markov logic network to describe the logical relation between first-order logic formulas which in turn specify contextual state constraints. Their approach enables uncertain inference of a system’s state by attaching weights to each formula, similar to Fuzzy logic. Wieland et al. [64] started to make use of the context to control the execution of a workflow model within a Smart Factory. The workflow modeling language BPEL\(^1\) was extended in order to explicitly assess the type of context considered at a particular process step and how it influences the control flow. Commonly, context descriptions and domain-specific actions are interwoven into a joint representation of a context-aware technical process. Unfortunately, context conditions at different process steps are evaluated independently, without being able to declare and verify temporal relations between different context manifestations. The integrated virtual operator presented by Realinho et al. [65] eases the development of context-aware mobile applications by allowing the developer to graphically model the context-aware application workflow. Context facts associated with a conditional execution of an activity are specified using logic operators without temporal aspects. In [66], workflow models are used to describe daily user activities rather than context-aware business or technical processes. The SOMNIUM middleware monitors daily user activities with the YAWL\(^2\) or jBPM\(^3\) workflow execution engines. Context reasoning happens outside of the workflow engines, using a rule-based complex event processing (CEP) engine. While rule formalisms for CEP engines became expressive and powerful, including the ability to specify relations between the attributes of successive events, they are highly complex, making it difficult and even infeasible for non-technical users to create and handle complex event patterns. Graphical editors for the definition of complex event patterns in general [67]–[69] try to simplify the process of complex event pattern design. However, similar to the use of the Event Calculus, users must have a deep understanding of how raw events contribute to the evolution of a situation.

### III. CONCEPT

The domain-independent situation modeling formalism SPICE as introduced in this article tries to combine different logical- and graph-based techniques into a fully integrated formalism in order to benefit from the expressiveness and soundness of logical languages and the descriptiveness and simplicity of graph-based approaches. It is intended to be used by human beings to formally express environmental circumstances that are either not representable by or too complex and cumbersome specifyable with the dynamic context modeling techniques discussed in the previous section.

SPICE models are meant to be used in combination with rule-based context-aware application logic to adapt the behavior of an RPCAS proactively once a modeled situation occurs. Figure 1 gives a high-level conceptual blueprint of a SPICE-empowered RPCAS and a yet simplified but concrete example of an air conditioning system. Domain- and use case-specific situation models formalized with SPICE are assumed to be provided by an application as input to the SPICE-empowered context reasoning component. The latter processes a continuous stream of sensor events and notifies the application logic once a situation is detected. The application’s reaction to the occurrence of a situation can be encoded as application-specific rules, such as the one illustrated within the climate control component. The context reasoning component is the core element of a SPICE-empowered RPCAS because it interprets the situation models, senses the environment via sensors, detects real-world incarnations of the encoded situations, and reports findings proactively to the application logic. It can serve multiple applications in the form of an application-independent context-aware middleware or as an integral part of an application. SPICE is a generally applicable situation modeling formalism and it does not prescribe the way an RPCAS’s architecture has to be designed nor the domain it is applied to. Examples of different situation models for the domains of LBS, intelligent transportation systems (ITS), and Smart Factories are given throughout this article to illustrate the flexibility and wide applicability of SPICE.

This section introduces the situation modeling formalism by means of examples and describes how the context reasoning component needs to interpret a situation model and

---

\(^1\) Business Process Execution Language  
\(^2\) Yet Another Workflow Language  
\(^3\) Java-based Business Process Management
to monitor the stream of sensor events with respect to the encoded situation. Prior to the introduction of SPICE, clear definitions of the terms context and situation will be given so as to have a notion of what the context constitutes and what a situation is generally made of with respect to SPICE (Subsection III-A). This is followed by an introduction of the situation representation in form of situation models (Subsection III-B) and a description of its formalization and execution semantics (Subsection III-C).

A. CONTEXT AND SITUATION

The meanings of the terms context and situation highly depend upon the research area. Nevertheless, a widely accepted definition of context in context-aware computing research, on which this work is built upon as well, is given by Dey et al. [70], stating that “Context is any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves.”. In other words, each environmental property that is relevant for an RPCAS, is conceptually considered to be context. For example, LBSs consider the location of mobile devices as relevant context. Each application-relevant environmental property is denoted as a contextual feature of the context. Its value type can be either of nominal, ordinal, or numerical nature. The current state of the context is given by a concrete snapshot-like vector of signal values \( \vec{x} = (v_1, \ldots, v_n) \) with contextual features 1..n. Since the value range of a contextual feature can be large or even infinite, e.g. in the case of the numerical feature time, so can be the total set of possible states of the context, here denoted as space \( X \).

We consider an RPCAS to execute a context-aware application function only in case the environment starts or ceases to exhibit the characteristics of a known contextual state. For example, automatic windscreen wipers start to operate (application function) as soon as the environment enters the contextual state for “raining”. The application-specific set of unique and distinguishable contextual states that might trigger context-aware actions is given by \( S = \{s_1, \ldots, s_m\} \), where \( s_i = (\vec{x} \in X | E_i(\vec{x})) \) is given as a subset of \( X \) with all elements satisfying the characteristic function \( E : X \rightarrow \{true, false\} \) of the contextual state. This function determines whether a given \( \vec{x} \) is a valid representative of the corresponding contextual state. Even though the values of \( \vec{x} \) change over time, a contextual state keeps being active as long as the current \( \vec{x} \) is a known representative of the contextual state. Given the example above, automatic windscreen wipers will keep operating as long as it’s raining. As a result, an environment remains within a contextual state for a dwell time \( d \neq 0 \), with a minimum \( d = 1/f \), given by the sampling frequency \( f \) of the sensors.

A proactive LBS considers the environment to be within a known contextual state if a mobile target is located within a particular region. Thereby, the characteristic function tests \( \vec{x} \), e.g. \( \text{latitude} : 52.51, \text{longitude} : 13.32 \), against a geofence. A geofence is a specific form to describe the set of all locations that fall into a particular region by the definition of a virtual fence around a geographical region. Analogously, we introduce the more general term contextfence for the declaration of virtual fences around the signal value ranges of arbitrary types of contextual features, be it e.g. enumerations for nominal or intervals for numerical features. A contextfence is hereby defined to be a description of the contextual state’s environmental characteristics by means of a characteristic function. While a geofence specifies only a single virtual fence, a contextfence is multidimensional and stands for an arbitrary amount of virtual fences for potentially different contextual features. For example, a contextfence might describe a contextual state in which the temperature is measured to be above 20°C and the local time is between 9:45 and 10:00.

The term situation is commonly used as a synonym for a contextual state [71], although a situation - as used in common speech - is a consequence of not just the current state of the context but also of previous states. For example, the moment a customer enters an automobile store becomes a situation of relevance for the car dealer in case the customer comes directly from a longer visit of a competitor’s store without actually having purchased anything yet. The relevance stems from the fact that this type of customer can be targeted with a special discount. But if the car dealer just takes notice of the current whereabouts of the customer, without considering historical information, then the complex character of the situation remains hidden and each customer entering the store is treated similarly. A more differentiated picture of the state of affairs is only possible by taking the former locations of the customer, the length of stay at the competitor’s store and the fact into consideration that the customer did not make the competitor happy. A situation, regardless of being relevant for an RPCAS or not, might therefore not only be characterized through a single contextual state, but by the successive occurrence of distinguishable contextual states, with each contributing to the evolution of a common situation.

These definitions of context and contextual state have their roots in the theory of context spaces [72]. Contextual states are thereby represented by \( n \)-dimensional subspaces (representable as geometrical objects) within an \( n \)-dimensional application space where \( n \) corresponds to the amount of contextual features addressed within \( \vec{x} \). The counterpart of the application space within our approach is given by the space \( X \). A concrete state of the context \( \vec{x} \) is represented by a point in the application space. A particular behavior of a environment can therefore be described by a discrete trajectory of successive context manifestations through the application space. A \( n \)-dimensional subspace, representing the eligible value ranges of the contextual state’s context features, are denoted in the theory of context spaces as a situation space [71]. This alternative naming for the notion of a contextual state is in line with our definition since a situation can also just be characterized by a single contextual state. However, we extend the notion of a situation beyond...
denoting only a single contextual state and prefer to use the term contextual subspace instead of situation space for an $n$-dimensional subspace of the application space. With SPICE, a situation respectively a particular behavior of the environment is represented by a set of possible discrete trajectories (sequences of context manifestations) through one or more clearly defined contextual subspaces of the application space. In other words, if the behavior of the environment given by a sequence of states of the context $x_{t-m}, \ldots, x_{t-1}, x_t$ at the moment $t$ matches one of the situation-specific discrete trajectories through the application space, the situation is declared to have been occurred. The purpose of SPICE is to make it possible to clearly and unambiguously specify a set of discrete trajectories and contextual subspaces (by means of contextfences) that commonly represent a particular situation of relevance for a SPICE-empowered RPCAS.

For example, Figure 2 borrows the metaphor of representing contextual subspaces as geometrical objects in the multidimensional Euclidean space from [72] in order to visualize the trajectory of a parcel’s context within the domain of logistics. One contextual subspace specifies the location of a parcel to be the hub, the time to be between 00:00 and 07:00, and the temperature to be between 3°C to 7°C. The other spans a subspace with the delivery base as the location, 08:00-15:00 and 3°C-7°C as permitted contextual ranges for the time and temperature. In logistics, both contextual subspaces may represent parts of delivery requirements for a heat-sensitive parcel. With the help of SPICE, a logistics provider should be able to clearly specify how a parcel needs to traverse the contextual subspaces so that service level agreements are met. A logistics system with a SPICE-empowered RPCAS is then capable to monitor the delivery with respect to the complex situational constraints.

B. SITUATION MODEL

A basic situation model in SPICE comprises a single contextfence $c$, regardless of the contextual features addressed therein. The represented situation can occur and disappear at any time. To materialize itself, it requires only that the signals of the addressed contextual features in $\vec{x}$ conform to the value ranges as defined by the contextfence $c$. For example, if the car dealer of the aforementioned scenario wants to be notified by an RPCAS as soon as any customer enters the store, regardless of the customer’s movement history, the situation is represented by a single contextfence. It contains a computer-readable form of the automobile store’s geographical extent. The application-specific context feature that is being monitored by the RPCAS is then the geographical position of a customer. The situation occurs in case the current geographical position of the customer lies within the automobile store’s extent while the previous was not. Most state-of-the-art RPCAS’s support these basic types of situations models in order to detect the appearance and disappearance of independent contextual states.

A composed situation model comprises at least two related contextfences $[c_1, c_2]$. The hidden contextual state represented by $c_1$ denotes a presituation while $c_2$ stands for the noticeable contextual state of the situation model. The presituation in form of the hidden contextual state is required to happen before in order for the situation to finally materialize itself through the appearance or disappearance of the noticeable contextual state. This type of dependency between two contextfences $c_1$ and $c_2$ is specified by a followed-by relation, meaning that the situation can only occur if and only if the presituation happened somewhere in the past and already vanished before the noticeable contextual state appears and/or disappears. If the presituation never happens, the situation can’t occur even if at some point in time the environment exhibits all the characteristics of the noticeable contextual state. Consider $c_1$ represents the hidden contextual state of being at the competitor’s store (the presituation) and $c_2$ represents the noticeable contextual state of being at the car dealer’s automobile store. The situation will only occur if a customer entering the automobile store visited the competitor’s store in the past. The contextfence for the presituation can be seen as an internal precondition, hidden to the application logic. Unless otherwise requested, the application logic does not need to be aware of the presituation’s occurrence as long as the context reasoning logic takes care of it. On the other hand, the application logic will take notice of the appearance and/or disappearance of noticeable contextual states in order to trigger the execution of situation-aware application functions.

Over time, the environment passes through a variety of distinct states until it reaches the moment in time a situation unfolds in its entirety. Usually, there exists more than one way an environment can evolve till the situation becomes reality. For example, a customer might not directly arrive at the automobile store from the competitor’s store but via short stops at a toy store or playground. If the situation model for the car dealer scenario only comprises $c_1$ and $c_2$ as described above, then anything is allowed to happen in between these two contextual states according to SPICE. The represented situation would even occur if the customer decides to revisit the competitor’s store before arriving at the automobile store. Being allowed to omit specify the intermediate behavior of the environment is an inherent property of SPICE.
An RPCAS equipped with this situation model helps a car dealer to tailor a car offer to the customer’s needs by bundling a child seat coupon with the offer.

The time an environment remains within or in between two contextual states might also be a relevant property of a situation. For example, a customer qualifies for a discount only in case of staying at the competitor’s store for at least 20 minutes - ergo not being a passerby - and sought to visit the toy store within 5 to 10 minutes after leaving the competitor’s store and entering the automobile store 15 to 130 minutes after leaving the toy store. According to these constraints, only customers visiting the competitor’s store and the mandatory intermediate stops on the same day as entering the automobile store will be considered by the car dealer. Within a CFG, contextfences and their followed-by relations can be optionally associated with duration constraints. The timing requirements are then translated into time constraints at the contextfence describing the visit at the competitor’s store (more than 20 minutes) and in between the latter and its counterpart representing the toy store visit (minimum 5 and maximum 10 minutes) and finally in between the latter and the contextfence representing the visit at the automobile store. Figure 3b visualizes a set of exemplary paths a customer might take to reach the automobile store. Each of these paths leads the system to detect the occurrence of the specified situation.

C. SITUATION FORMALIZATION

In the following, the concepts of contextfences and CFGs are thoroughly formalized. Thereafter, it is presented how a CFG has to be interpreted by a SPICE-aware context reasoning logic and how the environment needs to be monitored with respect to a particular situation model during runtime.

1) CONTEXTFENCE

A contextfence’s main purpose is to specify the contextual subspace of a contextual state by the concretization of the characteristic function \( \mathbb{1} : X \rightarrow \{true, false\} \). In addition, it defines whether the contextfence belongs to a hidden or noticeable contextual state of the situation. Duration constraints can be optionally associated with the contextfence to reason upon the dwell time of a contextual state. This leads a contextfence to be given by the 4-tuple

\[
c = < h \in H, \phi, \pi, d_s \in D_s >
\]

with

\[
H = \{(\eta_{enter}, \eta_{leave})|\eta_{enter}, \eta_{leave} \in T\}
\]

\[
\phi = (\lambda_1, \ldots, \lambda_n) \text{ with } n \in \mathbb{N}^{>0}
\]

\[
\pi = \gamma(\phi)
\]

\[
D_s = \{(t_{min}, t_{max})|t_{min}, t_{max} \in \mathbb{R}^{>0}\}
\]

where \( h \) deals with the type of contextual state, \( \phi \) with independent state-specific constraints over contextual features, \( \pi \) with state-specific constraints over all \( \phi \) and with \( d_s \) as the state duration constraints.
In a SPICE-empowered RPCAS, situation occurrences are announced by the context reasoning logic through the emission of proactive state-appearance and state-disappearance notifications. These notifications are only emitted in case a noticeable contextual state is entered or left. The “Automobile Store” contextfence shown in Figure 3a specifies a noticeable contextual state, as marked by an arrow-symbol. An arrow pointing to the right indicates that a state-appearance notification will be emitted by a context reasoning component while entering the noticeable contextual state. An arrow pointing to the left indicates that a state-disappearance notification will be emitted while leaving the noticeable contextual state. The components of \( h \) stand for the notifications out of the text set \( T \) that will be announced in case the environment enters \( (\eta_{enter}) \) or leaves \( (\eta_{leave}) \) the noticeable contextual state. A contextfence with at least one notification attached to it represents a noticeable contextual state. If notifications are omitted, then the contextfence specifies a hidden contextual state and its (dis-)appearance will only be taken notice of by the context reasoning logic. The notification \( \eta_{enter} \) attached to the “Automobile Store” contextfence might contain, for example, the text “Well-informed family-type customer entered the store” so that the car dealer is able to react properly.

The time the environment is permitted to show characteristics of a given contextual state so that the context reasoning logic still considers the environment to be within the this contextual state is given by the duration interval \( [t_{min}, t_{max}] \), respectively \( (t_{min}, t_{max}) \in D_s \). \( t_{min} \) defines the minimum and \( t_{max} \) the maximum duration allowed. The optional duration constraints of contextfences must fulfill the following condition

\[
\forall (t_{min}, t_{max}) \in D_s \mid t_{min} < t_{max}
\]  

because a contextual state is defined to last for a dwell time \( d_s \neq 0 \). If a contextfence has no duration constraints attached to it, then the contextual state can last for an arbitrary duration \( d_s \in \mathbb{R}_{>0} \). The consequence of violating the duration constraints (in case the environment stays in the contextual state shorter or longer than specified) is further explained in Subsection III-C3.

According to Crowley et al., a state in context-aware computing “is defined using a predicate expression” [73]. Our approach follows a similar path by specifying a contextual state by means of multiple simplified first-order logic expressions \( \lambda_1, \ldots, \lambda_n \) which are finally combined into a common expression \( \gamma(\phi) \) using the logical connectives of propositional logic. Each \( \lambda \)-predicate expresses conditions with respect to one or more contextual features, e.g. like position or weather. At a certain point in time, these conditions are either fulfilled or not fulfilled. An exemplary \( \lambda \)-predicate for the contextual state “Automobile Store” within the CFG of Figure 3 might express the fact that the value of the contextual feature “geographical position of the customer” lies within the limits of the automobile store. More generally, a \( \lambda \)-predicate is formalized as follows:

\[
\lambda = ((\land_{j=k}^{l}(\exists w \in W | \land_{j=1}^{l} p_j(w))), \alpha)
\]

with

\[
\alpha \in T \land n, \ m \in \mathbb{N}_{>0} \land n \leq m \land k, l \in \mathbb{N}_{>0} \land k \leq l \land k \leq |S| \leq l \land r \in \mathbb{N}_{>0}
\]

It states that all predicates \( p \) must hold for each element of an arbitrary subset of all sensors \( W \). The cardinality of the subset is limited by the freely-selectable interval \([k, l]\). A \( \lambda \)-predicate is therefore satisfied if all \( p \) hold for at least \( k \) and at most \( l \) sensors out of the total observable sensor set \( W \). Hence, it is possible to state that only a certain amount of all observable sensors \( W \) must meet some predefined criteria in order for the \( \lambda \)-predicate to be satisfied. In this case, it is not known until runtime, which concrete sensors will contribute to the fulfillment of the \( \lambda \)-predicate. For example, a \( \lambda \)-predicate can specify the contextual state that 15 (given by \( k \)) to 30 (given by \( l \)) arbitrary customers out of the total set of 1000 customers (given by \( W \)) are currently located within the automobile store (as specified by \( p \)). A SPICE-empowered RPCAS as part of a loyalty app for customers might trigger the car dealer to run a promotion show which is not worth or possible carrying out if too few or too many visitors are present. The exact subset of visitors that have the chance to participate in the show will be determined during runtime.

But how to specify the observable sensor set \( W \) with SPICE? To make the sensors known to a SPICE-empowered RPCAS, a modeler has the possibility to choose among two different ways to do so. The set can be explicitly predefined by the declaration of all sensors within the meta data section of the model. A situation model does therefore not only consist of a CFG but also of sensor-related meta data. If the set is specified explicitly, the selection of sensors referenced within the \( \lambda \)-predicate is known in advance, at the time the situation model is created. A predefined sample set of position sensors might comprise all GPS-equipped smartphones of a particular customer group that the car dealer considers to monitor.

Instead of defining the set explicitly within the meta data section, it is also possible to refer to a set of sensors for which a \( \lambda \)-predicate in a former contextfence was satisfied while the associated contextual state was active. For example, assume the car dealer to be interested in providing special offers to groups of customers rather than single customers, e.g. couples or families. But how does the car dealer specify the contextual state that two customers are located within the automobile store which previously visited the competitor’s store together? SPICE allows the car dealer to let the \( \lambda \)-predicates of the contextfences “Playground”, “Toy Store” and “Automobile Store” refer to a group of customers which will be selected at runtime, while the contextual state “Competitor’s Store” is active. For a customer to become a group member at

---

4The application logic consumes situation appearance and disappearance notifications to trigger context-aware application functions.
runtime, it is required that the customer stays within the competitor’s store somewhere at the time the state “Competitor’s Store” is active, together, with all the other group members. So if the contextual state is active for at least 20 minutes, then all customers that have been visitors at that time period— independent of the visiting duration - are considered to be members of a common group. The context reasoning logic is still able to reason upon this spontaneous set of sensors even after the state “Competitor’s Store” is already left. This is done by referencing it in a λ-predicate within a contextfence of a subsequent contextual state. Formally, the set of sensors for which all predicates $p$ hold at least at some point in time while the contextual state is active can be referenced in subsequent contextfences by the arbitrary set name $\alpha$. Each λ-expression of a contextfence defines its own unique $\alpha$ set name. For example, assuming the store owner to target a group size of 2 to 4 customers, then the λ-expressions of the state “Competitor’s Store”, “Playground” and “Automobile Store” would have the following form:

$$\lambda_{\text{competitor}} = ((\land_{i=2}^{4}(\exists w \in \text{Customers} | \text{inCompetitorStore}(w))), \text{"StartGroup"})$$

$$\lambda_{\text{toy}} = ((\land_{i=2}^{4}(\exists w \in \text{StartGroup} | \text{inToyStore}(w))), \text{"GroupAtToyStore"})$$

$$\lambda_{\text{automobile}} = ((\land_{i=2}^{4}(\exists w \in \text{GroupAtToyStore} | \text{inAutomobileStore}(w))), \emptyset)$$

(4)

with Customers representing the set of position sensors of all the customer smartphones. The Customers set needs to be defined explicitly within the meta data section and referenced by the “Competitor’s Store” contextfence while the StartGroup and GroupAtToyStore sets are determined during runtime. Hence, it is possible to express and detect situations in which a predefined amount of sensors expose similar behavior over a period of time, across multiple contextual states. This feature is a powerful way to temporally reason upon groups of sensors arising and breaking up dynamically at runtime and which are unknown at the time the situation model was designed. While former RPCAs were mainly focused on monitoring highly individual situations on a micro level covering a manageable amount of sensors, a SPICE-empowered RPCAS targets to be able to operate as well on a macro level, dealing with situation models that may encode and enforce the observation of several thousands of sensors. This becomes especially necessary with the advent of context-aware computing capabilities within IoT platforms [74], e.g. for the detection of situations that involve city-wide sensors installations within Smart City scenarios.

Figure 4 illustrates an exemplary evolution of an RPCAS’s state with respect to the movements of 4 potential customers. In this exemplary case, only 4 different selected customers are defined explicitly within the meta data section of the situation model and are referred to within $\lambda_{\text{competitor}}$ as Customers $\{c_1, c_2, c_3, c_4\}$. The contextual state “Competitor’s Store” becomes active as soon as at least 2 of these potential customers are located within the competitor’s store at the same time, e.g. $c_2$ and $c_3$ at $t_1$. Since the minimum duration for the contextual state “Competitor’s Store” is set to 20 minutes, at least 2 potential customers (specified by $i = 2$) need to be located within the competitor’s store at any point in time during the 20 minute period, starting at $t_1$. The contextual state is left as soon as less than 2 potential customers are present in the competitor’s store, e.g. at $t_3$. At this moment in time, $c_1$, $c_2$ and $c_3$ are considered to be a spontaneous group of customers, referable as StartGroup in $\lambda_{\text{competitor}}$. At $t_4$, the contextual state “Toy Store” becomes active with at least 2 potential customers visiting the toy store (specified by $i = 2$). But $\lambda_{\text{toy}}$ does not refer to an explicitly defined set of customers (like in “Competitor’ Store”), but instead, it refers to the set of customers which has been determined as a spontaneous group, denoted as StartGroup, while state “Competitor’s Store” was active. Hence, the $\lambda_{\text{toy}}$ can only be evaluated as being fulfilled if at least 2 of the group members $\{c_1, c_2, c_3\}$ visit the toy store at the same time. Since only 2 members of the group are present during the contextual state “Toy Store”, the spontaneous group shrinks to the members $c_2$ and $c_3$, forming a subgroup with name GroupAtToyStore in $\lambda_{\text{toy}}$. This subgroup can then be referred to within $\lambda_{\text{automobile}}$ to target only groups of customers which visited the competitor and toy store together in a clearly defined temporal order. To sum it up, two or a maximum of 4 potential customers (specified by $i = 2$ to 4) are identified to be a group of relevance for the car dealer in case they follow the same spatiotemporal movement pattern.

If all λ-expressions for a contextfence are defined (see (3)), they are composed into a single predicate $\gamma((\lambda_1, \ldots, \lambda_n))$ by means of the logical connectives conjunction $\land$, disjunction $\lor$ and negation $\neg$. Parenthesis can be used in addition to fix the order of evaluation. For example, assuming the car dealer to only qualify group members for a discount if they enter the automobile store not during a sales week and - in addition - if the amount of issued discounts does not exceed a predefined maximum. Under these conditions the characteristic function $\gamma(\phi)$ of contextfence “Automobile Store” will have the
following structure:
\[
\gamma_{\text{automobile}} = \left(\lambda_{\text{automobile}} \land \lambda_{\text{discounts}} \land \neg(\lambda_{\text{sales}})\right)
\]
with
\[
\lambda_{\text{automobile}} : \text{customer group visits automobile store}
\]
\[
\lambda_{\text{discounts}} : \text{discounts still available}
\]
\[
\lambda_{\text{sales}} : \text{sales week ongoing}
\]
(5)

with each \(\lambda\)-expression abbreviated for the sake of simplicity. If the combined predicate \(\gamma_{\text{automobile}}\) is satisfied, then the environment is considered to be within the associated contextual state if and only if the associated contextual state is feasible. A contextual state is feasible if and only if all the contextual states prior to it appeared and disappeared in the specified order as defined within the CFG and before the respective contextual state gets active.

2) CONTEXT FLOW GRAPH

If all contextfences are fully-specified, then a CFG defines how they are glued together by means of followed-by relations and optional time constraints. A CFG specifies thereby the temporal order of contextual states a situation is made of and the durations between the distinct contextual states. It is defined by the 5-tuple
\[
< I, C, \Psi, E, \Phi >
\]
with
\[
I \subseteq C
\]
\[
C = \{c_1, \ldots, c_n\}
\]
\[
\Psi : C \times C
\]
\[
D_I = \{(t_{\text{min}}, t_{\text{max}})|t_{\text{min}}, t_{\text{max}} \in \mathbb{R}_{\geq 0}\}
\]
\[
\Phi : \Psi \times D_I
\]
(6)

\(I\) represents the set of initial contextfences. \(C\) is the set of all contextfences (nodes) with \(\Psi\) being the set of all followed-by relations (edges) between contextfences. \(D_I\) is the set of all edge-related duration constraints and \(\Phi\) being the assignment function. A CFG is a directed graph which is allowed to contain cycles. During runtime, only one of the associated contextual states can be active at a time.

The optional duration constraints assigned to edges of the CFG have the same form as their counterpart within contextfences. They allow reasoning upon the time the environment stays in transition between two contextual states. If an edge has no duration constraint assigned to it, the allowed transition dwell time \(d_i\) is implicitly assumed to be arbitrary, \(0 \leq d_i \leq \infty\). In case a duration constraint is defined, it must fulfill the following condition:
\[
\forall(t_{\text{min}}, t_{\text{max}}) \in E \mid t_{\text{min}} \leq t_{\text{max}}
\]
(7)

Since the dwell time can be zero, either specified explicitly in form of a duration constraint \(t_{\text{min}} = 0 \leq t_{\text{max}}\) or implicitly possible by omitting a duration constraint, the environment might instantly change from one contextual state into another without being in transition. The possibility of an instant change of contextual states widens the application range of SPICE but it puts some restrictions on the involved contextfences and their followed-by relation. These will be further discussed at the end of this Subsection.

A CFG can be composed of several contextfences, with some of them representing hidden and some noticeable contextual states. The reasons to assign notifications to more than one contextfence of a CFG can be twofold. If the notification-enabled contextfences are located at the end of different paths through the CFG, then each of them may represent a valid variation of the situation. The application logic can then be notified differently in case each variation is supposed to trigger a distinct application function or can be notified similarly - with the same notification content - in order to hide the variability of the situation from the application logic. In contrast, attaching multiple notifications to contextfences on a common path through the CFG leads the context reasoning logic to emit notifications for each noticeable contextual state the environment passes through during the evolution of the situation. This enables application logic to react to presituations or to simply keep track of the situation-detection progress. It makes the context reasoning logic, if desired, more transparent to the application logic.

Figure 5 shows an exemplary CFG for a SPICE-empowered RPCAS as part of an ITS. The ITS dynamically adapts the speed limit of a particular section of a highway based on the situation occurrence notifications emitted by the context reasoning logic. In the CFG, all contextfences are associated with state-appearance notifications. The ITS is then able to dynamically adapt the speed limit because it gets proactively notified by the context reasoning logic each time a contextual state is entered. The use of a CFG instead of independent contextfences has the advantage that the speed limit can be stepwise increased within a defined period of time after the rain ended.

There are a few additional global constraints that a CFG must fulfill in order to be valid, so to avoid non-deterministic behavior of the context reasoning logic. Consider a function \(o_c : C \times C \rightarrow \{\text{true, false}\}\) that returns true if two
contextfences overlap each other. The function is given by
\begin{align}
o_v(c_i, c_j) &= \begin{cases} 
\text{true} & \exists \bar{x}((\gamma_{c_i}(\phi_{c_i}(\bar{x}))) \land \gamma_{c_j}(\phi_{c_j}(\bar{x}))) \\
\text{false} & \text{else}
\end{cases} (8)
\end{align}
with \(\bar{x}\) being a potential snapshot \((x_1, \ldots, x_n)\) of the environmental properties, ergo representing a possible state of the environment. If there exists a potential state of the environment that fulfills the constraints defined by two contextfences, then they are considered to overlap each other. In addition, consider a function \(o_1 : (\mathbb{R} \geq 0, \mathbb{R} \geq 0) \times (\mathbb{R} \geq 0, \mathbb{R} \geq 0) \rightarrow \{\text{true}, \text{false}\}\) that returns true if two time intervals overlap each other. In order for the CFG to be valid, the following global condition needs to be fulfilled:
\begin{align}
\forall (c_i, c_j, c_k) \in \Psi \ &\ |(c_i = c_j \land c_j \neq c_k) \\
\land \ &o_v(c_i, c_j) \\
\rightarrow \ &\exists (c_1, c_i, (l_{\text{min}}, l_{\text{max}})) \in \Phi, \\
\exists \ &((c_j, c_k), (k_{\text{min}}, k_{\text{max}})) \in \Phi| \\
\land \ &o_1((l_{\text{min}}, l_{\text{max}}), (k_{\text{min}}, k_{\text{max}}))
\end{align} (9)

So if a contextfence has outgoing edges to more than one successive contextfence and if at least two intervals of the edge-related duration constraints overlap each other, all the directly successive contextfences are not allowed to overlap each other. In other words, if two or more contextfences have a common predecessor and if they overlap each other, then the edges coming from the predecessor need to be annotated with non-overlapping duration constraints. Otherwise, the CFG would become non-deterministic because the context reasoning logic might be faced with the ambiguous choice to select a successor state out of two or more possible contextual states. Figure 6 shows a selection of deterministic (valid) and non-deterministic (invalid) sample CFGs of SPICE.

A CFG can also have more than one entry point \(|l| \geq 1\). This increases the conciseness of SPICE because a situation structure might vary at the beginning but converges at a later stage. But if there exists more than one entry point, then the respective contextfences are not allowed to overlap each other because the context reasoning logic might be faced with the same ambiguous choice as mentioned above. This time, at the beginning of the situation’s evolution. Figure 7 shows an exemplary CFG for a SPICE-empowered RPCAS within a mission-critical wireless communication infrastructure for mobile robots in a Smart Factory. It helps to determine whether the communication infrastructure should only restart partially or conduct a full system restart in case a mobile robot goes unexpectedly offline. Although the represented situation has two entry points, distinguishing between a moving and dormant mobile robot, the general workflow for tackling the problem with partial and full restarts of the infrastructure is the same. But since moving robots might experience a longer expected offline phase during a handover to a new communication cell, a partial restart should be scheduled with a longer lead time of 10 seconds than in case the mobile robot is standing still and being offline (with 5 seconds lead time). If the mobile robots remain offline for the lead times, then a partial restart is triggered by the application logic. Otherwise, the “all-clear” notification would trigger the application logic to abort all restart times. If a full restart of the infrastructure did not solve the issue, then a failure of the robot’s radio communication module is assumed.

A contextfence is allowed to be followed-by itself if and only if the followed-by relation has an attached duration constraint with duration \(d \neq 0\). Otherwise, with \(d = 0\), the former contextual state would be left and the latter would be entered immediately after the former occurred. This violates the global constraint that a contextual state has to last a duration of \(d > 0\). The same applies to two contextfences that are connected via a followed-by relation and which are overlapping. Since the former contextual state can appear based on a state of the context \(\bar{x}\) which at the same time could be a valid representation of the latter contextual state.

3) EXECUTION SEMANTICS
The task of a SPICE-aware context reasoning logic is to keep track of the environmental state concerning a predefined situation. During operation, the context reasoning logic considers the environment to be within one of three different situations:
of states: idle, contextual or transition. The contextual state has been described in detail within the former Subsections. If the environment passed through at least one contextual state and is currently not within another one but the context reasoning logic is waiting for the environment to show the characteristics of the next contextual state, following the previous one according to the CFG, then the context reasoning logic is considered to be within a transition state. In other words, the environment is not considered to be within any contextual state but is in the process of “moving” along an edge of the CFG respectively being in a transition phase. If the context reasoning logic considers the environment neither to be within a contextual nor a transition state, then the environment is in the non-situation-related idle state. The context reasoning logic gets into the idle state at initialization time, after leaving a contextual state of a contextfence without outgoing edges or by violating any duration constraints associated with contextfences or followed-by relations. Within the idle state, the environment might - at any time - change into any initial contextual state, represented by the initial contextfences.

RPCASs are mostly built as event-driven architectures. Every proactive-enabled sensor emits its own state in form of an environmental event. Either regularly, in case the state changes, or if a given condition is satisfied. Environmental events are processed by the context reasoning logic in the order received or according to their timestamps in order to have an accurate, valid, and complete picture of the actual state of the context. A SPICE-aware context reasoning logic follows a similar approach and listens continuously to the stream of incoming environmental events, updates the internal state of the context, if necessary, and reevaluates its picture of the environment with respect to the CFG. The conceptual execution behavior of the SPICE-aware context reasoning logic during the monitoring of a single situation is given by a meta model in form of a deterministic finite automaton (DFA) in Figure 8. If the state of a contextual feature has changed according to an environmental event, so will the common state of the context as represented by \( \vec{x} \), the snapshot \((x_1, \ldots, x_n)\) of the values of all context features. A context change, no matter how small or large, leads the context reasoning logic to evaluate a progress function respectively transition function to determine what to do next, either to remain within the current or to change into another state.

To formalize the execution semantics of the progress function, it needs to be first investigated how the CFG translates into a situation-specific DFA (sDFA), representing the situation-specific context processing behavior for the context reasoning logic. While the contextual states are defined explicitly through the specification of contextfences, the set of transition states and the idle state are given implicitly. The idle state exists exactly once in each sDFA representation of a CFG. In contrast, the set of transition states is inferred from the followed-by relations. Whether there exists a transition state between successive contextual states depends solely upon the existence and content of associated duration constraints. If a contextual state should appear immediately after another contextual state, without losing any time in transition, then there exists no transition state in between. This can be enforced by attaching a duration constraint to the followed-by relation between contextfences and restricting it to have no duration, with \( d_t = t_{\text{min}} = 0, t_{\text{max}} = 0 \). If the duration constraint specifies a duration \( d_t > 0 \), then there must exist a transition state in between. In case there are no duration constraints attached to the followed-by relation or if the duration constraints have the form \([0, y]\) with \( y \in \mathbb{R}^{>0} \), then the followed-by relation will be represented within the sDFA by a transition state as well as a direct transition between both contextual states. The direct transition represents the possible instant change into the next contextual state with \( d = 0 \) while the transition duration \( d \neq 0 \) is represented by the transition state. A contextual state is at maximum followed by one transition state. If a contextfence has more than one outgoing edge with potential transition durations \( d_1, \ldots, d_n \neq 0 \), then these edges share the same transition state within the sDFA. The sDFA of the CFG from the car dealer scenario as presented in Figure 3 is exemplarily shown in Figure 9.

Each CFG can be automatically translated into its corresponding sDFA. What follows is a thorough specification of the transition execution semantics of a SPICE-aware context reasoning logic for each state type of a sDFA. Hereby, \( s_i \) is considered to represent the idle state and \( S_j \), the set of all contextual states, with each being specified by a contextfence. It is also given the set of all the transitions states \( S_m \) and the
set of all transitions \( T : S_c \cup S_m \cup \{s_i\} \times S_c \cup S_m \cup \{s_i\} \), as automatically inferred from the CFG. Additionally, it is given a function \( \alpha : S_c \times X \rightarrow \{true, false\} \) that tests if the state-specific constraints are satisfied by a given state of the context \( x = (x_1, \ldots, x_n) \in X \). If the context reasoning logic is within the idle state, then the progress function \( f_{idle} : X \rightarrow \{s_i\} \cup S_c \) is defined as follows:

\[
f_{idle}(\bar{x}) = \begin{cases} s_c \text{ if } \exists s_c \in S_c|\alpha(s_c, \bar{x}) \land (s_i, s_c) \in T \\ s_i \text{ else} \end{cases}
\]

In addition, given the functions \( \theta_c : S_c \rightarrow \{true, false\} \) and \( \theta_m : S_m \rightarrow \{true, false\} \) that test if the elapsed time since entering a contextual state (\( \theta_c \)) and a transition state (\( \theta_m \)) is still within the allowed limits given by the duration constraints of the contextfence respectively by the union of the duration constraints of each outgoing edge. If the context reasoning logic is within a contextual state, then the progress function \( f_{con} : S_c \times X \rightarrow S_c \cup S_m \cup \{s_i\} \) is defined as follows:

\[
f_{con}(s_c, \bar{x}) = \begin{cases} s_c \text{ if } s_c \in S_c \land \theta(s_c) \land \alpha(s_c, \bar{x}) \land \\
\neg \exists s'_c \in S_c|s'_c \neq s_c \land \alpha(s'_c, \bar{x}) \land \\
(s_c, s'_c) \in T \\
 s'_c \text{ if } \theta(s_c) \land \neg \alpha(s_c, \bar{x}) \\
\exists s'_c \in S_c|s'_c \neq s_c \land \\
\alpha(s'_c, \bar{x}) \land (s_c, s'_c) \in T \\
s_m \text{ if } \theta(s_c) \land \neg \alpha(s_c, \bar{x}) \\
\exists s_m \in S_m|(s_c, s_m) \in T \\
s_i \text{ else} \end{cases}
\]

If the context reasoning logic is within a transition state, then the progress function \( f_{tra} : S_m \times X \rightarrow S_c \cup S_m \cup \{s_i\} \) is defined as follows:

\[
f_{tra}(s_m, \bar{x}) = \begin{cases} s_m \text{ if } \theta(s_m) \land \\
\exists s'_c \in S_c|(\alpha(s_c, \bar{x})) \land \\
(s_m, s_c) \in T \\
 s_c \text{ if } \theta(s_m) \land \exists s'_c \in S_c|(\alpha(s_c, \bar{x})) \land \\
(s_m, s_c) \in T \\
s_i \text{ else} \end{cases}
\]

If the environment shows the characteristics of a contextual or transition state shorter or longer than expected according to duration constraints, then the context reasoning logic aborts an ongoing observation of the situation and restarts within the idle state. The same applies in case the environment starts to show characteristics that are not represented by the current nor by immediately following contextfences. If the cancellation happens while being in a noticeable contextual state that is being represented by a contextfence with a state-disappearance notification \( \eta_{leave} \), then the respective notification will not be emitted because the encountered situation did not comply with the situation specification for \( \eta_{leave} \). The direct successor state of contextual states which are represented by contextfences without outgoing edges (last contextfences of potential paths through the CFG) is always the idle state.

IV. IMPLEMENTATION

To investigate the feasibility of CFGs, we have prototypically implemented a SPICE-empowered context-aware middleware, named Sitavio. It continuously analyzes a stream of incoming sensor events with respect to a set of predefined situation models. As soon as it detects the occurrence of a known situation, it proactively notifies all context-aware applications that are subscribed to receive situation-specific notifications. Sitavio encompasses an optional web-based user interface for administration purposes, a situation monitoring engine (SME), and sensor connectors. Sitavio’s architecture is closely related to the design of the situation recognition layer of the SitOPT approach [75]. However, as an RPCAS, SitOPT lacks the ability to model and detect a situation as a sequence of temporally-related contextual subspaces. An overview of Sitavio’s architecture is given in Figure 10. To experience Sitavio in an end-to-end fashion, we have implemented an exemplary context-aware mobile application that listens to and visualizes situation-specific notifications for demonstration purposes. It has built-in geofencing functionality so that Sitavio can reason upon the location of the mobile device with respect to predefined geofences. Hence, the mobile application serves as a context-aware application (for consuming notifications) and a sensor (for monitoring the mobile device’s position) at the same time.

A sensor connector attaches a physical, virtual, or logical sensor to Sitavio. Its main task is to process the stream of incoming sensor-specific events originating from a sensor or a sensor service and to detect whether a given condition as defined by a sensor predicate has changed from being fulfilled to not being fulfilled or vice versa. In case of such an incident, it will report the status of the sensor predicate (“fulfilled \( \rightarrow \) true”, “not fulfilled \( \rightarrow \) false”) to the SME. An exemplary predicate for a sensor connector linked to a geofencing sensor asserts that a mobile device is located within a particular region. The sensor connector will report the status of the sensor predicate to the SME as soon as

FIGURE 10. The architecture of Sitavio (marked in blue), plus the context-aware applications (upper right) and sensors/sensor services (bottom).
the observed mobile device enters or leaves the region of interest. The interface of a sensor connector towards the SME is clearly-specified and sensor-independent while the communication method towards the sensors or sensor services can be chosen arbitrarily, depending on the sensor-specific requirements and protocols. The prototypical setup of Sitavio encompasses a geofencing, a time, and a state sensor connector. The latter Enables the modeler to put distinct CFGs into relation to each other, a powerful feature for the definition of cascaded context abstraction logic. This is accomplished by the specification of a contextfence constraint that refers to a contextfence of another CFG. During operation, the contextfence constraint is satisfied if Sitavio considers the environment to be within the noticeable contextual state of the referred CFG's contextfence. With the state sensor connector, Sitavio acts as a context-aware middleware (emitter of state-related notifications) and, at the same time, as a consumer for state-related notifications of the referred contextfence within another CFG.

As the core component of Sitavio, the SME is responsible to keep track of the environmental states with respect to a set of situation models. It implements and runs a sDFA instance for each CFG as presented in Subsection III-C3. In case an sDFA changes into or leaves a noticeable contextual state, it will notify all subscribed context-aware applications about this incident. Since the sDFA instance is always within a definite state, the SME can parametrize the sensor connectors with the currently necessary state-specific sensor predicates. For example, during the state “Automobile Store” of the car dealer scenario, the geofencing sensor connector will monitor whether a customer is within the automobile store or outside. Former sensor predicates of the CFG, such as whether the customer is within the competitor’s store, are of no relevance for the current state of sDFA instance and are therefore not being monitored anymore. Each sensor connector monitors only sensor predicates of a CFG that are currently relevant in order for the SME to decide whether the sDFA instance remains within the current state or changes into one of the directly following states. Hence, the progress of a sDFA instance is driven by the sensor predicate reports emitted by the sensor connectors while the reports themselves are the product of configuring the sensor connectors with state-specific predicates, originating from the state-specific λ-predicates. It must be emphasized that there is no need for the SME to interpret the sensor predicates itself. This is a crucial detail because sensor predicates are highly sensor-specific, e.g. geographical polygons for geofencing sensors or time intervals for time sensors. The SME receives the predicates as part of the parametrization via situation models and forwards them in an agnostic way to the appropriate sensor connectors when needed, without being aware of their internal structure. Each sensor connector is then responsible to interpret the predicate and to keep the SME updated about the status of the sensor predicate. At a higher abstraction level, the SME is only responsible to monitor the progress of the sDFA. The lightweight sensor connectors, on the other hand, are not aware of the sDFA. They are only responsible to monitor a set of independent sensor predicates that are being managed by the SME. Out of the SME’s perspective, a sensor connector hides the sensor-specific signal communication method and acts as a proactive logical sensor that simply emits either true or false for a given sensor predicate. Decoupling the sDFA’s progress logic from the sensor logic makes Sitavio flexible in the sense that new arbitrary sensor types can be supported by adding a new sensor connector, without the need to modify the sensor-agnostic SME.
The web-based user interface serves primarily as a convenient means for technical and non-technical experts to create, edit, and manage situations graphically. The situation editor allows to interactively create and modify CFGs. Figure 11 shows an exemplary screenshot of the situation editor and a description of the graphical user interface component for a contextfence. Contextfence constraints ($\lambda$-predicates) are specified via sensor-specific constraint dialogs. Figure 12 shows an exemplary screenshot of a constraint dialog for geofencing sensors. Constraint dialogs hide the complexity of $\lambda$-predicates by giving a modeler the opportunity to choose the observable sensors or sensor groups ($W$) from a list (see upper left in Figure 12), to select the quantification type ("all", "at least", "at most", "at least and at most") from a drop-down list with editable interval parameters ($[k, l]$) (see upper right in Figure 12) and to define the sensor-specific part of the $\lambda$-predicate in a sensor-specific way (see bottom in Figure 12). A geofence for geofence sensors can be drawn as a geographical polygon on a map. The propositional logic combination of contextfence constraints ($\gamma(\phi)$) can optionally be specified in text form as a logic formula within the contextfence component in the situation editor. If it is omitted, then Sitavio assumes all $\lambda$-predicates to be combined in logical conjunction as this is the most intuitive notion of how a set of contextfence constraints is interpreted by a human. By adding state-appearance or state-disappearance notifications within the contextfence component in the situation editor, a contextfence is declared to represent a noticeable contextual state. The web-based user interface also facilitates the registration of new sensors and the creation and modification of sensor groups. Figure 13 shows the main view of the web interface: the situation management console. Situations are identified by names and annotated by tags. Situation monitoring can be activated and deactivated individually.

Alternatively, the SME can be customized via a RESTful interface so to let context-aware applications announce their own set of sensors and situation models, using well-structured JSON-based representations.

Although Sitavio is considered to be a prototypical implementation, non-functional properties such as extensibility, resilience, scalability, and robustness were already been taken into account during the design phase with aim to satisfy - from the very beginning - the requirements given by safety-critical potential application areas like Smart Cities and Smart Factories. In order to have a loose coupling between the SME and the sensors as well as the context-aware applications during operation, the communication happens on a publish/subscribe basis. This does not only allow to easily interchange sensor connectors but it does, in particular, provide a scalable way of communication with a potentially high amount of context-aware application or sensor connector instances. Sensor connectors and the SME are stateful by design and therefore critical bottlenecks when it comes to a high amount of observable sensors and/or sensor signal events. This challenge is tackled by encapsulating the sensor connectors into lightweight microservices which can be spun up flexibly on demand, backed up by a shardable NO-SQL database. The SME is built as a fully distributed actor system using Akka\(^5\), based on message passing principles and supporting location transparency. CFGs in form of sDFA's are thereby being filled exemplarily with situations of relevance for an ITS.

\section{V. DISCUSSION}

CFGs are essentially a combination of graphical and logical specifications of situations describing temporal relationships between different contextual subspaces. The latter concept is thereby specified by multiple first-order logic expressions that are stuck together by predicate logic. SPICE fully supports the aforementioned temporal precedence relation as introduced by Yau et al. \cite{31, 32} and, in addition, it is also possible to specify temporally unrestricted subsequent

\footnotesize
\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{constraint_editor.png}
\caption{Exemplary screenshot of the constraint editor for geofencing sensors. The selection of sensors or sensor groups in the upper left and the specification of the quantification type and parameters in the upper right is sensor-type agnostic. The user interface components for the definition of the sensor-specific constraints at the bottom are rendered in a sensor-type specific way, e.g. as a map for the graphical specification of geofences.}
\end{figure}

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{situation_editor.png}
\caption{Exemplary screenshot of the situation management console, filled exemplarily with situations of relevance for an ITS.}
\end{figure}

\footnotesize
\begin{itemize}
\item[5] \url{https://akka.io/}
\end{itemize}
conditions as proposed by Henricksen et al. [33] and similar approaches. Even though the CFG notation does not provide a special construct for the description of temporally recurring contextual conditions, such as it is possible with the repetition operator by Yau [31], such situations can be modeled through the use of loops with time constraints. Similarly, a sub-expression operator, as found in event pattern languages such as in Esper, is in the form of the every operator, is not supported by SPICE. Although the operator allows recurring event patterns to be described briefly and concisely, it is also more sophisticated to use and often, does not lead to the desired result. If parts of a situation need to be detected repeatedly, then these sub-situations need to be explicitly represented by loops in the CFG. The ability to specify loops has the additional advantage that recurrent patterns can optionally be dealt with in the context reasoning rather than the application logic.

To model temporal relations with ECA-rules, they are often linked to each other on a meta level [28]–[30], breaking the conciseness and readability of ECA-rule formalisms. SPICE, on the other hand, makes use of a graph-based formalism to interconnect temporally-related contextfences visually within a single CFG. Temporal aspects are thereby considered to be fundamental for situation modeling, rather than being dealt with on a meta level. Other ECA-rule-based approaches to situation modeling address temporal aspects of a situation in the event specification [21], [37], [40], [45], [47], [48]. So rather than linking different ECA-rules to form a common situation model, they make use of powerful and expressive logic languages to specify the triggering event of an ECA-rule. The Situation [42] or Event Calculus [43], [44], Temporal logic [34] or customized logic approaches [31], [33], [41] are very powerful with respect to the formal representation of situations, but require substantial knowledge and understanding of logic. Moreover, for the Event Calculus, the conceptual idea of a situation needs to be unraveled into possible sequences of raw events and then axiomatized properly. The same applies to pattern languages like Esper or the use of visual event pattern editors [67]–[69] for complex event processing systems.

SPICE follows a different path and tries to abstract away the event processing logic at the price of power and expressiveness. A contextfence describes the state of the environment and not the events that led to the appearance or disappearance of the contextual state. However, parallel/concurrent declarations [46] are not possible in SPICE and successive contextual states respectively contextfences cannot timely overlap within the specification. Hence, the “overlap” relationship between two intervals (here interpreted as the time intervals of the two contextual states) as described by Allen’s interval algebra [76] can not be explicitly modeled with SPICE. An additional “transition” state of the environment, in which constraints of the two temporally neighboring contextual states hold at the same time, provides a remedy, but it increases the complexity of the SPICE model by introducing a makeshift contextual state and because it also affects the specifications of the two temporally neighboring contextual states themselves.

Another shortcoming of SPICE deals with its inability to formalize global duration constraints. It is not possible, in its current form, to define a common duration constraint over multiple successive contextual states, as it is possible in logic-based approaches through nesting. This becomes necessary, for example, if a situation comprising of a linear chain of contextual states has to take a concrete maximum duration to evolve, independent of the individual durations of each contextual state. In SPICE, this can only be accomplished by extending the formalism with the possibility to nest a subgraph of the CFG into a single contextfence. The contextfence will then describe constraints that must hold for the whole subsituation, as represented by the nested subgraph. This would make it possible to specify nested “child” contextual states that appear or disappear together with the “parent” contextual state (Allen’s “start” and “finish” interval relation) or appear and disappear somewhere during the time the parent contextual state (Allen’s “during” interval relation) was active. From a conceptual point of view, the introduction of nesting capability would not break the SPICE formalism as defined in this article. It would provide the modeler with the possibility to specify a contextual state in a more fine-granular manner. But to keep the complexity level of nesting low for the situation modeler, features such as selective visibility and zooming should be integrated into graphical situation editors for SPICE.

As CFGs are not bound to any type of context feature, they can be used to describe complex geospatial situations as partly demonstrated within the car dealer scenario. Complex situations that are specified by the Geofencing 2.0 notation [25] to describe temporal and geospatial conditions for proactive LBS are therefore fully supported. The same applies to the complex geospatial conditions specified with the storyboard editor [20] or the formalism used by Welbourne et al. [54]. It is even possible to describe these geospatial situations in a more differentiated manner by taking additional context features besides the position of moving objects into consideration. For example, as introduced in iCAP [56], personal relationships could be considered as additional context information in order to detect spatial encounters of family members or friends.

Location-aware approaches make it possible to specify the situation with respect to a single moving object (entity according to Dey’s definition of the context [70]), a CFG can also be used to describe a situation in which multiple moving objects and their contexts may contribute to the evolution of a targeted situation. In more general terms, SPICE-empowered RPCASs are able to detect the occurrence of situations in which multiple entities of the same type - which can be determined during operation - experience similar situations respectively contextual circumstances. Although situation networks [58] describe contextual states and their sequential order of
appearance, it is not possible at runtime to dynamically derive the entities that share a common state from those of previous states. The same applies to the UML activity diagrams used in [62] to represent sequential context abstraction logic. For the context inference process with basic Petri nets [58], [60], multiple entities and their relevant environmental properties that commonly contribute to the manifestation of a situation could be represented by tokens. But tokens are by definition equivalent and indistinguishable. The same subset of tokens that triggered a transition within a Petri net can not be referenced to trigger a subsequent transition. They are consumed by transitions rather than being passed to the following place. With colored Petri nets, on the other side, tokens are distinguishable and can therefore represent moving users [59]. If a certain amount of tokens (users) with particular characteristics (outside of the building) turns a transition into an enabled state (users in front of a gate), then a location-aware function (gate opens) can be executed and the same tokens can be reasoned upon within a subsequent place (within the building). This corresponds to the functionality of SPICE, but without being able to define time restrictions. Lee et al. extend transitions of colored Petri nets with time constraints [57] to reason upon the time the environment remains within a contextual state. But Lee’s approach makes use of the token’s color to express sensor values instead of sensor identifiers. Therefore, it is not possible to determine nor to reason upon a set of entities with similar temporal characteristics at runtime.

Extending transitions of Petri nets with time constraints [57] would lead the context reasoning logic to act (fire a transition) as soon as internal timers expire. This may happen even in case the environmental circumstances did not change. In a SPICE-empowered RPCAS, in contrast, the evaluation of the progress function is only triggered by external environmental events. Time conditions of contextual states or transitions are only checked if an external event signals that the value of an environmental property under investigation entered or left the range as specified by the λ-predicates. This has the decisive advantage that the states of the sDFA instances can be efficiently persisted at runtime between the processing of two successive environmental events without having to continuously reserve resources for timers. This corresponds to the operation principles of modern scalable CEP engines.

In [66], a CEP engine is used to detect the occurrence of situations that influence a user’s decision making during her/his daily activities. Despite being powerful, the event pattern languages as the ones used in [66] to describe situations in textual form quickly become complex and unmanageable as soon as relationships between successive events, e.g. same person enters Automobile Store (event) after leaving playground (event), need to be specified. A CFG, on the other hand, describes the temporal causalities mainly by means of graphical nodes and edges. Although CFGs partly consist of non-graphical elements, SPICE can rather be classified as a graphical event pattern formalism specialized and abstracted for the domain of context-aware computing. It has less power with respect to the range of event patterns being specifiable than domain-independent text-based event pattern languages such as Esper’s rule language. For example, with Esper’s rule language, new events can be generated on the basis of an event pattern and be consumed by another event pattern of a higher abstraction level. Although this cannot be modeled with one CFG, it can be simulated by connecting two CFGs using the state connector. Aggregating values of event properties with the group-by operator in Esper’s rule language is useful when it comes to the calculation of averages across multiple successive events. But there is no need for such an operator in SPICE because the progress of a CFG is only triggered by events stating that a sensor predicate is fulfilled or not fulfilled. A distinct feature of event pattern languages for CEP engines is the possibility to define multiple branches of an event pattern that can or should happen in parallel in order for the event pattern to occur. A SPICE-empowered context reasoning logic, in contrast, can only consider an sDFA instance to be within one contextual state at a time. If the context reasoning logic decided to follow one of two possible branches of a CFG because the first contextfence of the selected branch matched the environmental circumstances, then the other branch becomes automatically unreachable. This limitation and the resulting loss of flexibility is traded against a more comprehensive notation that is inspired by deterministic finite automatons. If desired, however, parallel execution can be simulated by interconnecting several “slave” CFGs via state connectors with a “master” CFG. Each “slave” CFG represents thereby a single subbranch that is executed in parallel and they jointly determine the progress of a “master” CFG instance.

Enhancing workflow models with context-based conditions as done in [64], [66] allows a business process developer to incorporate situational aspects into the progress logic of a business process engine. But the event patterns need to be either formulated as event patterns with Esper’s or Drool’s rule language [66] or defined as rather simple conditions, e.g. whether the value of a contextual attribute is within the limits of a defined value range or not [64]. In the workflow model presented in [65], specified mobile application logic is interwoven with situation aspects, defined in the form of propositional logic over contextual attributes. These workflow-based approaches are able to monitor different aspects of the context in each step of the workflow, but they fall short to adequately map the temporal relationships of the different contexts underlying a situation. Each context-aware step in the workflow leads to the evaluation of the context, independently of a past evaluation. SPICE, on the contrary, models a situation as a context workflow rather than as a condition that is only being evaluated at a single point in time during the workflow model’s execution. However, SPICE can be used to empower the runtime environments of these approaches with the ability to reason upon the temporal characteristics of the environment by transparently replacing the CEP engines in [66], the Nexus platform in [64] or the mobile sensor processing logic in [65] by a
Since the context’s quality or, in general terms, uncertainty is not addressed by SPICE, CFGs are primarily used to formalize high-level situations under the assumption that the underlying contexts (as provided by the sensor connectors) already undergo a quality assurance process. Sensor connectors only notify the SME if a sensor predicate changes from being fulfilled to not being fulfilled or vice versa. A metric for the reliability of this information is not given to the SME and is therefore not considered in SPICE. However, there is nothing to stop a modeler from using CFGs to encode the quality-assurance logic over raw sensor signals itself. For example, a CFG might describe a situation that a mobile device entered a particular region of importance if and only if all geographical positions over the last 5 minutes fall into that region. The minimum dwell time acts thereby as a context quality threshold that prevents the context reasoning logic to consider a mobile device to be within the region of importance in case a single erroneous position measurement falls into that region by accident. This example of a low-level situation abstraction over raw sensor signals demonstrates the powerfulness and flexibility of SPICE. Situation models can also be split up into multiple parts, with each CFG encoding a different level of context abstraction logic. A contextual state of a high-level CFG can then directly depend upon the state of a low-level CFG, constituting a step-by-step multi-level situation abstraction process by interrelating different CFGs. Inter-CFG dependencies can thereby be modeled by means of state connectors, a powerful feature to simplify the specification of complex situations through the decomposition into multiple related simple situation specifications.

Nevertheless, whether SPICE and its reference implementation Sitavio can be used in practice depends upon a multitude of factors. Without a basic understanding of the underlying mechanisms of CFG processing and event pattern detection, it is possible to model situations that, in the form of sDFA instances, unintentionally lead to a high utilization of the context reasoning component and/or the sensor connectors. Users of event pattern languages for CEP engines or similar event processing technologies are faced with the same challenge that even a fully distributed technically scalable event processing engine quickly reaches its resource limits, e.g. as soon as there are no or weak temporal constraints defined for the occurrence of an event pattern. For example, omitting time constraints between contexts “Competitor’s Store” and “Playground” of the car dealer scenario leads a SPICE-empowered context reasoning logic to continue monitoring all customers leaving the competitors store for an infinite time, even if the chances are low that these customers will ever visit the playground or toy store. The fact that the number of sensors and sensor predicates to be observed per contextual state in an sDFA instance can be arbitrarily high is even an aggravating factor. In the worst case, several thousands of sensors are monitored with respect to a contextual state although it’s unlikely to occur.

VI. CONCLUSION

This article presented a concise, powerful and well-formalized way to model situations for rule-based proactive context-aware systems at a high level of abstraction with context flow graphs. The situation modeling formalism SPICE combines thereby proven concepts of logic and graph-based process modeling languages to form an integrated modeling method for the specification of domain-specific complex situations with temporal constraints. SPICE interprets a situation to comprise of potentially multiple contextual states that need to appear and disappear in a specific order and under particular duration constraints in order for the situation to occur and to be detected. Hence, it empowers rule-based proactive context-aware systems with the abilities to reason upon interrelated changes of the context over time, upon the duration a predefined contextual state is observed, and upon the time an environment spends in transition between contextual states. Instead of being restricted to specify situations only out of the perspective of a single entity, situation models in SPICE are able to formalize environmental circumstances in which multiple entities are supposed to experience similar contextual states from within a common situation. This distinct feature makes SPICE, implemented in a context reasoning component, particularly interesting for application areas where the individual circumstances, in which a multitude of entities can be present in, contribute commonly to a single situation as seen out of a macro perspective, e.g. such as traffic situations in a smart city. Sitavio’s web-based user interface, as the first proof of concept, shows that most of the SPICE’S complexity can be hidden behind simple-to-use type-specific constraint editor dialogs in order to keep it as easy as possible for the non-technical modeler but also as powerful as necessary for the expert. Although the corresponding reference implementation of a context-aware middleware demonstrates the technical feasibility of a SPICE-empowered context reasoning
logic, Sitavio’s practical feasibility in terms of scalability and the usability of the prototypical web-based user interface still needs to be evaluated and proven in practice.

Nevertheless, the potential application areas for SPICE-empowered RPCAs are abundant, ranging from the context-dependent automation of actions on mobile devices such as smartphones and in-car-infotainment systems, over the monitoring and proactive control of smart homes and Smart Cities, including intelligent transportation, logistics, and supply chain management systems, to the control of mission-critical connectivity systems within Smart Factories, to mention just a few. Whether its application is reasonable depends upon the premises, on the one hand, that relevant situations are well-known in advance and representable in SPICE and, on the other hand, that the unsolicited automation of a situation-dependent application function creates a significant added-value over its manual execution.

**FUTURE WORK**

Due to the SME’s property of being sensor-agnostic, Sitavio can be quickly and easily extended with new contextual features. To add a new feature, a new sensor connector and - if sensor-specific predicates should be specifiable via the graphical user interface - a sensor-specific constraint dialog for the web interface need to be implemented. Besides the already integrated geofencing, time, and state sensor connectors, we are planning to integrate several new sensors, e.g. to reason upon the weather conditions and the day of the week. Moreover, we are seeking to couple Sitavio with already well-established ECA-rule services, on the one hand, to make use of the wide range of sensors they are already supporting and, on the other hand, to enable Sitavio to act as a logical sensor for these services.

Another envisioned technical extension concerns the situation-specific notifications that are emitted by Sitavio and consumed by context-aware applications. So far, these notifications only signalize that situations of relevance appeared or disappeared, no matter what concrete sensors (or entities) actively contributed to the materialization of these incidents. This is not sufficient for some context-aware applications because they also need to get informed about the sensors that lead to the incident. For example, an RPCA for the monitoring of patients suffering Alzheimer disease might not only raise an alarm in case a patient leaves a building but it must also report the identity of the patient to the medical attendants, as inferred from the wearable position sensor. In the future, situation-specific notifications emitted by Sitavio will optionally encode sets of sensor identities so to give context-aware applications the possibility to respond in a more targeted way.

**REFERENCES**

[1] M. Weiser, “The computer for the 21st century,” *Sci. Amer.*, vol. 265, no. 3, pp. 66–75, 1991.
[2] B. Schilit, N. Adams, and R. Want, “Context-aware computing applications,” in *Proc. 1st Workshop Mobile Comput. Syst. Appl. (WMCSA)*, Dec. 1994, pp. 85–90.
[3] C. Bettini, O. Brdiczka, K. Henricksen, J. Indulska, D. Nicklas, A. Ranganathan, and D. Riboni, “A survey of context modelling and reasoning techniques,” *Pervasive Mobile Comput.*, vol. 6, no. 2, pp. 161–180, Apr. 2010.
[4] U. Alegre, J. C. Augusto, and T. Clark, “Engineering context-aware systems and applications: A survey,” *J. Syst. Softw.*, vol. 117, pp. 55–83, Jul. 2016.
[5] M. Cornacchia, K. Ozcan, Y. Zheng, and S. Velipasalar, “A survey on activity detection and classification using wearable sensors,” *IEEE Sensors J.*, vol. 17, no. 2, pp. 386–403, Jan. 2017.
[6] J. Hong, E.-H. Suh, J. Kim, and S. Kim, “Context-aware system for proactive personalized service based on context history,” *Expert Syst. Appl.*, vol. 36, no. 4, pp. 7448–7457, May 2009.
[7] O. Coutand, “A framework for contextual personalized applications,” Ph.D. dissertation, Dept. Fachbereich Elektrotechnik/Informatik, Univ. Kassel, Kassel, Germany, 2008.
[8] I. H. Sarker, A. S. M. Kayes, and P. Watters, “Effectiveness analysis of machine learning classification models for predicting personalized context-aware smartphone usage,” *J. Big Data*, vol. 6, no. 1, p. 57, Dec. 2019.
[9] A. Bikakis and G. Antoniou, “Rule-based contextual reasoning in ambient intelligence,” in *Semantic Web Rules*, M. Dean, J. Hall, A. Rotolo, and S. Tabet, Eds. Berlin, Germany: Springer, 2010, pp. 74–88.
[10] P. D. Costa, J. P. A. Almeida, L. F. Pires, and M. van Sinderen, “Evaluation of a rule-based approach for context-aware services,” in *Proc. IEEE Global Telecommun. Conf. (GLOBECOM)*, Nov. 2008, pp. 1–5.
[11] A. Demeure, S. Caffiau, E. Elias, and C. Roux, “Building and using home automation systems: A field study,” in *Proc. 5th Int. Symp. End-User Dev. - Develop. Environ. - Usability, P. Díaz, V. Pipek, C. Ardito, C. Jensen, I. Aedo, and A. Boden, Eds. Cham, Switzerland: Springer, 2015, pp. 125–140.
[12] M. Coronado and C. A. Iglesias, “Task automation services: Automation for the masses,” *IEEE Internet Comput.*, vol. 20, no. 1, pp. 52–58, Jan. 2016.
[13] F. Cabitza, D. Fogli, R. Lanzilotti, and A. Piccinno, “Rule-based tools for the configuration of ambient intelligence systems: A comparative study,” *Multimedia Tools Appl.*, vol. 76, no. 4, pp. 5321–5341, Feb. 2017.
[14] E. Gilman, “Exploring the use of rule-based reasoning in ubiquitous computing applications,” Ph.D. dissertation, Dept. Comput. Sci. Eng., Univ. Oulu, Oulu, Finland, 2015.
[15] A. Ranganathan and R. H. Campbell, “A middleware for context-aware agents in ubiquitous computing environments,” in *Proc. ACM/FI/USENIX Int. Conf. (Middleware)*, New York, NY, USA: Springer-Verlag, 2013, pp. 143–161.
[16] A. Ranganathan and R. H. Campbell, “An infrastructure for context-awareness based on first order logic,” *Pers. Ubiquitous Comput.*, vol. 7, no. 6, pp. 353–364, Dec. 2003.
[17] K. Henricksen and J. Indulska, “A software engineering framework for context-aware pervasive computing,” in *Proc. 2nd IEEE Int. Conf. Pervasive Comput. Commun. (PerCom)*, 2004, p. 77.
[18] J. C. Augusto, J. L. Crawford, S. Dobson, and D. Garlan, “Context is key,” *Commun. ACM*, vol. 48, no. 3, pp. 49–53, 2005.
[19] Z. Jaroucheh, X. Liu, and S. Smith, “Recognize contextual situation in pervasive environments using process mining techniques,” *J. Ambient Intell. Humanized Comput.*, vol. 2, no. 1, pp. 53–69, Mar. 2011.
[20] Y. Li, J. I. Hong, and J. A. Landay, “Topiary: A tool for prototyping location-enhanced applications,” in *Proc. 17th Annu. ACM Symp. User Interface Softw. Technol. (UIST)*, 2004, pp. 217–226.
[21] A. Coronato and C. De Pellegrini, “Situation awareness in applications of ambient assisted living for cognitive impaired people,” *Mobile Netw. Appl.*, vol. 18, no. 3, pp. 444–453, Jan. 2013.
[22] D. Moldovan, C. Clark, and S. Harabagiu, “Temporal context representation and reasoning,” in *Proc. 19th Int. Joint Conf. Artif. Intell. (IJCAI)*, San Mateo, CA, USA: Morgan Kaufmann, 2005, pp. 1099–1104.
[23] M. Wieland, F. Leymann, M. Schäfer, D. Lucke, C. Constantinescu, and E. Westkämper, “Using context-aware workflows for failure management in a smart factory,” in *Proc. 4th Int. Conf. Mobile Ubiquitous Comput. Syst., Services Technol. (UBICOMM)*, 2010, pp. 379–384.
[24] H.-C. Liao and C.-C. Tu, “A RDF and OWL-based temporal context reasoning model for smart home,” *Inf. Technol. J.*, vol. 6, no. 8, pp. 1130–1138, Nov. 2007.
[25] S. Rodriguez Garzon and B. Deva, “Geofencing 2.0: Taking location-based notifications to the next level,” in *Proc. ACM Int. Joint Conf. Pervasive Ubiquitous Comput. (Ubicomp)*, 2014, pp. 919–930.

S. Rodriguez Garzon, B. Louis: Context Flow Graphs: Situation Modeling for Rule-Based Proactive Context-Aware Systems
P. J. Brown, J. D. Bovey, and X. Chen, “Context-aware applications: From the laboratory to the marketplace,” IEEE Pers. Commun., vol. 4, no. 5, pp. 58–64, 1997.

A. K. Dey, “Context-aware computing: The CyberDesk project,” in Proc. AAAI Spring Symp. Intell. Environ., 1998, pp. 51–54.

A. Schmidt, K. A. Aidoo, A. Takahama, K. A. Valaerhoven, and W. Van de Velde, “Advanced interaction in context,” in Proc. 1st Int. Symp. Handheld Ubiquitous Comput. (HUC), Springer-Verlag, 1999, pp. 89–101.

P. D. Costa, L. F. Pires, M. van Sinderen, and T. Broens, “Controlling services in a mobile context-aware infrastructure,” in Proc. 2nd Workshop Context Awareness Pervasive Syst. (CAPS), no. 2, K. David and S. Haseloff, Eds. Kassel, Germany: Kassel Univ. Press, Jun. 2006, pp. 153–166.

Y. Qiao, K. Zhong, H. Wang, and X. Li, “Developing event-condition-action rules in real-time active databases,” in Proc. ACM Symp. Applied Comput. (SAC), 2007, pp. 511–516.

S. S. Yau and F. Karim, “Reconfigurable context-sensitive middleware forADS applications in mobile ad hoc network environments,” in Proc. 5th Int. Symp. Auto. Decentralized Syst., 2001, pp. 319–326.

S. S. Yau, F. Karim, Y. Wang, B. Wang, and S. K. S. Gupta, “Reconfigurable context-sensitive middleware for pervasive computing,” IEEE Pervasive Comput., vol. 1, no. 3, pp. 33–40, Jul. 2002.

K. Henricksen and J. Indulska, “Developing context-aware pervasive computing applications: Models and approach,” Pervasive Mobile Comput., vol. 1, no. 1, pp. 37–64, Feb. 2006.

A. Galton, Ed., Temporal Logics and Their Applications. New York, NY, USA: Academic, 1987.

A. M. Hinze, “A-medias: Concept and design of an adaptive integrating event notification service,” Ph.D. dissertation, Berlin, Germany, Freie Universität Berlin, 2003.

T. Beer, J. Rasinger, W. Höpken, M. Fuchs, and H. Werther, Exploiting ECA-Rules for Defining and Processing Context-Aware Push Messages. Berlin, Germany: Springer, 2007, pp. 199–206.

J. C. Augusto and C. D. Nugent, “The use of temporal reasoning and management of complex events in smart homes,” in Proc. 16th Eur. Conf. Artif. Intell. (ECAI), 2004, pp. 778–782.

M. Fisher, D. M. Gabbay, and L. Vila, Handbook of Temporal Reasoning in Artificial Intelligence (Foundations of Artificial Intelligence), vol. 1, 1st ed. Amsterdam, The Netherlands: Elsevier, 2005.

J. U. Liu, J. C. Augusto, and H. U. Wang, “Considerations on uncertain spatio-temporal reasoning in smart home systems,” in Applied Artificial Intelligence, P. D’hoedt, P. F. Fantoni, and D. Ruan, Eds. Singapore: World Scientific, 2006, pp. 817–824.

J. G. Tan, D. Zhang, X. Wang, and H. S. Cheng, “Enhancing semantic spaces with event-driven context interpretation,” in Pervasive Computing, H. W. Gellersen, K. Want, and A. Schmidt, Eds. Berlin, Germany: Springer, 2005, pp. 80–97.

T. Magnherini, A. Fantechi, C. D. Nugent, and E. Vicario, “Using temporal logic and model checking in automated recognition of human activities for ambient-assisted living.” IEEE Trans. Human-Mach. Syst., vol. 43, no. 6, pp. 509–521, Nov. 2013.

J. McCarthy, “Situations, actions, and causal laws,” Stanford Artif. Intell. Lab., Stanford, CA, USA, Tech. Rep. AD-785 031, 1963.

R. Kowalski and M. Sergot, “A logic-based calculus of events,” New Gener. Comput., vol. 4, no. 1, pp. 67–95, Mar. 1986.

R. Kowalski, “Database updates in the event calculus,” J. Log. Program., vol. 12, nos. 1–2, pp. 121–146, Jan. 1992.

C. Elftanni, A. Friday, N. Davies, and K. Cheverst, “Utilising the event calculus for policy driven adaptation on mobile systems,” in Proc. 3rd Int. Workshop Policies Distrib. Syst. Netw., 2002, pp. 13–24.

L. Chen, C. Nugent, M. Mulvenna, D. Finlay, X. Hong, and M. Poland, “Using event calculus for behaviour reasoning and assistance in a smart home,” in Smart Homes Health Telematics, S. Helal, S. Mitra, J. Wong, K. C. Chang, and M. Mokhtar, Eds. Berlin, Germany: Springer, 2008, pp. 81–89.

B. Hu, T. Patoks, A. Chibani, and Y. Amirat, “Rule-based context assessment in smart cities,” in Web Reasoning and Rule Systems, M. Krötzsch and U. Straccia, Eds. Berlin, Germany: Springer, 2012, pp. 221–224.

T. Patoks, I. Chryssakis, A. Bikakis, D. Plexousakis, and G. Antoniou, “A reasoning framework for ambient intelligence,” in Artificial Intelligence: Theories, Models and Applications, S. Konstantopoulos, S. Perantonis, V. Karkalekis, C. Spyropoulos, and G. Vouros, Eds. Berlin, Germany: Springer, 2010, pp. 213–222.
[74] C. Perera, A. Zaslavsky, P. Christen, and D. Georgakopoulos, “Context aware computing for the Internet of Things: A survey,” IEEE Commun. Surveys Tuts., vol. 16, no. 1, pp. 414–454, 1st Quart., 2014.
[75] P. Hirmer, M. Wieland, H. Schwarz, B. Mitschang, U. Breitenbücher, S. G. Sáez, and F. Leymann, “Situation recognition and handling based on executing situation templates and situation-aware workflows,” Computer, vol. 99, no. 2, pp. 163–181, 2017.
[76] J. F. Allen, “Maintaining knowledge about temporal intervals,” Commun. ACM, vol. 26, no. 11, pp. 832–843, Nov. 1983.

SANDRO RODRIGUEZ GARZON (Member, IEEE) received the Dipl-Ing. degree in computer engineering from the Technische Universität Berlin, in 2005, and the Ph.D. degree in computer engineering from Technische Universität Berlin, in 2013. From 2006 to 2007, he worked as a Technical Consultant for the sd&m AG (now Capgemini sd&m AG) and has participated in several IT projects for major car manufactures such as Volkswagen and Audi. From 2007 to 2013, he worked as a Research Assistant with the Daimler Center for Automotive IT Innovations, Technische Universität Berlin, with a three-month research guest visit at Mercedes-Benz Research and Development North America, Palo Alto, CA, USA. Since 2013, he has been working as a Senior Researcher with the Telekom Innovation Laboratories within the group of Service-centric Networking of the Technische Universität Berlin. His research interests lie in the areas of the Internet of Things, intelligent and ambient environments, smart cities, and Industry 4.0.

BERND LOUIS received the B.Sc. and M.Sc. degrees in computer science from the Technische Universität Berlin in 2017. Prior to his studies in computer science, he worked as a Software Engineer in the fields of web applications, web services, and community systems (Erasys GmbH, Berlin, Germany), software engineering (Swarc GmbH, Düsseldorf, Germany), and in gaming (Zombiefood GmbH, Berlin, Germany, now Digidiced GmbH). During his studies, he worked as a Student Assistant with the Distributed Artificial Intelligence Laboratory and the Database Systems and Information Management Group (DIMA), Technische Universität Berlin. From 2017 to 2018, he worked as a Research Assistant with the Telekom Innovation Laboratories within the group of Service-centric Networking at the Technische Universität Berlin. Since 2019, he has been working as a Software Engineer at SoundCloud Limited, Berlin.