CIM-CSS: A Formal Modeling Approach to Context Identification and Management for Intelligent Context-Sensitive Systems

Ali Mahmoud Baddour1,2, Jun Sang1,2, Haibo Hu1,2, Muhammad Azeem Akbar1,2, Hassan Loulou3, Ahmad Ali4, Kanza Gulzar1,2

1Key Laboratory of Dependable Service Computing in Cyber Physical Society of the Ministry of Education, Chongqing University, Chongqing 400044, China.
2School of Big Data & Software Engineering, Chongqing University, Chongqing 401331, China
3STIC Doctoral School, Paris-Saclay University, Paris 91400, France
4Faculty of Mechanical and Electrical Engineering, Tshreen University, Latakia 2234, Syria

Corresponding authors: Ali Mahmoud Baddour (baddour.ali85@gmail.com), Jun Sang (jsang@cqu.edu.cn), and Haibo Hu (haibo.hu@cqu.edu.cn)

This work was supported by National Natural Science Foundation of China (No. U1836114) and Chongqing Research Program of Basic Science and Frontier Technology (No. cstc2017jcyjB0305).

ABSTRACT Context modeling is often used to relate the context in which a system will operate to the entities of interest in the problem domain. It remains the case that context models are inadequate in emerging computing paradigms (e.g., smart spaces and the Internet of Things), in which the relevance of context is shaped dynamically by the changing needs of users. Formal models are required to fuse and interpret contextual information obtained from heterogeneous sources. Here, we propose an integrated and formal context modeling approach for intelligent systems operating in context-sensitive environments. We introduce a goal-driven, entity-centered identification method for determining which context elements are influential in adapting the system behavior. We then describe a four-layered framework for metamodeling the identification and management of context. First, the framework presents a formal metamodel of context. A formalization of context using first-order logic with relational operators is then presented to specify formally the context information at different abstraction levels. The metamodel therefore prepares the ground for building a formal modeling language and automated support tool (https://github.com/metamodeler/CIM-CSS). The proposed model is then evaluated using an application scenario in the smart meeting rooms domain, and the results are analyzed qualitatively.

INDEX TERMS Context, Relevance, Context Modeling, Metamodeling, Formal Methods, Context-Sensitive Systems

I. INTRODUCTION

The “Ubiquitous computing” era envisioned by Weiser [1] is becoming a reality. In particular, the miniaturization of electronic devices and the proliferation of smart objects have allowed advanced information and computing technologies to be used in people’s everyday activities. These developments have transformed desktop computing, and various new paradigms are emerging, including context-aware systems (CAS), ambient intelligence (AmI), self-adaptive systems (SAS), the Internet of Things (IoT), cyber-physical systems (CPS), and ubiquitous information systems (UIS) [2]-[5]. These systems are used in a variety of domains and at various scales, including in smart homes and rooms, web and mobile applications, smart mobility systems, and logistics. Such systems are generally sensitive to the context in which they operate. In other words, a system should be aware of contextual changes and adapt its behavior (in terms of service and information) to accommodate users’ needs [2]-[5]. This requires the system to observe, process, and understand the situation in which it is used, to allow it draw high-level abstract conclusions about it [6]. These conclusions are called the situational contexts, are of great interest to the system, and can be used to provide smart functionalities [7]. Dey and Abowd [8] stated that context is any information that can be used to characterize the situation of an entity (e.g. a person, a place, an object, a user, or an application). The Problem Frames (PF) approach [9] represents the context in which a problem exists, and in which a machine will operate in terms of domain properties and shared interfaces.
Context-sensitive systems (CSSs) increasingly operate in complex and dynamic environments that influence their behavior, thus it is crucial that the context in which a CSS is embedded must be assessed, analyzed, and modeled when the CSS is being designed [10]. There are, however, many challenges related to context modeling, particularly related to the complexity of context and its relevance to a CSS, and these must be taken into account. More specifically, the system world in which a problem exists and in which an application runs can be reflected in a great number of possible states of the physical and virtual entities in the surrounding environment (e.g., temperature); of the users (e.g., preferences); and of technologies (e.g., network status) and hardware components (e.g., battery level). However, only a partial state of the world is influential when designing a CSS, and it is this that can be referred to by the term “relevant context”. Moreover, these states are subject to various types of constraints such as validity (e.g., for which point of time or during which range of time a state is valid and usable by an application), quality metrics of the data (e.g., credibility and precision), different modalities (e.g., binary, numeric, and categorical values), and rich relationships among states (e.g., causality, contradiction, aggregation, and dependence). With the development of above-mentioned computing paradigms and cognitive domains (e.g., smart spaces [11]) that are becoming increasingly complex and highly dynamic at run-time, the identification of relevant states and the management of diverse contextual aspects become non-trivial tasks that directly influence the decision processes regarding design activities, including context modeling. Therefore, an approach is required for (1) assessing the relevance of context states to determine the most influential in CSSs behavior, and (2) addressing the complexity exhibited by the relevant states at different levels to fuse and interpret sensor data and domain knowledge into a context model enriched with situational contexts that are essential for and highly consumable by CSSs.

In practice, such an approach is needed when engineering systems operate in IoT-based environments such as smart spaces that are characterized by large volumes of data sensed by heterogeneous devices and sensors [12], and thus a large number of states and variations can be generated and considered potential elements of context. Furthermore, the contextual information can be exchanged and shared as services among a diversity of smart objects in the IoT (e.g., [13]). Other factors can complicate the engineering of intelligent systems operating in these environments, such as the need to allow different interpretations of the context to be made, and they can then be used by multiple applications to provide customized services and/or information to users at different levels of usage; individual level (e.g., a healthcare monitoring application in a smart home system should provide suggestions and send notifications based on an inhabitant’s monitored health status: good, fair, serious, or critical), group level (e.g., a smart home system should auto-operate home appliances based on family activities: outdoor, indoor, party, reading, cooking, and eating), city level (e.g., a smart mobility system should support the planning of trips and travel arrangements based on monitored traffic, current environmental conditions, and travelers’ preferences and budgets), and national level (e.g., a smart e-government system of systems should support decision-making and present evaluations of policies based on high-level coarse-grained measurements of energy consumption, pollution, and congestion). Besides the increasing complexity of context evolution at different application levels, a many-to-many relationship between visible states at the sensor data level and the abstract states at a particular application level must also be taken into account. For instance, several devices (e.g., smartphone sensors, RFID tags, cameras, and physical sensors) available in a smart home system might generate the location contexts (e.g., ‘livingRoom’ and ‘kitchen’) and other physical contexts (e.g., ‘noisyLivingRoom’ and ‘highOccupancy’), which can be logically correlated with other sensed contexts (e.g., ‘weekendTime’ and ‘stoveOn’) to infer simultaneous activity contexts (e.g., ‘cooking’ and ‘party’). The above-mentioned scenarios necessitate modeling and inferring capabilities that allow multi-level fusion and interpretation for designing intelligent CSSs [14].

In light of the foregoing, it is difficult to determine the relevant context elements from both the data management and system design perspectives [15]. Accounting for the former (i.e., by identifying which visible, monitorable, and domain-dependent facts can be respectively sensed, inferred, and accumulated from heterogeneous sources in a system or system of systems) is essential to make data storage and processing more efficient, while accounting for the latter (i.e., by identifying which abstract states are the most interesting, in which different applications should adapt their behaviors at different levels) is essential to make the design process more effective. Therefore, we attempted to develop an integrated approach to identify and manage the contextual information for CSSs operating in complex and dynamic environments. Such an approach is required to design context models that can connect the recognition of context (context producers e.g., sensors) and the use of context (context consumers e.g., applications).

A comprehensive identification of context is required to allow CSS experts and engineers to select a consolidated list of context elements that are influential in affecting a CSS behavior during its run-time. Indeed, the inclusion of human experts and end-users with different cognitive capabilities in context management and system design processes has recently been recognized as a key enabler for intelligent CSSs [16], [17]. Thus, the addition of a further level of complexity in multiple stakeholders’ aspects makes evaluating the context an even more challenging task. Therefore, we developed a context identification method that takes different views into account and addresses: (1) device expert’s concerns through bottom-up decision processes i.e., collecting low-level facts recognizable by devices, which can be abstracted to higher-
level states through further analysis, (2) CSS design expert’s concerns through top-down decision processes i.e., collecting high-level states of interest to a CSS, which can be refined to lower-level states, and (3) the concerns of domain experts and end-users/operators through observations and assessments i.e., extracting cognitive knowledge and facts (e.g., semantic patterns or procedures that can be used to correlate two concepts) about organizational or operational environments and individual users in a particular domain. The elicitation of domain knowledge and facts can support seamless integration of data and information acquired from diverse sources and belonging to different levels of abstraction.

Therefore, an effective and efficient design process for CSSs should incorporate such a hybrid method of identification as a fundamental activity. The efficiency of a CSS design process can be improved using a method that systematically assesses the contexts from which the most influential can be identified. Using an approach that consolidates the data and knowledge acquired from different sources and/or multiple stakeholders into a context model from which the CSS design can be derived and specified, it is also possible to improve the design process effectively at an early stage of the development of the software.

Several approaches for modeling the context can be found in the literature. There has recently been a growing tendency to use model-based tools [18] and formal techniques [19] when designing a CSS. In particular, recent studies have used formal modeling notations such as metamodels, ontologies, and logical programs for effective specification and management of the context and associated concepts [20]-[26]. The main aim was to address the new characteristics of CSSs and meet their emerging context requirements from the perspectives of data management and system design. However, it remains the case that none of the existing studies provides a full formal treatment for the different and complementary aspects of the context mentioned above: the majority of the studies provided no assessment or formalization of the relevance of context (e.g., [20]-[25]); some provided partial specification capabilities that are inadequate for expressing and managing the diversity and variety of the context in terms of modeling elements related to its structural, behavioral, and dynamic aspects (e.g., [20]-[24], [26]); they lack a multidisciplinary elicitation mechanism that allows multiple stakeholders to address their different aspects and concerns in a context model. In other words, none of the existing approaches support a comprehensive modeling approach that formally facilitates the identification and management of the context at both the data and system levels for designing intelligent CSSs. Therefore, we attempted herein to propose such an approach, termed CIM-CSS, by first developing a multi-view context identification method driven by focal elements (e.g., goals/requirements shared between the user’s world and the software system). To achieve this, the relevance of context to a CSS needs to be identified by eliciting and assessing the context elements from three knowledge management and system design perspectives each of which takes a different stance on the nature of context and contextual processes: (i) technological (e.g., the visible and inferred contexts generated by devices available in a CSS [14]), (ii) functional (e.g., the influences of the dynamic part of context and the intentional or interactional aspects of a CSS on each other [27], [28]), and (iii) cognitive (e.g., the semantic properties of and relationships between the entities of interest in a particular domain and the structural part of context [29]). The elicited elements, including the context, can be represented as run-time components reflecting the continuously changing state of a system using Models@Run.Time [30].

The second aim of this study was to consolidate key notions related to the concept of context in an integrated modeling approach. This was achieved using a formal treatment of the various aspects of context through (i) a four-layer metamodeling framework aligned with the Meta-Object Facility (MOF) standard and Model-Driven Architecture (MDA) and (ii) formal methods including unified modeling language (UML), object constraint language (OCL), and first-order logic (FOL).

Here, we attempt to address the following research questions (RQ).

(RQ1) How can a context element relevant to a system be identified systematically.

(RQ2) How can the constituents and features that support context identification and management be meta-modeled and formalized?

**ROADMAP**

 Relevant previous work is described and discussed in Section II. The formal modeling approach (CIM-CSS) to identifying and managing the context is described in Section III. The model-driven development of an automated support tool is outlined in Section IV. The proposed approach is evaluated in Section V using a smart meeting room scenario, then the proposal is analyzed qualitatively. Implications and limitations of CIM-CSS are described in Section VI. Conclusions and future work are presented in Section VII.

**II. RELATED WORK**

Several methods have been proposed for handling the context and enabling a software system to be context-sensitive. However, in this study we focus on requirements engineering (RE) models and context modeling approaches that specify how and to what extent they deal with the context. First, the work on engineering context-sensitive systems is reviewed. The approaches for modeling the system requirements with the context are then discussed. Next, we review widely accepted definitions of context in the literature, particularly those that show the complexity caused by the different views of context and its variability (e.g., locations, profiles, and physical conditions). Important approaches related to context modeling are then presented and analyzed.
A. ENGNEERING CONTEXT-SENSITIVE SYSTEMS
A significant number of methods have been proposed for engineering context-sensitive systems over the last two decades (see e.g., [2]-[5] for a survey). A taxonomy of methods, middleware, approaches, and frameworks for considering context-awareness in the Internet of Things (IoT) and self-adaptive systems have been reviewed in [5] and [4], respectively. These methods and frameworks can assist software engineers to model, manage and reason about context information. However, there is an assumption that the context information is identified.

In a recent survey, Alegré et al. [2] analyzed methodologies and techniques for engineering context-aware systems. Their analysis [2] shows that the methodologies do not consider definition and understanding of context as a preliminary and explicit stage in the design process. In addition, they conclude that the work they investigated offers no methods and tools for identifying the context at an early design phase [2]. Based on the results of a questionnaire carried out with 750 context-aware systems developers, they determined that one of the most desirable features in a context-aware design process is representing situations (i.e., situational contexts) where the system should adapt [2]. Based on a study of the literature, they state that: “All context information modeling and reasoning techniques need to enable the situation representation, but there is no support for understanding the situations and the contexts that they are going to be represented, stemming from the requirements.” [2, p.23]. Our own approach, in which identification of the context is a preliminary and fundamental activity in the CSS design process, thus addresses an important problem for system engineers and other stakeholders involved in CSS design.

In another recent survey, Guinea et al. [3] reviewed work on software development for ubiquitous systems. Based on the results of their review, they state: “It being such an important concept in the development of ubiquitous systems, there is a need for more research on issues related to the concept of context, important for different phases of the development cycle. In relation to design, it is necessary to further investigate how to design systems that can automatically adapt based on their context, as well as to study light-weight context modeling techniques to avoid overheads and low performance of the running system.” [3, p.19]. Our approach, in which modeling the contextual information is driven by system goals and is based on a formal language and a graphical tool, thus meets an important requirement for the CSS design phase.

B. MODELING REQUIREMENTS WITH CONTEXT
Several works consider the importance of domain knowledge for a successful engineering of computer systems. RE methodologies e.g., KAOS [31] analyzes and models user’s needs and system’s objectives in goal models where domain concepts are represented in terms of conceptual objects e.g., entities, associations, attributes, and events, whereas i* [32] provides strategic dependency model with social concepts and relations. Alternatively, the Problem Frames approach [9] introduces context diagrams to represent the problem world and the machine solution using three categories of domain entities: biddable, causal, and lexical as well as interfaces among them. Moreover, several approaches investigated the relationship between system aspects and context at an early stage of software development. For example, context-oriented domain analysis (CODA) [33] is a method for capturing context requirements using variation points, context conditions and context-dependent adaptations. In another work, Ali et al. [28] propose an approach for modeling and analyzing contextual requirements, which explicitly represents the relationship between context and goals by weaving contexts to variation points at the goal level, and classifying the influence of context on goals into activation, required, and quality conditions. Alternatively, an RE method for ubiquitous systems (REUBI) [34] introduces an interdependency graph for modeling goals with context by enriching the models with contextual dependencies e.g., context situation, context attribute, and context value.

However, the work presented above is generic and lacks expressive notations for modeling domain particularities e.g., sources, dimensions and types of contextual facts as well as contextual associations e.g., acquisitions (sensed, derived, profiled, and static) and permissions (private, group, restricted, and all), that allow the embedded heterogeneity to be captured, and thus the complexity of the context to be handled. Therefore, a modeling language that provides formal constructs, rich semantics, model checking and inferencing capabilities is required [35] for better understanding, identification, modeling, and derivation of detailed and tacit domain knowledge from which context information can be drawn.

C. DEFINITIONS AND PERSPECTIVES ON CONTEXT
It is essential to define context before using it in the design of CSSs [36]. However, there is no consensus on the definition of context among different academic disciplines [37]. Different perspectives on context have been used in the computer science discipline, and this can be attributed to different understandings and usages of the term ‘context’ in different fields.

Various definitions of and perspectives on context are shown chronologically in Table 1. Some definitions ([8], [38], [39], and [40]) have a static view of the context (i.e., conceptual and representational perspectives) because context elements can be acquired and specified using a model at design-time stage, and they also have a dynamic view of the context (i.e., interactional and operational perspectives) because the context will depend on a particular interaction while a system is running (i.e., run-time). On the other hand, some definitions are perspective-oriented: [9], [28], and [41] indicate that the understanding of context is established from the perspective of the problem domain, the actor, and the user.
D. CONTEXT MODELING

As shown in the work reviewed above, continuous research efforts have been dedicated to the subject of context management and modeling, and several approaches have been proposed. To review the relevant work as comprehensively as possible, we proposed an analysis framework composed of a set of criteria/requirements to evaluate the features of context modeling approaches from different viewpoints: what aspects of context are addressed (i.e., a theoretical viewpoint), what capabilities of context model are provided (i.e., a conceptual viewpoint), and what techniques supporting the design process are developed (i.e., a practical viewpoint). Some of these requirements are collected from surveys [2]-[6], [19], while others are identified based on our experience in CSSs research and development. The most recent work related to modeling context information for CSS design is discussed below.

MLContext [20] is a textual domain-specific language (DSL) developed for modeling the context information in context-aware systems. To represent the context, the MLContext metamodel is specified in terms of a collection of meta-classes such as ComplexContext, SimpleContext, ContextInformation, ContextSource, CategorySection, and Category. MLContext is entity-centered rather than category-centered and it supports a taxonomy of types of context and code generation. MLContext models are platform independent and can be reused in different context-aware applications. However, this work presumes context information is pre-determined and it does not provide a means for gaining insight into the context to determine which elements are relevant to a system. The behaviors and dynamics of context are partially supported by MLContext, which particularly require capabilities to represent important concepts e.g., focus, validity constraints, permissions, acquisition types, and quality metrics. In addition, MLContext is not accompanied with a detailed process that provides methodological guidelines for pursuing the modeling and design activities.

CONSERT [21] is an ontology-based model and engine to represent and infer context information. The proposed ontology is used in the CONSERT engine to support rule-based event processing and reasoning, context statement manipulation, and constraint validation. CONSERT provides an expressive representation of and ontological reasoning on context using semantic web technologies such as OWL and SPARQL. To specify and manage the context information, CONSERT model uses three sub-modules: core vocabulary including concepts such as ContextEntity, and ContextAssertion for context modeling, annotation vocabulary (e.g., ContextAnnotations) for context manipulation, and constraint vocabulary (e.g., ContextConstraintTemplates) for constraint detection. The main activity cycle of CONSERT is based on purely technical procedures and architectures. However, this work does not incorporate multiple stakeholders to allow them to consider different design perspectives that influence modeling and decision processes, particularly for supporting the multi-disciplinary collection of context information and identification of relevant elements. Moreover, CONSERT ontology does not express important relationships among contexts such as dependency and causality.

<table>
<thead>
<tr>
<th>Work</th>
<th>Definition</th>
<th>Perspective</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dey and Abowd, 1999</td>
<td>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 [8].</td>
<td>Conceptual</td>
</tr>
<tr>
<td>Brezillon and Pomerol, 1999</td>
<td>Context is relevant to the focus that is a step of problem-solving, task performing, or decision-making process [43].</td>
<td>Interational</td>
</tr>
<tr>
<td>Jackson, 2001</td>
<td>Context represents properties of the problem world (i.e., the physical context) and the machine (i.e., the software) in terms of hiddable, causal, and lexical domains, and shared interfaces between the domains [9].</td>
<td>Knowledge management</td>
</tr>
<tr>
<td>Jang and Woo, 2003</td>
<td>A context is created from sensors and identified by the 5W1H: Who (user identity), What (object identity), Where (location), When (time), Why (intention), and How (user gesture) [41].</td>
<td>Problem-oriented</td>
</tr>
<tr>
<td>Dourish, 2004</td>
<td>Context is relevant to a particular user’s activity when thinking about its role in interactive systems, and it is dynamically defined, and actively produced, maintained and enacted with reference to that activity [38].</td>
<td>User-centric</td>
</tr>
<tr>
<td>Bradley and Dunlop, 2005</td>
<td>The dimensions of context that affects user’s actions and application’s services include social events, user preferences, computing capabilities, tasks, temporal, and physical conditions [44].</td>
<td>Multidisciplinary</td>
</tr>
<tr>
<td>Zimmermann et al., 2007</td>
<td>Individuality, activity, location and time, and relations are fundamental categories for characterizing the dynamic properties of context emerging from context transitions and sharing contexts among entities [39].</td>
<td>Operational</td>
</tr>
<tr>
<td>Ali et al., 2010</td>
<td>Context is a partial state of the world that is relevant to an actor’s goals [28].</td>
<td>Goal-oriented</td>
</tr>
<tr>
<td>Vieira et al., 2011</td>
<td>A contextual element is any piece of data or information that can be used to characterize an entity in an application domain [40]. The context of an interaction between an agent and an application is the set of instantiated contextual elements that are necessary to support the task at hand [40].</td>
<td>Interational</td>
</tr>
</tbody>
</table>
CTXs-Maude [22] is a formal DSL based on model-driven engineering (MDE) techniques and formal methods to provide algebraic specification and verification capabilities for developing context-aware systems. The CTXs-Maude grammar supports formal semantics for formalizing the structure and behavior of context-aware systems. To represent the context information, the metamodel of CTXs-Maude is composed of four layers in which the following meta-classes are specified: Sensor and SensorsTypes (the sensing layer); Context, AtomicContext, CompositeContext, ContextState, ElementaryState, and HighState (the context layer); Action, ComponentActions, and PortActions (the interaction layer); and Component, Port, and Service (the functional layer). However, this method did not use an expressive conceptual model for the context. The development process of CTXs-Maude did refer to the idea of ‘separation of concerns’ between the system functional elements and contextual information for having identification steps allocated to each; nevertheless, this process did not provide a means to guide or support elicitation and identification activities of the context.

CAMeOnto [24] is a generic ontology for modeling the context. CAMeOnto is used by a context-aware reflective middleware called CARMiCLOC. CARMiCLOC provides a set of services including context acquisition, context modeling, context reasoning, context distribution, and context service management. CAMeOnto is based on a set of models to consider six contextual classes: activity, user, time, device, service, and location. The activity model describes features and abilities that can be involved in an activity. The time model represents temporal properties of the context. The user model encodes user’s profile and preferences. The device model represents the characteristics and status of software and hardware components. The service model describes the characteristics and qualities of the required services. The location model represents the properties of the environment, outdoor and indoor components, and positions. CAMeOnto supports an expressive representation of the structural aspect of context using a variety of classes, subclasses, and relationships (e.g., Activity, User, Time, Device, Service, Location, and Role), and based on a hierarchy of parent ontologies. However, CAMeOnto does not explicitly incorporate classes to fully represent the behavioral aspect of context (e.g., Focus, Action, and Rule) nor does it address the dynamic constraints on context (e.g., Acquisition Types, Validity, and Access Types). In addition, the design and instantiation of CAMeOnto does not offer a methodological process for collecting and identifying the context considering different views and concerns.

CAADA [23] is a model-based approach for context-aware application development. CAADA is validated through an MDA-based framework called DONCIR. It is based on the following components: context generator, context manager, controllers, and context executer. The context generator automatically builds a context instance (XML file) based on the inputs from a user. The context manager fetches changes in context elements based on the instance context model when an application is running. The controllers observe the dynamic context, check a context value acquired from a context provider, compare it with the current context, and trigger a context change event (if a new value is sensed) to the context executer. The context executer allows an application to adapt if a change event is triggered. The context metamodel and architecture underlying CAADA support context management, context change management, and adaptation management, which allows a separation of concerns. CAADA models the context using a set of classes such as context, context property, context provider, context association, and focus. However, the development process of CAADA does not involve an activity to assess the context before its representation. Furthermore, the collection of contexts from different stakeholders is not considered in CAADA, through which the context can be only inserted based on a technical perspective (i.e., a developer inputs a context model).

3LConOnt [25] is a three-level model composed of an upper-level ontology, a middle-level ontology, and a lower-level ontology for modeling context information in context-aware computing. The proposed ontology is validated in a service-oriented computing paradigm using a context-aware framework throughout the context lifecycle: acquisition, modelling, reasoning, and distribution. The upper-level ontology represents the highest abstract level in this work. It defines context information and entities in terms of a taxonomy of high-level classes such as Time, Profile, Environment, Role, States and Status, Location, Activity, Resource, and Agent. The middle-level ontology is defined by a set of modules that can be reused, extended, and adapted within the same level and with the upper- and lower-level ontologies. For instance, an Activity module can be divided into concepts such as Action, Event, Task, and Process, and properties such as has-Event and has-Performance. The lower-level ontology represents domain-dependent classes and properties. The 3LConOnt context model is built by collecting and integrating context knowledge from different ontologies to support a standardized and rich representation of context. However, this work assumes the context is identified and encodes it into models for several scenarios without evaluating it to decide and select which elements are influential. Therefore, modeling context based on 3LConOnt given a particular scenario is not sufficiently effective or efficient for deciding the relevancy of context to certain functional elements at run-time. In addition, 3LConOnt did not provide capabilities to integrate the dynamic constraints on context such as acquisition types (Sensed, Profiled, Static, and Derived), temporal constraints (Fixed and Relative time intervals), and access types (Private, Restricted, Group, and All). Additionally, this work did not offer a checking capability for the integrity constraints on context.
Van Engelenburg et al. [26] proposed a logic programming-based method to understand and analyze the context for designing context-aware systems. This method can be integrated into the design process to specify the conceptual model for the context of a context-aware system, and it includes steps for gaining insight into the context, determining the components needed to sense the context and adapt to it, and determining the rules according to which the system should adapt. To represent context information, the following predicates are used to define the abstract syntax of a modeling language for this method: context rule, context relationship rule, environment element, situation, focus, context relationship, context element, and context. This method defines the context elements and context relationships using a set of context rules and context relationship rules, in which the context can be determined depending on the truth values of expressions about environment elements. The relevance can be expressed as a relationship between two components such as a user and provided information. It is important to recognize the difference between this method and our own approach; the work presented in our study is concerned with supporting the multi-view collection of contextual states, the evaluation of which are the most influential and interesting, and the selection of what should be encoded in context models when designing CSSs. We thus focus on expressing the influences on user-system interactions in a formal manner to discover what facts can be sensed, derived, and elicited, and what abstract states can trigger adaptive actions in a CSS at runtime. In addition, this method does not offer a tool to instantiate and validate context information in machine-processable artifacts. To manipulate and interpret the context, a set of concepts is used in this method (e.g., situation, focus, and adaptor elements). However, the proposed language does not provide terms to represent the dynamic aspect of context such as temporal constraints, permissions, acquisition types, and quality metrics.

Table 2 presents an overview of the comparative analysis of the related approaches considering several requirements in three sets. The rows in the first set (theoretical viewpoint) indicate how and to what extent each approach studies aspects of context; (1) **Multiple views**: the different perspectives of context (from multiple stakeholders) should be addressed, (2) **Relevance**: the influence relationship between the context and system-level elements should be addressed, (3) **Structures**: the structural aspect of context should be addressed to expressively represent context information, (4) **Behaviors**: the behavioral aspect of context should be addressed to appropriately specify the adaptation to the current context, and (5) **Dynamics**: the dynamic aspect of context should be addressed to manage changes in the context at run-time. The rows in the second set (conceptual viewpoint) indicate the modeling capabilities that were developed in each work (model); (i) **Elicitation capability**: the model should explicitly support collection of the context from various sources, (ii) **Identification capability**: the model should explicitly support evaluating and selecting the context elements relevant to a system, (iii) **Manipulation capability**: the model should explicitly support processing the features and properties of context, (iv) **Interpretation capability**: the model should explicitly support multi-level context fusing and inferring, and (v) **Validation capability**: the model should support checking the integrity constraints of the conveyed context information. The rows in the third set (practical viewpoint) indicate the technical methods and/or tools presented in each work to verify the applicability and usefulness of the proposed model; (a) **Design process**: a methodological process is required to follow systematically when modeling and managing the aspects of context for designing intelligent CSSs; (b) **Modeling language**: an expressive formal language (e.g., a metamodel, ontology, and/or logical formalism) is required for specifying an instance model of contextual variants reflecting a dynamic environment; and (c) **Automated support**: a computer-aided tool (e.g., a graphical editor, code
generation, or representation engine) is required to convey easily the analyzed data and elicited information, and transform them into a machine-processable format (i.e., a context specification) that can be used by a system to reflect the sensor data inputs, check changes in the context, and infer the actual situation to adapt accordingly at run-time. Based on the overview provided in Table 2, we conclude that the current approaches do not provide a comprehensive modeling strategy that meets all the identified criteria and requirements. Although the analyzed approaches addressed, to some extent, technical aspects related to the development of CSSs, they also missed solutions for tackling some issues, which could make the context management more efficient and the development process more effective. Most of the analyzed work did not propose treatments of the context, dedicated to different perspectives/times (i.e., representational/development time and interactional/runtime) [20]-[25], which support the system engineers with methods and tools to understand and evaluate the context before encoding and implementing it. Furthermore, it can be seen that none of the approaches have incorporated a multi-view context consolidation into the designed model to address different concerns about the context. In addition, most of the approaches extensively focused on addressing and expressing the structural aspect of context but they did not sufficiently deal with or specify conceptually the dynamic and behavioral aspects of context [20]-[22], [24]-[26].

We therefore present in the following sections our proposal for addressing the limitations identified above through a formal modeling framework for Context Identification and Management for intelligent Context-Sensitive Systems (CIM-CSS).

III. CIM-CSS: A PROPOSAL FOR FORMAL CONTEXT IDENTIFICATION AND MANAGEMENT FOR INTELLIGENT CONTEXT-SENSITIVE SYSTEMS

A. THE NEED FOR METAMODELING AND FORMALIZING CONTEXT IDENTIFICATION AND MANAGEMENT

It can be noted in the recent literature on context modeling that the context is treated as structures (i.e., classes or attributes) or as associations (i.e., links between two or more concepts). However, the context is a dynamic construct or process (i.e., the context is in a constant state of flux) shaped by the phenomena shared between a user and a system. We therefore attempted to conceptualize and metamodel the following CSS design activities: context identification, context manipulation, context interpretation, and context formalization, which can be used to simulate the dynamic flow of context throughout its life cycle from formation (i.e., a context relevant to a system originates from a phenomenon of interest) to dissemination (i.e., the relevant and interpreted context is published as unique states that can be consumed by different applications to adapt their logic accordingly). The advantages of such treatment are as follows. (1) A comprehensive understanding of the relationship between the system and the context, (2) a modular management of the features of context e.g., the acquisitions, the quality metrics, and the abstractions, and (3) an accurate and realistic representation of contextual information.

Here, we present the ‘context identification and management approach for intelligent context-sensitive systems’ (CIM-CSS), which is a formal modeling approach for identifying and managing context information in a context-sensitive system.

We consolidated key notions related to the concept context including Focus [43], Relevance [40], and Goal [31]. We used the context classifications defined in [42], [44]-[45]. To address the relevance issue and complexity problems identified above, we (i) developed a goal-driven, entity-centered method that assesses the contexts and determines which elements are most influential and should be represented in the context model, (ii) developed a metamodeling framework aligned with Model-Driven Architecture (MDA) [46] and the Meta-Object Facility (MOF) standard [47] to conceptualize and formalize the context identification and management in context-sensitive systems, and present a formal context metamodel, and (iii) introduced a formalization of context in a Backus–Naur Form (BNF) formula grammar to represent context information at different abstraction levels in first-order logic (FOL) with relational operators.

A methodological design process is required to allow intelligent context-sensitive applications to be systematically designed [48], and the CIM-CSS approach will assist software engineers in context identification and management activities, as shown in Figure 1. The main modeling concepts are depicted in four macro-activities: context identification, context manipulation, context interpretation, and context formalization.

1. Context Identification: this activity evaluates the behavioral and interactional relationships between the context and system-level elements to decide the most influential and interesting context elements for the system. Context identification activity consists of three micro-activities.

(i) Focus specification: in this activity, a set of goals or operations (Focal Element), entities of interest (Entity), and active agents (Agent), together with their roles, are elicited and specified through goal-driven analysis and functional scenarios to discover variations in users’ needs and/or required variations in software behaviors - the so-called system foci (System Focus). In other words, each system focus expresses a distinct (direct or indirect) user-system interaction that is context-dependent.

(ii) Context collection: given every focus specified in the previous activity, multiple stakeholders and system engineers/experts can elicit and compile a list of potential contexts based on their different perspectives e.g., a device expert captures low-level contexts and/or sensory information that can be generated by the technologies and devices available in a system.
(iii) Relevance Evaluation: in this activity, a system analyst uses comparative procedures to assess the domain particularities, aggregations, generalizations, and refinements of the collected contexts to decide which (domain-specific, aggregated, generalized, or refined) context elements are relevant to (Relevance) a given focus.

2. Context Manipulation: this activity handles the structural and dynamic aspects of context to process the features of context (e.g., sensed values, data properties, and dimensions: Temporal, Social, User, Computing, Activity, and Physical) and control the flow of context (e.g., temporal constraints and acquisition methods from sources: sensors, online profiles, and databases), respectively.

3. Context Interpretation: this activity analyzes the granularity/abstraction of the context elements (e.g., low-level visible, middle-level monitorable, or high-level abstract states) and the logical relationships between context elements (e.g., dependency, contradiction, and temporal sequence) to allow inferences to be made and support context usage at different levels.

4. Context Formalization: this activity specifies the interpreted contextual information from the previous activity in BNF formulae to make multi-level fusion and conclusions of context using FOL.

---

B. FORMAL METAMODELING FRAMEWORK FOR CONTEXT IDENTIFICATION AND MANAGEMENT

1) METAMODELING PROCESS

The main aim of this work is to develop a context modeling approach that can be used in different context-sensitive systems and various application domains. We therefore adapted a metamodeling creation process from FAML [49] as shown in Figure 2. To provide a rich and comprehensive modeling of the context, a set of adaptations to the FAML methodology of metamodeling is added as follows: (1) inclusion of step 0 for the collection of context definitions and models, (2) inclusion of the MDA metamodeling mechanisms and tasks; generalization, specialization, aggregation, and attribution of concepts, in step 5, and (3) inclusion of step 6 for the identification of constraints on concepts and relationships.

We used the 7-step metamodeling process as a guide to establish and synthesize our metamodel for context identification and management. The primary objectives of the steps are described below.

Step 0: Collection of context definitions and models that facilitate understanding, representing, and managing the context and supporting the design of context-sensitive systems. We identified important definitions and models for the context from previous surveys on context modeling and metamodeling [19], [50]-[52].

Step 1: Classification of context definitions and models into sets according to their perspective on context (e.g., representational, interactional, and operational perspectives).

Step 2: Extraction of notions and concepts associated with context-awareness and context management in each set created in step 1.

Step 3: Identification of the concepts commonly used in the definitions and models. These concepts are important to express because they appeared in most of the analyzed definitions and models.

Step 4: Classification of the concepts selected in step 3 into three categories: context identification (e.g., contextual, structural, intentional, and functional elements), context manipulation (e.g., sources of context, dimensions of context, and quality metrics), and context interpretation (e.g., context relationships and context abstractions).

Step 5: Identification of conceptual relations among the selected concepts and of the attributes of each concept.

Step 6: Identification of integrity and consistency constraints on selected concepts and relationships.

Step 7: Creation of the context metamodel based on steps 4, 5, and 6. A clear specification of the conceptual model for the context will be described in the following sections.

---

FIGURE 1. CIM-CSS: design activities and concepts

FIGURE 2. Metamodeling Process
2) FOUR-LAYERED METAMODELING ARCHITECTURE

Model-Driven Architecture (MDA) is a promising software development paradigm that provides metamodeling and model transformation techniques for building platform-independent models (PIM) and transforming them into platform-specific models (PSM) and generating code e.g., Java/EJB, C#/.NET, and XML/SOAP [53]. The metamodel is a model that raises the abstraction level for developers by defining high-level concepts in terms of which other artifacts are constructed [54]. MDA is supported by the Object Management Group (OMG) Meta Object Facility (MOF) standard [47], which provides metamodeling architecture based on four levels: (i) a meta-metamodel level defining a highly abstract language for specifying metamodels; (ii) a meta-model level defining a more elaborate language for specifying models; (iii) a domain model level defining a language for describing a specific domain; and (iv) an instance data level defining concrete domain information. The architecture of our modeling framework is based on four abstraction layers and was aligned with the MOF standard as described below and shown in Figure 3.

M3 (meta-metamodel layer) is the meta-language within which the highest-level abstractions are defined for structuring our context metamodel and language. The MOF meta-metamodel contains object-oriented constructs such as MetaClass, MetaProperty and MetaAssociation. In our modeling framework, the meta elements are Meta-Concept, Meta-Relationship, Meta-Attribute, and Meta-Constraint, which represent the components of a modeling language supporting the CIM-CSS.

M2 (metamodel layer) is the language within which domain-independent abstractions are defined for structuring formal specifications from multiple aspects of the system (e.g., the contextual, structural, intentional, and functional, views). In this layer, the CIM-CSS modeling language contains the meta-level constructs: (1) meta-concepts (e.g., Context Element, Entity, Goal, Operation, and Agent), (2) meta-relationships between meta-concepts (e.g., Relevance, Characterization, and Role), (3) meta-attributes of meta-concepts and meta-relationships (e.g., description of Context Element, and level of Relevance), and (4) meta-constraints on meta-concepts and meta-relationships (e.g., EntityKey meta-constraint on Entity meta-concept).

M1 (domain model layer) contains concepts specific to a modeled domain or organization e.g., the light controller, room, and participant are specific to the smart meeting room domain. It contains instances of the meta-level abstractions. For example, the MeetingRoom concept in Figure 3 is an instance of the Entity meta-concept, the HostsMeetingNow concept is an instance of the Context Element meta-concept, and the concept Achieve[LightsSwitchedOn] is an instance of the Goal meta-concept. The instantiations of the meta-relationships are links between the instances of the meta-concepts. For example, characterizes is an instance of the relationship between the context element HostsMeetingNow and the entity MeetingRoom, and isRelevanTo is an instance of the relationship between the context element HostsMeetingNow and the goal Achieve[LightsSwitchedOn].

CIM-CSS can be used to build context diagrams to model aspects of the context at the design time. A context diagram is structured from instances of domain-specific concepts, relationships, attributes, and constraints. The domain-specific concepts and relationships must satisfy their constraints or well-formedness rules (i.e., instances of the meta-constraints).

M0 (data model layer) contains concrete instances of domain-specific concepts in the running application. This level contains sensed data; obtained information; observed entities, places, devices, and people; requirements; software and hardware components. The data elements can be acquired and represented as run-time entities [30]. Instantiations of the design-time diagrams can be implemented and maintained in models reflecting the highly dynamic and continuously changing context using Models@Run. Time to reason on the data and adapt the system behavior accordingly [55].

![FIGURE 3. Four-layered architecture for metamodeling context identification and management](image-url)
3) CIM METAMODEL SPECIFICATION
As mentioned above, the meta-metamodel layer M3 contains meta-objects and is used to construct the metamodel level M2. Formal methods supporting object orientation and reuse are therefore appropriate to formalize the context and associated concepts. We defined the abstract syntax of our metamodel for context identification and management (CIM) using the UML class diagram [56] and specified its integrity constraints using the formal expression language OCL [57].

A common technique for specifying a modeling language is to define the syntax of the language and describe its static semantics. The abstract syntax for our context modeling language is presented using UML class diagrams in Figures 5, 6, and 7, to illustrate the meta-concepts, meta-attributes, and meta-relationships. Each meta-concept is represented as a class. The meta-relationships are shown as association classes. The meta-attributes are shown as attributes of classes and association classes. The diagrams also show some of the meta-constraints, particularly the multiplicity constraints of the meta-relationships that apply cardinality restrictions on instances of the meta-concepts in the context model. We defined the semantics of the meta-concepts and the meta-relationships and their meta-attributes in natural language. The association ends have names, therefore OCL expressions can be specified for the corresponding meta-elements to provide a well-formed description of the metamodel. The metamodel was designed and validated using the Papyrus tool [58].

4) METAMODELING OF CONTEXT IDENTIFICATION
A goal-driven, entity-centered method for context identification is described below. A collection of meta-elements for formalizing the identification activity based on this method is then presented. The context identification method was developed using a design science research (DSR) approach [59]. A DSR was conducted with the problem at hand (i.e., the relevance of context), which is important to CSS designers. The need for such a method is established as follows. In practice, there is a need for this method when designing a CSS in various domains (see Section I). We then reviewed the relevant previous work for an appropriate method. However, the relevant work did not offer such a method (see Section II). Besides, existing design methodologies lacked a preliminary activity and expressive technique for identifying and modeling the context, respectively (originating from system requirements) [2], [3].

First, we started building the method by using primary notions from goal-oriented requirement engineering (GORE) techniques [31] that help (i) elicit high-level, strategic objectives and business goals for the system or organization being considered, (ii) refine the elicited goals to be realized by technical requirements, (iii) assign roles to system agents, and (iv) operationalize the requirements through human and environmental tasks and/or software operations. Furthermore, the Focus [43] and Relevance [40] notions allow the context to be assessed and identified at an early stage of the design process. In other words, the context elements that should be modeled later on are those parameters that affect the system or software requirements. Second, we developed a graphical modeling tool to support the use of this method (see Section IV). Third, we evaluated the method in practice by applying it when designing a CSS for a smart meeting room, and discussing the results qualitatively in Section V.

The context identification method is shown in Figure 4 and described below. It is composed of three micro-activities. The system experts and human in the loop involved in the activities can be supported with the automated tool and the conceptual model for the context as shown in Figures 5 and 6. Activity 1: Focus Specification. The requirements engineer uses input requirements models from which a list of system foci (Focus) can be specified in terms of the dimensions of the phenomena affecting both the user’s world and the software solution. More specifically, a focus item (F_i) combines pieces of information from the structural aspect, the intentional/requirement aspect or the functional aspect, and the responsibility aspect of the system. The first is intended to explore passive concepts (Entity) in domain knowledge i.e., E_i indicates an entity of interest. The second is intended to explore the user’s intentions/system requirements or the software services (Focal Element) i.e., G_i indicates a focal goal that concerns the entity and O_i indicates a focal operation that affects the entity. The third is intended to explore active concepts (Agent) in the system i.e., A_i indicates an agent with a role (Role), which is responsible for the focal element (G_i or O_i) indicated by F_E_i.

Activity 2: Context Collection. The experts take the list of foci and pursue iterative sub-activities for compiling context information (C) from three engineering standpoints (experts’ perceptions of context), each of which has particular purposes and addresses certain concerns about the context.

(i) Technological viewpoint: Given an F_i, the device expert conducts a bottom-up elicitation to determine potential, associated visible and inferred facts that can be recognized through sensors (Device) available in a system e.g., a low-level data “70-decibel sound” acquired from a microphone sensor makes a room noisy. The results of this activity embody technology-generated contexts (i.e., hard data [60]).

(ii) Functional viewpoint: Given an F_i, the CSS designer conducts a top-down elicitation to determine contexts that are of interest to a system (Application), in which adaptations are required to meet changing needs at run-time e.g., a situational context like “a user is resting in the living room now.” is important to an entertainment application in a smart home, in which a favorite program on TV can be shown. The results embody application-oriented (operational) contexts.

(iii) Cognitive viewpoint: A domain or organization is characterized by a set of environmental elements (Place, Object, and Person) and users (User). Given an F_i, the domain expert and end-user observe, decide, and elicit facts about the particularities of domain entities [61] e.g., preferences, social contexts, or spatiotemporal correlations. A domain ontology is useful for supporting observations and provides a rationale for decision processes. The results embody human-contributed contexts (i.e., soft data [60]).
Activity 3: Relevance Evaluation. A CSS analyst conducts further analysis (e.g., abstraction, refinement, aggregation, and assessment) of the collected contexts and establishes comparative procedures to determine the relevant elements according to their impact and relation/relevance to a given focus, indicated by impactOn() and Rel(), respectively. The level of relevance indicated by levelOf(Rel()) can take the value High, Medium, or Low. The impact of a context on a focus can be mutual; the impact of an activated focus on a context indicated by impactOn(F,C) can take the values Changed and Unchanged, while the impact of the context on a focus indicated by impactOn(C,F) can take the values Activation, Required, Quality, or None. For example, in some observable state (Cj) it will be necessary to fulfill the focal element FEj that concerns the entity Ei by the agent Ai. Therefore, this state acts as an activation context on the focus Fi at run-time; impactOn(Cj,Fi) = Activation and levelOf(Rel(Cj,Fi)) = High. The conclusion is then that Cj is relevant to and contextualizes Fi. This also applies on the so-called quality context, which may affect positive or negative contributions to the quality of the system goals. When an Fi is activated, the system needs to trigger an adaptive (rule) action to accommodate the current context (Cj) i.e., a condition holds. This actuation, in turn, might lead to a change of another state (Ck) of the system world i.e., impactOn(Fk,Ck) = Changed. In addition, other states can be less important or a (required) pre-condition for some focus. In other words, a system may need to accumulate more information before making a decision or performing an operation in some states, but it needs to react immediately in certain other states. We will investigate the effects of other types of context on different system levels, e.g., the effects of activation and quality contexts at the goal level [28].

We will also investigate (supervised) machine learning techniques for supplementing the identification method with an automated relevance algorithm for big data and IoT-based applications that involve large volumes of sensory data, inspired by filtering algorithms (e.g., [62]).
The meta-elements for formalizing the identification of the context are shown in Figure 5 and described below.

**Goal, Role, Agent, and Operation** are meta-concepts from a goal-oriented RE methodology called KAOS [31]. These meta-concepts are used for identifying contexts relevant to system foci. Human agents wish high-level goals to be satisfied and can perform tasks in a system. Devices (e.g., sensors or actuators) and software components can be assigned low-level goals (i.e., the so-called expectations and requirements) and perform environmental and software operations, respectively [31]. Therefore, the **Focal Element** meta-concept is introduced as a superclass of the classes **Goal** and **Operation**. The **mode** specifies the occurrence mode (i.e., a way a focal element may occur or is performed). The Focal Element **mode** can have the values **Meaningful** and **Incidental**. The **priority** specifies the degree to which a focal element is important or preferred. The Focal Element **priority** can take the values **Normal**, **Important**, and **Critical**. The **Entity** describes a domain-specific concept related to the system goals and can be affected by the system operations. The **Entity type** can have the values **Person**, **Object**, **Device**, **Place**, **User**, and **Application**. The **Focus** meta-concept is a composite of a focal element, a system agent responsible for the focal element and its role, and an entity related to or affected by the focal element. The **Relevance** is an influence relationship between the **Focus** and **Context Element**, and helps identify relevant contexts by measuring the type and pertinence to the given focus. The **weight** specifies the relevance level. The **Relevance weight** can have the values **High**, **Medium**, and **Low**. The **argument** specifies the rationale supporting the relevance of a context element to the given focus. For example, an argument may state that a certain context activates, is required by, or affects the quality of a given goal. It may state that the performance of a given operation changes a particular context.

When a context is identified and recognized by an intelligent system, it should make a decision on whether to adapt or to collect more contexts. Therefore, a focus may activate a set of rules (**Rule**). Each rule is composed of a set of conditions (**Condition**) and actions (**Action**). To handle the adaptive behavior of the system, a set of actions (e.g., notifications, information adaptations, business adaptations, presentation adaptations, and access controls) can be executed dynamically to accommodate changes in the context.

**5) METAMODELING OF CONTEXT MANIPULATION**

The meta-elements shown in Figure 6 and described below are used to formalize the features required to manipulate the context, from acquisition to dissemination.

First, the meta-elements that define the constituent parts of the context are indicated by **Context Element** and **Property**. The **Context Element** captures all atomic and composite data, information, knowledge, and wisdom considered relevant to the focus of a system, and characterizes the state of an entity related to the focus in the modeled domain. The **description** meta-attribute defines, in natural language, what the context prescribes in terms of the states of the world. The **formalSpecification** meta-attribute provides a formal statement of the context. The **Property** captures low-level data.

The **name** is a unique identifier of the contextual property (e.g., light level, noise level, or location). The **Property** can have values within a range. These values can be multiple or fixed or vary temporarily, and are imposed by access restrictions or permissions (e.g., **Private**, **Restricted**, **Group**, or **All**). Multiplicity type, rigidity type, and permission type values respectively are indicated by the **multiplicity**, **rigidity**, and **access** meta-attributes of the **ValuesIn** meta-relationship associating the **Property** with the **Range** meta-concept. The **HistoryProperty** captures the awareness of historical values through the **timestamp** meta-attribute.

Second, the meta-elements that define the acquisition of context. The **Context Source** indicates the sources of the context such as physical sensors (RFID tag, microphone, or GPS), logical/soft sensors (e.g., digital clock, online profiles, or databases). The **Acquisition** meta-relationship indicates a contextual association between the **Context Element** and **Context Source**. The **updateFrequency** meta-attribute specifies the temporal rate at which the context data are acquired (Never, Occasionally, Frequently, or Continually). **Acquisition** has four sub-classes (**Sensed**, **Derived**, **Profiled**, and **Static**) to represent the possible contextual associations [44]. The **Derived** indicates dynamic associations with contexts calculated or derived from other contextual information. The **expression** specifies the derivation rule. The **Profiled** indicates dynamic associations with contexts supplied by the user (e.g., preference contexts). The **Static** indicates static associations with high constant information (e.g., identity contexts in terms of date of birth or gender). The **Sensed** indicates dynamic associations with contexts acquired through sensors.

Third, the **Quality** meta-element defines the quality of context (QoC). The **parameter** meta-attribute indicates the type of quality metric (e.g., freshness, degree of credibility, probability of correctness, or precision).

Fourth, the categories of context are indicated by the **Atomic** meta-concept and its sub-classes. The **Atomic** indicates a single (dimensional) context [42], e.g., (**Cognitive**, **Social**, **Physical**, **Temporal**, **Computing**, **Activity**, **User**, or **Domain** context [45]). The **User** context is divided into **Identify** and **Preference**. The **Temporal** context has three sub-classes: **Past**, **Current** and **Future**. **Location** is a sub-class of the **Physical** context.

Finally, there are some meta-elements that indicate the **Validity** of context (e.g., **Time Constraint**, which indicates a valid time for the context to be used). The **Fixed** sub-class indicates a fixed time interval. The **Relative** sub-class indicates a relative time interval.

**6) METAMODELING OF CONTEXT INTERPRETATION**

The meta-elements shown in Figure 6 and described below are used to formalize the features required to interpret the context at the system level.

First, the meta-elements that define the granularity of context are the **isAbstract**, **isMonitorable**, and **isFactual** meta-attributes of the **Context Element**. These indicate three different levels of context abstraction, from the highest to the lowest. The **isAbstract** meta-attribute specifies whether the nature of the context is abstract or whether any information to verify the truth value of the context is lacking.
FIGURE 6. CIM meta-elements for context manipulation and interpretation.

The isMonitorable meta-attribute indicates whether the context can be supported by visible facts. The isFactual meta-attribute indicates whether the context is a visible fact. The isVerifiable meta-attribute indicates the verifiability of context at run-time (i.e., whether the truth value of the contextual element can be verified when the system is running).

An abstract context may be supported by a set of factual and monitorable contexts through further analysis and refinement. The situation-aggregated many-to-many self-reference association for ContextElement indicates high-level abstractions of the context and aggregates lower-level contexts that are logically interrelated to allow different interpretations of the situation of an entity to be made i.e., situational contexts, in which different applications may have interest. The situational context characterizing an entity is indicated by the name meta-attribute of the Characterization meta-relationship. The Composite meta-concept indicates a compound context.

Second, the Relationship meta-element indicates relations among contexts. The Relationship meta-element indicates an association between a source context and a target context. The name identifies the relationship. The Relationship meta-element has the sub-types Refinement, Causal, Parallel, and Conflict. The Support meta-element indicates a special kind of refinement between contexts (e.g., a monitorable context and visible facts [28]). The Causal, Parallel, and Conflict meta-element indicate implications between contexts, independent contexts, and contradictory contexts, respectively. A context use mechanism must specify the adaptive behavior of the system to allow the interpreted context to be implemented and provide smart interactions between the user and the system, which accommodate user’s changing needs and situations. A rule-based strategy can therefore be used as indicated by the Rule, Condition and Action meta-elements as shown in Figure 5.

FIGURE 7. Main constituents of the CIM metamodel
To this end, the formal constructs presented above can be instantiated in a context model to encode various data and information (including context elements, sources of contexts, domain entities, and temporal constraints), contextual associations (such as acquisition, characterization, validity, and context relationships), and other constituents of the context model, as shown in Figure 7. In addition, we emphasize the importance of the focus class and relevance association, which offer two benefits: (1) the ability to filter and select the contexts that most strongly affect the system, and (2) the ability to use relevant contexts to support software adaptability through taking context-dependent actions such as notifications, business adaptations, presentation adaptations, information adaptations, and access controls, as shown in Figure 5.

7) CIM METAMODEL INTEGRITY CONSTRAINTS

UML uses OCL [57] to specify well-formedness rules for a class model. Well-formedness rules are integrity constraints that the modeler should apply to the classes. In our formal CIM metamodel, we use class invariants that are constraints that must be true for all well-formed instances of the corresponding constructs. An OCL expression that specifies an invariant on a formal construct usually has the following pattern:

context Construct_Name invariant Constraint_Name:
‘This is an OCL expression with stereotype <invariant> in the context of Construct_Name’

The context definition of an OCL expression specifies the meta-element for which the OCL expression is defined. In our formal metamodel, this is a class or an association class defined in the class models depicted in Figures 5, 6, and 7. Some well-formedness rules expressed in OCL are defined in Figure 8. All integrity constraints were implemented in the metamodel and checked using the Papyrus tool [58].

The EntityKey, SituationKey, PropertyKey, and ContextKey are primary key constraints for the Entity, Characterization, ContextElement, and ContextModel, respectively. The Verifiability constraint states that the truth value for facts and monitorable contexts can be verified at run-time. The Granularity constraint ensures that the finest-grained context is a fact that must have at least one contextual property, contrasting with the high-level abstract context. The EveryContextIsEmbeddedInTheTime constraint establishes that any composite context is a compound of at least two atomic contexts and that one is temporal. The ContextIsNotRelatedWithItself constraint ensures that a relationship between a context and itself cannot be defined. The ContextIsMonitorableIfSupportedByFacts states that a context is monitorable if and only if it can be specified by a set of facts supporting it. The AcquiredContextFromSourceIsFact ensures that all contexts acquired through a context source are facts. The ContextMustBeRelatedToFocus constraint ensures that ‘for every context element ce characterizing an entity e in the management view there is a focus f in the identification view, linked to ce and e by a relevance relationship r’. The last invariant is an inter-view consistency rule for compatibility and complementarity between the identification and management views of the context. In other words, the identification and management views of a context model are structurally consistent if they meet the invariant constraining their elements.

C. CONTEXT FORMALIZATION

The CIM metamodel presented in the previous section provides a formal language for modeling context information. This metamodel defines the context at three levels of abstraction using, (i) a context property indicating the lowest level of sensed data; (ii) an atomic context raising the abstraction of context property to define a semantic context type; and (iii) a composite context combining atomic contexts and/or other composite contexts to define a higher-level situational context. Here, we present a formalization of context in first-order logic (FOL) to make inferences about the context and derive implicit contexts. Due to its expressiveness, generality, and capability of performing different types of logic reasoning, FOL was selected as a language for formally specifying the context information at different abstraction levels. The context formalism indicates atomic and composite contexts in the atomic and composite formulae, respectively, as depicted in Figure 9 and defined below.

Definition 1. (Atomic Formula)
An atomic formula is a world predicate (predicate) or function (function) applied to a tuple of terms (term) on domain-specific entities (entity) or context properties (property). The data types for values (value) and variables (variable) are Float, Integer, String, and Boolean. The relational operators (operator) among terms can be <, >, ≤, ≥, =, or ≠.

Definition 2. (Composite Formula)
A composite formula is a combination of atomic formulae through the logical connectives ¬ ‘not’, ∧ ‘and’, ∨ ‘or’, → ‘implies’, and ↔ ‘equivalent to’, that can be prefixed by universal (∀) or existential (∃) quantifiers.

![FIGURE 8. Integrity constraints for the meta-elements shown in Figures 5, 6, and 7, expressed in OCL](image-url)
Definition 3. (Context Relationship)
A context relationship is a logical link between two or more contexts. The support relationship is mapped to the logical connector ‘→’ (implies). The translation of the support relationship is defined as below.

Let $S$ be the set of support relationships in a context model, $I$ is an instantiation of the model (i.e., instances of context elements and instances of context relationships), $I(s)$ represents the set of tuples in $I$ belonging to the support relationship $s \in S$, and $C$ is the set of instances of context elements in the model. Then, an assertion of the form $\varphi \rightarrow c_d$ in the definition $s(c_1,\ldots,c_n) \equiv \varphi \rightarrow c_d$ (where $s \in S$, each $c_i \in C$ for $1 \leq i \leq n$, and $\varphi$ is a composite formula combines $c_1,\ldots,c_n$ using the logical connectives $\land$, $\lor$, and $\neg$) will be true for $I$ if there is a tuple $<c_1,\ldots,c_n>$ in $I(s)$ where $c_1,\ldots,c_{n-1}$ are refinements or sub-elements of $c_n$, and false otherwise.

According to the above definitions, a BNF formula grammar for specifying the context in FOL with relational operators was introduced as shown in Figure 9. The ‘<w>’ term in the definition meta-language means ‘any instance of the syntactic category w’. The ‘: =’ term is the definition meta-symbol. The parentheses ‘( ‘ and ‘)’ can be used to change the precedence. The precedence order for the operators is specified, from the highest to the lowest, as ‘\land’, ‘\lor’, ‘\Rightarrow’, ‘\geq’, ‘\leq’, ‘\neg’, ‘\land’, ‘\lor’, ‘\Rightarrow’. The ‘|’ indicates an alternative choice. The ‘*’ meta-symbol means ‘0, 1, or more occurrences of’. Any atomic formula is a well-formed formula. Any composite formula is a well-formed formula. Nothing else is a well-formed formula.

![FIGURE 9. A BNF formula grammar in first-order logic for formalizing the context](image)

IV. MODELING TOOL DEVELOPMENT

In this section, we outline the process through which an Eclipse-based graphical editor (a tool support for the CIM-CSS approach) was developed. The tool can be downloaded from [https://github.com/metamodeler/CIM-CSS/](https://github.com/metamodeler/CIM-CSS/).

The main frameworks used to build the tool were the Eclipse Modeling Framework (EMF) [63] and the Graphical Modeling Framework (GMF) [64]. A model-driven development (MDD) approach [65] based on the EMF and GMF was used to generate automatically the code and to develop the tool. The formal CIM metamodel (CIM3) was used to construct a domain model of the tool. The development steps followed to implement the graphical concrete syntax of the tool are shown in Figure 10 and outlined below.

Step 1: Create a domain model.

In the first step, the context identification and management metamodel (CIM3) was used as a foundation on which to create a domain model using the EMF. We loaded the metamodel to create the CIM3.ecore domain model in XMI specification [66]. This Ecore model contained the platform independent model (PIM) information.

Step 2: Derive an EMF generator model.

The CIM3.genmodel EMF generator model was derived from the Ecore model using the GMF dashboard. This model contained the platform specific model (PSM) information and code generation options for the context model domain elements.

Step 3: Derive a tooling definition model.

The CIM3.gmftool tooling definition model was derived from the Ecore model using the GMF dashboard. This model defined the elements of the diagram palette, as shown in the tool bar in Figure 11.

Step 4: Derive a graphical definition model.

The CIM3.gmfgraph graphical definition model was derived from the Ecore model using the GMF dashboard. This model defined the graphical elements (e.g., nodes, labels, and connectors), as shown in the workspace in Figure 11.

Step 5: Create a mapping model.

The CIM3.gmfmap mapping model was created by combining the previous three models. This model linked the domain elements from the CIM3.ecore model to their graphical representations in the CIM3.gmfgraph model and assigned them to the palette in the CIM3.gmftool tooling model.

Step 6: Create a diagram editor generator model.

The CIM3.gmfgen generator model was created using the GMF dashboard. This model was composed of customizable elements for modeling the context in the identification and management activities. The diagram editor code was generated using the model-to-text (M2T) transformation of the CIM3.gmfgen model. The model-driven steps created a list of Eclipse plug-ins necessary to run the diagram editor and use it as a modeling tool. The tool is comprised of the components: title bar, property bar, workspace, and tool bar as shown in Figure 11. The tool was built using the EMF, GMF and MDA technologies, starting by transforming the designed CIM metamodel to the Ecore model and then performing model-to-model and model-to-text transformations.
Figure 11 shows the graphical element distributions in the automated support tool. On the title bar both the names of the application and the current working context diagram are displayed. In the property bar the modeler can view and edit the properties (i.e., attributes and constituents) of the context identification and management elements on the workspace. The properties are different according to the selected element; however, all context identification and management elements have some common properties such as name, type, and value. The context identification and management elements are organized in the following categories in the tool bar: Abstractions, Relationships, Constraints, Domain level, and Instance level.

V. EVALUATION

In this section we evaluate the applicability and effectiveness of our approach by performing a proof-of-concept using a smart meeting room use case and conducting a qualitative analysis.

A. A SMART MEETING ROOM SCENARIO

The applicability of our approach to context identification and management was tested using an application scenario for a smart lightening system for meeting rooms as a proof-of-concept. Providing an intelligent lighting application in the cognitive domain of meeting rooms is one of the top preferred features, especially for light automation and energy saving. An informative document or ontology that captures the generic domain knowledge of smart meeting rooms (see e.g., [67] for a survey) can be used to support decision processes and modeling activates for building a rich context model. The following activities were carried out on the scenario along with using the CIM-CSS modeling language and tool for the elicitation and modeling of context information.

(i) Context Identification.

In this activity, our proposed context identification method was applied following its micro-activities as described below.

(ii) Focus Specification. A set of system foci was specified as shown in Table 3. First, a lighting tuning goal (G1) was elicited and specified, as “The system shall automate the adjusting and control of lighting in a meeting room [r] based on the meeting participant’s needs and energy saving”. This goal was refined to be realized by four functional requirements (G1.Req1, G1.Req2, G1.Req3, and G1.Req4) that concern the entity meeting room [r]. These requirements were classified as behavioral (Achieve) goals [31], and their responsibilities were given to a smart dimmer agent with different roles (R1, R2, R3, and R4). The respective foci were defined in F1-F4.

TABLE 3. System foci in the smart meeting room domain

<table>
<thead>
<tr>
<th>Focal Goal</th>
<th>Agent</th>
<th>Role</th>
<th>Entity</th>
</tr>
</thead>
<tbody>
<tr>
<td>F1 G1.Req1: AchieveDimming the light auto-adjusted</td>
<td>A1: smart dimmer</td>
<td>R1: dim</td>
<td>E1: meeting room [r]</td>
</tr>
</tbody>
</table>

FIGURE 11. Eclipse-based graphical tool for the CIM-CSS

(ii) Context Collection. Given the elements in F1-F4, the system experts elicited a preliminary set of potential contexts; C1= room occupancy, C2= lighting, C3= time, and C4= noise.

(iii) Relevance Evaluation. The context elements that are influential in adapting the system behavior were determined as follows. The context elements CE1 = “The room [r] is empty”, CE2 = “The light level is high”, and CE3 = “The current time is 10:05 a.m.” are relevant to F4 as the aggregated context activates Req4 to turn off the light for energy-saving power mode in the final situational context, say SC1 = “There is no meeting activity in the room [r]”, whereas the context elements CE1’ = “A person [p1] enters the room [r]”, CE2’ = “The light is very low”, and CE3’ = “The current time is 25. a.m.” are relevant to F3 as the newly aggregated context acts in the requirement Req3 to turn on the light in the final context, say SC1’ = “The meeting is about to start”. The context elements CE1’’ = “At least two persons are located in the room [r]”, CE2’’ = “The current light level is high”, CE3’’ = “The current time is 10:30 a.m.”, and CE4’’ = “The room [r] is very noisy” are relevant to F1 as the aggregated context activates Req1 to dim the light for meeting presentation mode in the context, say SC1” = “The room [r] hosts meeting now”, whereas the context elements CE1” = “At least two persons are located in the room [r]”, CE2” = “The light level is low”, CE3” = “The current time is 12:25 p.m.”, and CE4” = “The noise level is low in the room [r]” are relevant to F2 as the newly aggregated context activates Req2 to brighten the light in the final context, say SC1”’ = “The meeting is about to end”.

(iv) Context Modeling. An excerpt of the context model for the exposed scenario designed using the CIM-CSS tool is shown in Figure 12. The tool is enriched with a context modeling language to help the designer to select the elements associated with identifying and managing the context. The graphical notation supported the modeling of context elements (e.g., CE1”, CE2”, CE3”, and CE4”) that are relevant to F1, as indicated by the contextual variant (F1-SC1”-CV1).

(2) Context Management.

In this activity, the identified context information was handled through (i) context manipulation that specifies the core constituents of the contexts e.g., sources, properties, and dimensions, and (ii) context interpretation that specifies the abstractions and relationships between the contexts.
(i) Context Manipulation and Modeling.

The context modeling tool allowed capturing and modeling the sources of identified contexts (e.g., camera, light sensor/LDR, digital clock, and microphone), the types of sensors (e.g., Physical, and Logical), the contextual properties (e.g., location, light level, noise level, and time), the contextual dimensions (e.g., Physical, Location, and Temporal), and the acquisition methods (e.g., sensedFrom, derivedFrom, and profiledln) as shown in Figure 12. Quality metrics and time constraints can be elicited and linked to their elements in the model by applying respective design iterations.

(ii) Context Interpretation and Modeling.

The tool supported the modeling of identified contexts as atomic contexts in the diagram, the links to their fine-grained data properties (e.g., valuesIn and composedOf), coarse-grained abstractions (e.g., composite context), and dependencies between the contexts (e.g., AND refinement and causal). Moreover, Figure 12 shows different sub-variants and multi-sensor fusion that imply the same situational context F1-SC1 by using the “support” association and “AND” and “OR” nodes, linking the root context with its contextual variants such as CV2 (activity, start time, and place name contexts that are profiled in a participant’s diary entry, mail, or calendar i.e., soft sensors), CV3 (motion, weight, heat, and/or humidity contexts that are sensed from physical sensors in a meeting room), and CV4 (temperature and location contexts that are sensed from a thermostat and the smart phone of a participant, respectively). Not only can this mechanism allow the designer to consider and model varying contexts in which the system will operate, it can also allow the designer to specify multiple fusion and different interpretations that can be acquired through various sources and smart objects. The resulting context model can be saved in a database for query at run-time in order to reflect sensor data recognition and perceive the current situation in which the system needs to adapt its functionalities. Indeed, the instantiations of the CIM-CSS language can provide more expressiveness and extensive semantics for modeling context information than UML class diagrams, as shown in the right-hand part of Figure 12.

(3) Context Formalization.

Context information was encoded in a formal description to allow new contexts to be inferred when the system is running. The contexts elements were mapped to formal specifications using the BNF syntax, and the context facts at the bottom of Figure 12 were mapped to atomic formulae that could be verified at run-time. The elements (i.e., situational contexts) at the top of the context diagram were mapped to composite formulae derived from the atomic formulae stated formally below.

- $\exists p1, p2: \text{Person}, r: \text{Meeting Room}$
  - locatedAt(p1,r) $\land$ locatedAt(p2,r) $\land$ noiseLevel = ’high’ $\land$
  - time = ’10:30’ $\land$ lightLevel = ’high’ $\rightarrow$ hostsMeetingNow(r)
  - $\land$ inMeetingNow(p1) $\land$ inMeetingNow(p2)
- $\forall r: \text{Meeting Room}$
  - HostsMeetingNow(r) $\rightarrow$ Occupied(r)

(4) Context Usage.

This step specified the use of the formalized contexts to adapt the system behavior using a rule-based mechanism. In the first statement above, the contexts inMeetingNow(p1) and inMeetingNow(p2), named as SC2 and SC3, indicated the current activity contexts of person[p1] and person[p2], respectively. From the perspective of a smart room system, the inferred situational contexts are more interesting than the raw data or other atomic pieces of information because it can be used to trigger personalized actions e.g., “sending meeting materials and agenda to the participants’ mobile devices”.

In the second statement, the inferred context Occupied(r), named as SC4, indicated the situational context of the meeting room[r], which can also be used as a relevant condition to trigger adaptive rules at the level of a smart building system e.g., “adjusting the heating and cooling services as more participants enter a meeting room at real-time”. As illustrated by this scenario, the proposed integrated formal modeling method is applicable to context information at different abstraction levels. The model allowed the context and its variations to be better understood and managed, and supported the identification of the situations in which the system should adapt its behavior to changing users’ needs (originating from system requirements).
B. Qualitative Analysis

In literature on the software engineering and systems engineering field e.g., [68] it has been stated that engineering models have five characteristics, namely abstraction, understandability, accuracy, predictiveness, and cost. Modeling and metamodeling offer three advantages, an unambiguous representation of the system of interest, a base for system analysis, and reusability [69]. In the light of the proof-of-concept presented above, we will analyze the effectiveness of the proposed CIM-CSS to ensure the context modeling language possesses the characteristics mentioned above and offers advantages to software engineers and other stakeholders when analyzing and specifying the system context, as described below. Abstraction: CIM-CSS allows different types of contexts at different levels of the modeled system (low-level sensed parameters, dimensional contexts, and situational contexts) to be represented. Using CIM-CSS, it is possible to define the context sources, the acquisition methods (sensed, derived, or profiled), and the relationships between contexts (refinement or causal). CIM-CSS therefore allows the complexity of the context aspect to be addressed effectively. Understandability: The CIM-CSS palette allows contexts to be expressed visually to facilitate the heterogeneity of the modeled data to be conveyed with little intellectual direction. Accuracy: The CIM-CSS tool allows designed models to be validated using the integrity, consistency, and completeness constraints imposed on the metamodel to ensure the modeled system context is represented accurately. Predictiveness: Integrating formal analysis methods into CIM-CSS makes it possible to infer future contexts from current observable states and profiled states and from historical contexts. Cost: CIM-CSS allows the context to be specified in a model that is inexpensive to construct and maintain because the original context diagrams can easily be exported, stored, and reasoned on in a model repository for the EMF, which supports the purpose of Models@Run.Time, e.g., XML Metadata Interchange (XMI) [66], EMFStore, and Connected Data Objects (CDO) [70]. It is possible to generate Java code embedding the context specification automatically using model-to-model and model-to-text transformations [53]. Unambiguity: The semantics of the CIM-CSS language allows the context to be specified in a more precise and explicit manner than general purpose modeling languages (e.g., class diagrams). This allows system engineers and user representatives to achieve a common and organized understanding of the aspects of context. A base for system analysis: CIM-CSS is driven by system goals and is focused on domain entities, so it allows the gap between modeling the problem domain and the interpretation of system and software requirements to be mitigated. It is therefore possible to use CIM-CSS as a basis for analyzing context-dependent system variants and to make choices between design alternatives. Reuse: CIM-CSS supports reusability because the foundational CIM metamodel is domain-independent and can be used and instantiated in different application domains and systems.

VI. Implications and Limitations

CIM-CSS provides a lightweight multi-disciplinary method for identifying the relevant context, and a formal multi-layered model and tool for specifying and managing the contextual information in CSSs. An intelligent CSS, through its sensing and inferring capabilities, observes internal and external states, assesses their influences, and collects the information to determine the actual context that holds, and adapts accordingly at run-time. CIM-CSS has an impact on the advance of context-sensitive systems design by implying (1) minimal ambiguity and uncertainty by eliciting and consolidating data and knowledge from multiple sources and different stakeholders, (2) reduced development complexity and enhanced software modularity by separating the context processing and management logic from the application logic i.e., separation of concerns, and (3) higher reliability by specifying multiple variants and fusion when interpreting the context information, which can support the system to use contextual alternatives when a state cannot be assessed, a sensor data is inaccurate, or a sensor failure occurs.

CIM-CSS models are outcomes of stakeholders’ elicitation and analysts’ evaluation of how human-machine interactions are influenced by (explicit and/or implicit) sensed data and context information. Therefore, CIM-CSS is specification-based rather than learning-based, and it supports the construction of models based on the knowledge of system experts. Consequently, CIM-CSS can be used to model several types of tacit facts (e.g., known unknowns, unknown knowns, and unknown unknowns [35]). It can be used to infer context elements from the first and second categories using logical reasoning on context models, but it cannot be used to infer elements from the third category (i.e., unknown unknowns), which particularly can be discovered by applying machine/deep learning techniques. In other words, a CIM-CSS instantiation model can be customized (e.g., using “derived” and “aggregated” associations) to express both learned and learning contexts at design time but it has no learning capability to check or predict the true value of a learned context element while a system is running.

VII. Conclusions and Future Work

The CIM-CSS presented here is a context modeling approach for intelligent context-sensitive systems. The approach establishes a four-layered modeling framework for conceptualizing and formalizing the different aspects of context identification and management, and takes into account the relationships between the context and the intentional and functional aspects of the system. We integrated the meta-object facility metamodeling techniques and formal methods to establish the framework and to present a formal metamodel of context. We used a model-driven development approach to build a context modeling language and tool, and retained a formal syntax and semantics. The presented model for the use-case of smart meeting room highlights the expressiveness and usability of the graphical tool when dealing with the context aspects and variations. The model was found to be able to identify the contexts in practice and manage the related aspects effectively.
In future work, we intend to extend the modeling approach to (1) build a reasoner on the graphical editor. The reasoner will be realized by adding mapping rules between the CIM metamodel and the BNF syntax to support bidirectional transformations between visual models and formal statements (e.g., the discovery of implicit relationships such as conflicts and contradictions between contexts in the context diagram), (2) add fuzzy meta-elements to the metamodel in order to specify the featured values of context, and (3) supplement the metamodel with run-time meta-elements that handle the adaptivity for self-adaptive CSSs.

REFERENCES


