CAMID: Architectural Support of Middleware for Multiple-Domain Ubiquitous Computing and IoT

Hamed Vahdat-Nejad
Pervasive & Cloud Computing Lab, Faculty of Electrical and Computer Engineering, University of Birjand, Iran
vahdatnejad@birjand.ac.ir

Abstract
Current Internet of Things (IoT) systems concentrate on one domain, such as smart home, university, office, etc., while many entities such as humans and mobile phones are mobile entities, roaming between different IoT domains. Most of the research on context-aware middleware for IoT addresses a specific domain and limited kinds of applications. Realizing the ultimate intelligence and automation in IoT will need a multiple-domain system. The architectural design of a multiple-domain context-aware middleware for IoT and Pervasive computing envisages serious challenges, including extensibility of the environment, need for distribution, resource limitation of mobile devices, mobility, and need for unique name allocation, which have not been fully addressed by related studies. This paper proposes the architectural design of a multiple-domain context-aware middleware (CAMID), which uses a distributed two-layer architecture to handle the above issues. CAMID has been developed in Java in two layers, including H-CAMID and L-CAMID. Scenario-based Architecture Analysis Method (SAAM) shows that CAMID satisfies the target quality attributes and outperforms previous multiple-domain solutions. Finally, an experimental evaluation of CAMID shows acceptable response time in a critical scenario.

Keywords- Context, Context-aware middleware, Architecture, Ubiquitous computing, Internet of Things, Domain

1. Introduction

Pervasive and ubiquitous computing, referred to as the next distributed computing paradigm [1], provides intelligent services in an anywhere/anytime manner to users. Internet of Things (IoT) is the most well-known concept, which describes a ubiquitous computing domain. Typical applications in ubiquitous computing and IoT are based on the information provisioned by sensors, called contextual information. As a result, ubiquitous computing and IoT are based on the concept of context-awareness [2]. Sensor-based or context-aware applications distinguish the ubiquitous from traditional distributed computing [3]. The main characteristic of a smart context-aware application, which makes it different from a classic application, is utilizing context information of the involved entities in the IoT environment to provide adaptive intelligent services to users [4]. A smart context-aware application should firstly discover the required context from distributed sources, then perform modeling (for transforming raw context data to meaningful information) and reasoning (to obtain high-level information from simple low-level contexts), and finally, adapt its action's behavior to this context. This approach yields sophisticated stand-alone smart applications requiring a long development time.

The scientific approach to implementing a smart place is to develop a context-aware middleware [5] [6], responsible for context discovery and registration, aggregation, modeling, reasoning, and distribution to the interested applications. In fact, the existence of this kind of middleware releases the application developers from context gathering and processing. Before designing a context-aware middleware, the characteristics and scope of the environment should be specified. Typical domains available in pervasive computing and IoT are personal, mobile, ad hoc, as well as domains physically limited to an IoT-based geographical zone such as smart places (e.g., smart room, home, office, university, etc.). Each of these domains can support specific kinds of smart applications.

Previously, several frameworks and toolkits have been developed to support context-aware applications for single domains (e.g. [7] [8] [9] [10] [11]), and also many architectures for context-aware middleware have been proposed (e.g. [12] [13] [14] [15] [16] [17]). As these solutions are designed for a single domain environment, they can support the development of very specific and limited kinds of applications. In fact, they are mostly developed for a special purpose [18] in a particular domain, such as a smart meeting room [12]. The following scenario helps to explain this limitation:

On a typical day, Kate goes from home to the university at 8 am. At 10 am, she goes to an office for official work and then to her husband's company to have lunch together.

As evident in this scenario, a typical user roams between different IoT domains regularly and casually: From home to outdoor (mobile domain), then university, and so on. Therefore, the user's context is generated in different domains, and middleware for a single and specific domain cannot provide the contexts produced in another domain. For example, when Kate is at the office, the smart home context-aware system cannot obtain the activity of her or her health situation or the exact place of her (meeting room, manager's room, or waiting room in the office building). Activity can only be derived from a group of sensors (motion, noise, light, etc.) installed in the office. Her health profile may be available in another context-aware system (e.g., a hospital or clinic), and her exact location can only be obtained from the sensors available in the office (not even the GPS of her mobile phone can provide it). Therefore, a context-aware middleware designed for just a home has serious limitations in supporting diverse kinds of context-aware applications. In fact, it can only provide the context information produced in the house and, at best, the context that can be sensed by the user's mobile phone (such as GPS, Calendar, and contact lists). In summary, an IoT system that only serves a single domain (such as a smart home) has limited intelligence and capabilities to offer to a user, and we need a multiple domain IoT system to realize the ultimate intelligence and automation.

To realize the pervasive computing and IoT vision, a context-aware middleware for a multiple domain environment should be developed to support beyond-domain smart applications. On the other hand, multiple domain environments introduce new challenges such as extensibility, mobility, the requirement for a uniform namespace, a uniform context distribution model, etc., that do not exist in single IoT domains (or could be dealt with much easier). Although a limited number of studies investigate the architectural design of context-aware middleware for a multiple domain environment [19] [20], they do not address most of the above requirements. Moreover, they are usually restricted to a centralized architecture and do not consider the constraints imposed by the fact that most of the devices available in the environment are mobile phones and PDAs with a highly dynamic nature and limited computational and availability capacities. In fact, none of the studies investigating platform support for context-aware applications provides a distributed and extendable architecture for a multiple-domain IoT environment that addresses the constraints of context-aware computing.

This paper proposes the architecture design of the Context-Aware MIDdleware (CAMID), aiming to provide an open architecture for a multiple-domain IoT and pervasive computing environment. The paper extends our previous paper [21], in which a middleware has been designed for a smart home domain. Moreover, we extended the distribution mechanism presented in our previous works [22] [23] and employed it in the proposed architecture. To address the constraints of mobile devices, CAMID is designed in two layers: The light-weight part (L-CAMID) and the Heavy-weight part (H-CAMID). L-CAMID is installed on any local server as well as limited-resource computational device aimed to be part of the environment by providing or consuming context. H-CAMID is installed on global dedicated servers aiming to hold a full image of the environment and support the L-CAMID layer in providing services. The structure of both of the layers is distributed. In addition, extensibility and openness are addressed by providing easy-to-use services for inserting new IoT domains, entities, and contexts. Last but not least, CAMID provides a unique and uniform naming allocation scheme, which has not been addressed by previous research.

We leverage the Scenario-based Architecture Analysis Method (SAAM) to evaluate CAMID and compare it with previous multiple domain middleware systems. The evaluation results show that CAMID satisfies the target quality attributes and outperforms the previous multiple domain architectures.

After this introduction, in the next section, our working definition of context and context-aware middleware as well as a described list of designing challenges of multiple domain context-aware middlewares are provided. Section 3 reviews the related multiple-domain architectures and discusses their capabilities and deficiencies. In section 4, an overview of the architecture of the CAMID is presented. Section 5 evaluates the proposed architecture, and section 6 discusses how the proposed architecture addresses the specified challenges. Finally, section 7 concludes the paper and presents the future direction of this research.

2. Designing challenges

In this section, we provide the definitions and then discuss the challenges that should be dealt with in the design stage of a multiple-domain context-aware middleware. Among available definitions of context [24] [25] [2], we refer to the definition proposed by Dey [26]: "context is any information that can be used to characterize the situation of an entity. An entity is a person, place, or object relevant to the interaction between a user and an application, including the user and application themselves". Moreover, a Context source is an entity that produces contextual data. In IoT and ubiquitous computing, most contextual sources are sensors.

Context-aware middleware "lies on top of the operating system of mobile phones, PDAs, personal computers, servers, laptops, and other computational devices available in the environment to discover context sources firstly, and then perform gathering, aggregation, conflict resolution, modeling, and reasoning, and finally create transparent distribution mechanisms to distribute pure context information among interested applications" [5]. To design the architecture of a single-domain context-aware middleware for IoT, many issues should be handled, such as the dynamic nature of context, the increasing number of context elements, context sources and consumers (e.g., applications), mobility of the entities, and the need for distribution, extensibility, and scalability. Dealing with these concerns in a multiple-domain IoT-based environment is much more challenging. Moreover, new challenges emerge in these environments that do not exist in single domains. In the remaining part of this section, different challenges that should be handled during the architectural design of a multiple-domain context-aware middleware are discussed:

- *Resource limitation of devices*: Most of the devices available in the environment (e.g., in the IoT environment) are mobile devices with limited storage, computation, communication, and availability capacities. They cannot play the role of high-performance computers and may be inaccessible or even off. Therefore, any plan for using them as the middleware infrastructure should consider these limitations.
- *Need for distribution*: Centralized architectures for middleware can introduce problems. They usually lose efficiency when applied to a large-scale IoT environment. In addition, they rely on a centralized component that could create a bottleneck in the system when accidentally the system load increases. Moreover, they introduce a single point of failure. As a result, the architecture of a multiple-domain context-aware middleware should provide enough distribution.
- *Dynamic nature of the environment*: Handling a dynamic environment is a traditional issue in distributed systems. However, an IoT-based pervasive computing environment is highly dynamic in which devices join and leave frequently and context changes. Some devices such as mobile phones may be regularly switched off and on. Similarly, some of the context types, such as a vehicle's location and activity of a person, are changing fast.
- *Extensibility and scalability:* In a multiple domain IoT-based environment, the number of domains may increase by joining other homes, offices, organizations, hospitals, and user personal environments, so the middleware should be extensible. Besides, there are numerous entities and several types of contexts for each entity. Furthermore, new entities, things, and contexts are introduced over time, and new context sources (e.g., sensors)

and context-aware applications are registered to the system. The middleware should not lose efficiency when the scale of the environment increases.

- *Need for context aggregation*: In a multiple-domain IoT environment, the context of a distinct entity is produced in different domains, and also, there may be more than one source for acquiring a single type of context. For example, when Kate goes from home to office, her correct activity can be sensed in her current domain (while the previous domain may still provide the old and wrong value of this context). On the other hand, her location can be obtained from both her mobile phone GPS and the sensors available in the office. At the same time, her health records may be available in a clinic domain. These remarks raise the need for virtually aggregating context and providing the illusion of a centralized context store.
- *Mobility*: In multiple-domain IoT environments, handling the mobility of entities is a challenging issue. Entities such as people, mobile phones, laptops, etc., continuously roam (revisiting or entering for the first time) between different domains such as home, office, university, etc. These entities are usually sources of context data and sometimes platforms for residing context-aware applications (e.g., mobile phones). In either case, they impose new tracing challenges to the system (especially in distributed architectures). This issue especially complicates context discovery, aggregation, and distribution.
- *Need for unique name allocation*: considering the fact that in IoT, mobile things roam between different domains and their context is produced in various domains imposes that these entities should be recognized by a unique name all over the environment. To our best knowledge, none of the multiple-domain studies gives a solution for this issue. They usually ignore this problem by assuming that all entities already have a unique name.
- *Global and transparent context distribution mechanisms*: Ultimately, the goal of the IoT-based middleware is to support the development of intelligent context-aware applications. The benefits and services of the middleware to the applications are provided and seen through context distribution APIs. There are different approaches for data distribution in the traditional distributed systems, such as publish-subscription and query languages. However, adapting them to the multiple-domain IoT-based middleware envisages new challenges such as the requirement for a general query structure that applies to every domain, a uniform namespace for all the entities and things available in the environment, and transparency of the distribution mechanisms from the view of application developers.

3. Related work

While there are many papers related to single-domain context-aware middleware(e.g. [17] [9] [15] [11] [27] [28] [10] [29] [30] [31] [32]), in this section, we only survey the existing studies related to multiple-domain context-aware middleware architecture. These research studies have been accomplished under different titles. Still, they propose the architecture of a part of the context-aware middleware for IoT or pervasive computing.

Gaia [12] is an early middleware architecture designed for a multiple-domain Active Space. The defined active space consists of homes, offices, and meeting rooms. Gaia involves a central context service component responsible for context discovery, aggregation, modeling, reasoning, and distribution. It maintains the information about all context sources in the environment. The context service also lets applications query and register for their required context types. To support the mobility of the users, a presence service responsible for detecting the place of the people and devices and a Context File System responsible for helping users locate their required context in the environment have been designed. The centralized architecture of the context file system makes bottleneck as well as scalability concerns in context discovery, aggregation, and distribution. The architecture is merely suitable for a limited and fixed environment.
SOCAM [33] [34] is a service-oriented context-aware middleware for a multiple-domain environment. It gathers and converts various domains from which contexts are produced into a single semantic space. Then, by providing a central component for service locating, it supports context discovery, aggregation, and distribution. This scheme envisages

the scalability issue as the size of the environment or the number of domains increases. In fact, SOCAM does not support enough distribution. Moreover, before including a new domain, its ontology should be formed and inserted into the hierarchy, which introduces extensibility concerns and makes the architecture suitable for a predefined and roughly fixed environment.

Context management framework (CMF) [35] is part of the architecture of a multiple-domain context-aware middleware. The central registry should handle numerous inclusions, updates, and modifications from context sources and help respond to the queries of applications in finding their suitable context sources. Guaranteeing uninterrupted access to the registry is difficult in a mobile environment and faces scalability issues when the size of the environment increases. Moreover, the mobility issue has not been addressed in this study either.

The Awareness project [36] [37] [38] [39] [40] considers four domains, including mobile, home, office, and ad-hoc. Firstly, a local middleware for each of the domains has been developed, and then by designing bridges between each two of these domains, the interactions between them are established. A bridge is a functional component that enables the local context-aware applications to obtain their context requirements from another domain. The main issue is the large number of bridges required to fully connect many domains, which makes the introduction and insertion (extensibility) of a new domain very difficult.

Feel@Home [41] (and also OPEN [42] and iPlumber [43] frameworks of the same authors) considers three different domains- home, office, and outdoor. Each domain has a middleware with a centralized architecture. A centralized component, referred to as Global Administration Server (GAS), is responsible for interlinking the local middlewares. The main contribution of this research is handling the mobile entity problem and proposing an inter-domain context distribution mechanism. Nevertheless, the system architecture is central, and GAS should handle and track all entities of the environment and answer the inter-domain queries (for context distribution), which are issued by context-aware applications. Hence, there is always a huge load on GAS, and in the case of failure, the whole system almost fails.

Multi-level Context-aware Framework (mlCAF) [19] provides a multiple domain context modeling and reasoning strategy by proposing ontologies for a few domains as well as reasoning methods. It does not investigate context distribution and extensibility of the environment.

AUM-IoT [20] primarily considers healthcare and transportation domains and proposes a semantic agent-based middleware. The architecture of AUM-IoT consists of four layers: perception, middleware, fog, and cloud. The perception layer is responsible for context acquisition. Middleware and fog layers deal with context processing, and the cloud permanently stores context and hosts context-aware services.

Moreover, none of the previous research considers the requirement that all the entities available in the environment should be recognized by a unique name. This paper proposes the open architecture of Context-Aware MIDdleware (CAMID) for a multiple-domain IoT-based pervasive computing environment, which aims to support the development of more powerful applications running on mobile devices. The issues considered during the design stage are extensibility and scalability, support for distribution, and mobility management. Furthermore, the assignment of a unique id to the entities is presented, and transparent mechanisms for distributing context are provided in the architecture.

4. Proposed architecture

In this section, an overview of the architecture of CAMID is presented. CAMID is designed for a multiple-domain IoT-based ubiquitous computing environment in which IoT supports each domain. Homes, universities, offices, hospitals, museums, and city transportation systems are typical domains of IoT. IoT services are usually presented to users via installed screens (e.g., the control panel screen in smart homes) as well as mobile phones. Since mobile devices have limitations in memory, computation, battery charge, and availability, a two-level infrastructure, which consists of dedicated global servers and mobile, local servers, or non-dedicated computational devices, is considered to host the multiple-domain IoT-based middleware. In accordance, CAMID is designed in two layers: Light-weight (L-CAMID) and Heavy-weight (H-CAMID). This architecture helps to achieve better scalability and extensibility in a multiple domain IoT environment.

L-CAMID is primarily designed for mobile devices but can be used on any computational device or local server that does not aim to serve as a dedicated global server. In fact, each device involved in the environment by hosting a context provider (e.g., things that embed sensors) or a context consumer (e.g., smart context-aware application) should execute L-CAMID. L-CAMID provides components for helping context providers (things) register and publish their context and supporting context-aware applications (smart applications) to find their required contextual information. On the other hand, H-CAMID maintains a global view of the IoT environment, implements the massive part of the CAMID, and serves as the foundation for supporting the light-weight middleware (L-CAMID) installed on devices with limited capabilities. It lies on global dedicated servers, which can reside in any domain or cloud. Figure 1 shows an overview of the architecture of CAMID. In the remaining part of this section, we describe the components and functionality of the CAMID.

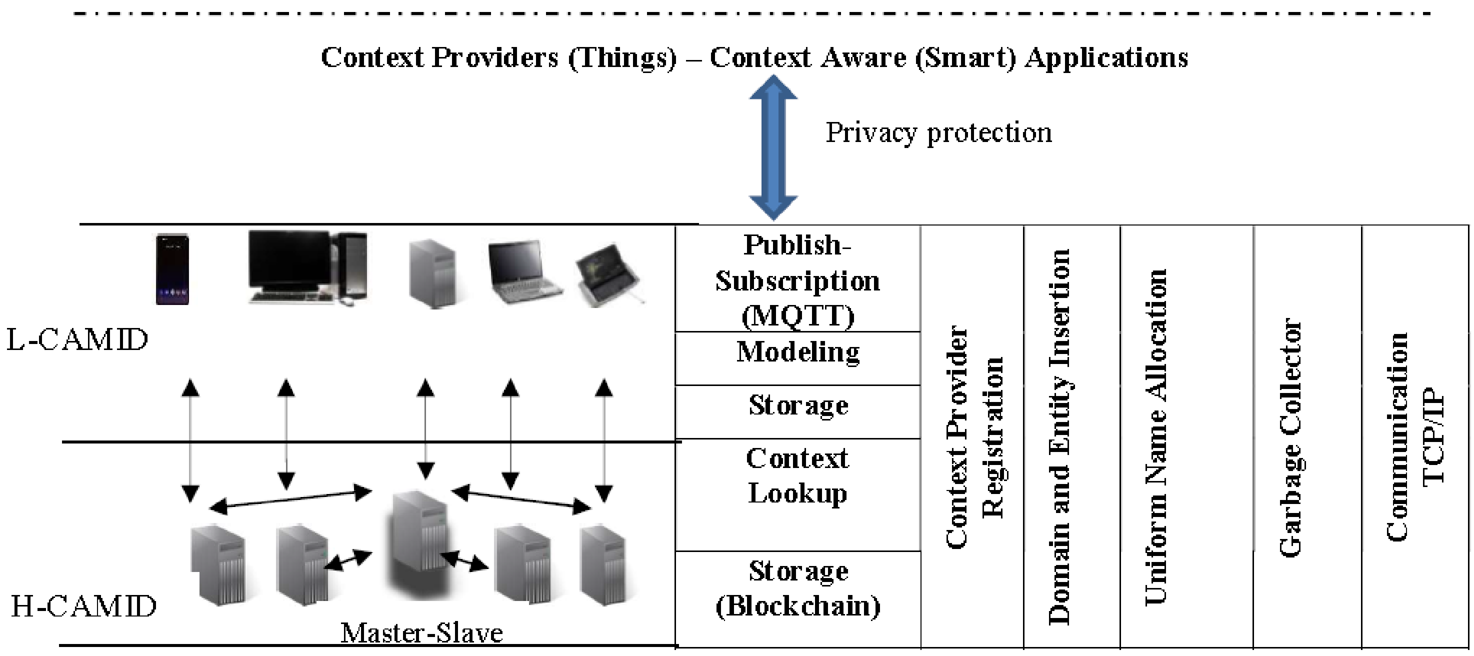


Fig. 1 The overall architecture of the CAMID[1]

4.1 Context provider

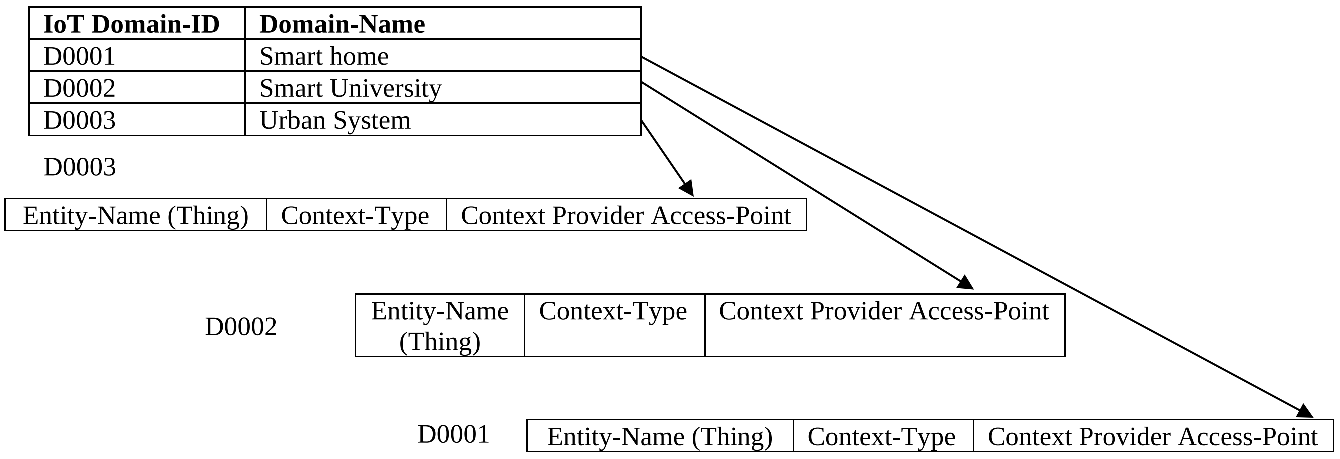
A simple context provider is a light-weight module responsible for gathering a context from a context source and modeling it to be used by applications. A more complicated context provider is responsible for inferring a high-level context (such as activity or nearby digital devices) from low-level context types. In the IoT environment, context providers obtain the contextual values by interacting with sensors. A context provider program uses the services provided by L-CAMID and can be executed on a smartphone or a thing. It stores the values of the context with a timestamp specifying the generated time.

4.2 H-CAMID

H-CAMID is the main part of CAMID and is installed on dedicated servers. It stores the name of all IoT domains and their identities in the Domain table. To ensure reliability, it uses blockchain technology for storing domains and entities. For each domain, a separate table maintains its registered entities (things), context types available for each

[1] Horizontal modules show the layers' functionalities, while vertical modules show the system functionalities

entity, and the access point of the devices hosting the context provider (for each context). It should be noted that there may exist more than one device hosting different context providers of a specific context type of a single entity. Figure 2 shows the schema of these tables. By storing the tables, each of these servers tries to maintain a full view of all IoT domains, entities, and contexts available in the environment. They utilize the master-slave paradigm on TCP/IP communication to keep their tables updated. If the master server goes offline or envisages a failure, another server is elected as the master. When a slave updates its database, it propagates the updates to the master. Slave servers periodically poll the master for probable updates.
H-CAMID provides a registration method, which can be called by L-CAMID's registration method. The aim is to register the information of a new context provider. The method gets the domain id, entity name, context type, and access point of the provider, updates the table associated with this domain id, and then disseminates the information to the master.



IoT Domain-ID	**Domain-Name**
D0001	Smart home
D0002	Smart University
D0003	Urban System

D0003

Entity-Name (Thing)	Context-Type	Context Provider Access-Point

D0002

Entity-Name (Thing)	Context-Type	Context Provider Access-Point

D0001

Entity-Name (Thing)	Context-Type	Context Provider Access-Point

Fig. 2 An example of the schema of the tables maintained in H-CAMID

Besides, H-CAMID provides insert-domain and insert-entity services, which could be called by L-CAMID's corresponding domain and entity insertion methods. These services provide the extensibility for inserting new things and domains into the IoT environment. On the contrary, it provides a garbage collector service to remove useless domains, entities, and context providers to prevent the tables from getting too large with redundant data. The garbage collector service is described in another subsection. In the case of new IoT domain insertion, a unique identifier (in an incremental way) is assigned to the new domain, and the corresponding record is inserted into the Domain table and stored in a new block. Afterward, a new table for this domain is created containing its entities (things), contexts, and access point of the context providers, and the information is sent to the master.
The insert-entity service, which aims to register a new entity or thing, has two parameters: domain-id and entity name. If the entity is previously available in the mentioned domain, an error message is returned. Otherwise, since a mobile entity such as a human may have been previously registered in another domain, a list of the entities with the same name in other domains is returned to the calling method (in L-CAMID). Upon receiving the confirmation, the entity is inserted into the corresponding domain table. Subsequently, as stated before, this update is forwarded to the master.
Finally, H-CAMID provides a lookup method, which aims to find the access point of context provider(s) of a typical context. L-CAMID's publish-subscription service can call the method. It gets domain-id, entity name, and context type as parameters. Then, by searching the table of this domain, it returns the access point of the provider(s) (if available) or Null (if the information is not available).

4.3 L-CAMID

Each computational device (such as smartphones or things) that hosts a context provider or smart application (context consumer) should execute the light version of CAMID. L-CAMID provides services for context modeling (utilized by context provider program), registration, inserting new entity or domain, storage, and context publish-subscription. L-CAMID uses the SensorML [39] to provide the modeling service. Registration service is utilized by context providers and is responsible for registering and distributing the information about new context providers. It gets the information about the domain id, entity name, context type, and access point of the provider and notifies a known server by calling its registration service available in the H-CAMID layer.

A context provider can utilize the local domain-insertion service, which gets the new IoT domain name, entities, and their corresponding contexts and access point of the providers. L-CAMID sends this information by calling the domain-insertion method of H-CAMID from a known server. Subsequently, as stated before, a unique identifier is assigned to the new IoT domain, and the updates are propagated in the system.

L-CAMID provides the local insert-entity service for things so that a context provider can publish the context of a new entity or thing. For this, entity name (thing) and IoT domain id (the domain associated with the thing) are specified as parameters and sent to a known fixed server by calling the related method from H-CAMID. As stated before, the global server responds with an error (if the thing is already available in the specified IoT domain) or with a list of similar entities in the other domains. In the latter case, L-CAMID provides the list to the context provider and waits for the response. Consequently, if the new entity is different from all of the entities available in the list, the insertion is confirmed and sent to the H-CAMID.

For devices hosting a context provider (such as the things embedding sensors), the storage service maintains the history of the produced contexts together with their timestamp. For devices hosting smart applications (context consumers), the storage service maintains the access point of the context providers together with the IoT domain id and entity name of the contexts, as follows:

Consumed-Context (Context Type, Entity Name, Domain ID, Context-Provider Access-Point).

Storage service also maintains a list of known servers together with their access point address. It can update this list by sending a request to a server.

Finally, the publish-subscription service is designed to disseminate context among interested smart applications. It gets IoT domain-id, entity name (thing), and context type as parameters and checks the local storage service to see if a local context provider module provides the context. If not, it searches the Consumed-Context table to determine whether the associated context provider(s) access point is available. If the table does not have any record associated with the specified context, the context lookup method of a known server is called. Afterward, the result returned from H-CAMID is cached in the Consumed-Context table for probable later requests. Then through the communication service (MQTT protocol) provided by L-CAMID, it subscribes to the remote context provider and delivers the published values to the context-aware application.

4.4 Uniform namespace

CAMID assigns a unique id to each IoT domain, as elaborated previously. For referring to things, it categorizes the entities into two categories: independent and dependent. An entity is called independent if it could be described and known independently of other entities in its domain. For example, a human is considered an independent entity. On the other hand, dependent entities use other entities for identification, such as the laptop or mobile phone of a person or the electronic lock of a room. In the first category entity name is a simple name (e.g., John Cooper), but in the latter case, it is an entity chain starting with an independent entity (e.g., John Cooper. Laptop, Seminar Room. Electronic lock). CAMID uses the unique "Domain id.Entity name" structure for referring to an entity. Similarly, a unique triple structure, "Domain id.Entity name. Context type, " refers to a context.

4.5 Context registration and dissemination

Any computational device available in the environment can host a context provider. The context provider module makes use of the services offered by the L-CAMID layer installed on the device. It gathers data from a sensor, models

it using the primary modeling service, and registers it by calling the registration service. In addition, a context provider can introduce a new IoT domain and thing by utilizing domain-insertion and entity-insertion services, respectively, and provide contextual information about the new entities. A context provider may go offline; anytime a context provider reinitiates, it calls the registration service.
Similarly, any computational device available in the IoT environment can host a smart application. A smart application can utilize the publish-subscription service provided by L-CAMID to receive its contextual needs. For this, the application should specify IoT domain-id, entity name, and context type. Subsequently, the CAMID uniquely finds the provider and establishes a connection between the L-CAMID of the context provider's device and the application's device by using the (Message Queue Telemetry Transport) MQTT[2] protocol [44]. MQTT is a light-weight M2M[3] messaging protocol for IoT applications. It implements the publish-subscription paradigm for data transferring. Since there may be more than one context provider for a single context, the application can choose among them according to the timestamp of the generated values.

4.6 Garbage collector

Since any context provider can insert new IoT domains and things, the size of the system tends to grow over time. Besides, the pervasive computing environment is highly dynamic, so many context providers may operate temporarily. To address this issue, a garbage collector service is provided, which is periodically executed on the master server and deletes the out-of-order context providers and useless things and domains. Toward this purpose, if an application realizes that a context provider is not responsive or finds its context values useless, it can report it to the L-CAMID as either nonresponsive or useless. L-CAMID deletes the associated record(s) from the Consumed-Context table and redirects the report to a known server, which in turn propagates it to the master. In the case master receives more than a threshold number of reports about a context provider, it removes the corresponding record. Moreover, it periodically checks the tables and removes the records associated with IoT domains and things with no specified context provider.

5. Evaluation

We used Java as a portable programming language to develop the CAMID architecture prototype. Java programs are translated to byte code, which is runnable on the Java virtual machine. Java virtual machine runs on various platforms; therefore, Java code can run on any device such as a smartphone, tablet, PC, server, etc. H-CAMID functionalities, including IoT domain and entity (thing) insertion, context provider registration, and context lookup, are implemented as Java methods, which are remotely invoked via Java RMI[4] from the components of the L-CAMID.
Scenario-based methods are the most important approaches to evaluating software architectures [45]. We use the Scenario-based Architecture Analysis Method (SAAM) [45], as the main scenario-based method [46] to evaluate the proposed architecture. We compare the proposed architecture with Feel@home [41] [42] and mlCAF [19] as the available multi-domain context management architectures. To this end, the quality attributes of reusability, expandability, transparency, and distribution are investigated. We adopt the scenarios presented in tour-guide [14], Ubihome [15], Feel@home [41], and Hotel room [9].
Reusability- Reusability is an important quality attribute for the middleware. In the IoT environment, developers should be able to reuse context providers. CAMID provides the context registration mechanism by which context providers and things can register their contextual capacities. On the other hand, it provides the context dissemination mechanism by which context consumers can find and reuse the required context providers. Feel@home does not investigate and support the reusability of components. In mlCAF, the "High-level Context Notifier" component provides reusability of contextual information.
Expandability- User, domain, and environment expansion are typical of IoT and pervasive environments. Each user can add their device to the environment by installing H-CAMID. Therefore, the device can host a context provider or

[2] https://mqtt.org/
[3] Mobile-to-Mobile
[4] Remote Method Invocation

a context-aware application. Besides, users can easily expand the environment by using entity and domain insertion services. In Feel@home and mlCAF, the domains and entities are predefined and cannot be expanded. AUM-IoT provides an entity registration service but does not support domain registration.

Transparency- The transparency of the context dissemination mechanism is important from two aspects, including location and access. Context provider components are developed on various platforms such as Linux, Windows, Android, IOS, etc., and are located in different places and domains. In CAMID, when a context-aware application requires a contextual element, it invokes the local context dissemination mechanism, which in turn finds the corresponding context provider and subscribes to it. Afterward, the application receives the context through the publish-subscription paradigm. As the context-aware application requests for the context through a uniform API, regardless of the context provider platform and location, and without the need for knowing the platform and location, the dissemination mechanism is transparent from both aspects of location and access. Feel@home provides two mechanisms for context dissemination. Publish-subscription for local context elements and query-based method for non-local contexts. To this end, the user should be aware of the location of the context element to be able to retrieve it. As a result, context dissemination is not transparent from both aspects of location and access (As there are two mechanisms for accessing context). However, the dissemination mechanism is transparent inside a single domain (which is publish-subscription). In AUM-IoT, retrieving the global context is not transparent. mlCAF uses a transparent mechanism for context fusion.

Distribution: Centralized architectures are based on a single point of control, while distributed architectures are based on multiple control points. CAMID is designed in two layers; both of them are totally distributed. L-CAMID uses a peer-to-peer architecture in which each device runs L-CAMID services. H-CAMID uses a distributed master-slave architecture. On the contrary, Feel@home uses a central global server, which is responsible for most of the duties of the system, including tracking entities and context dissemination. Similarly, mlCAF relies on the centralized context ontology manager.

In the final stage of SAAM, the proposed architecture is compared with previous research. In this regard, table 1 summarizes the comparison results of CAMID with Feel@home and mlCAF regarding the investigated quality attributes.

Table 1 Result of the evaluation using SAAM

Quality attributes	Feel@home [41]	mlCAF [19]	AUM-IoT [20]	CAMID
Reusability	×	✓	✓	✓
Expandability	×	×	Partly	✓
Transparency	×	✓	×	✓
Distribution	×	×	Partly	✓

In continue, we perform an experimental simulation to further evaluate CAMID. We adopt a recent scenario from [20], which is briefly described as follows:

Alice is a BSc degree student currently participating in the IoT class. During the class session, she suddenly faints. The smart healthcare application on her smartphone detects this event immediately, automatically asks for an ambulance, and notifies Dr. Cooper, the clinic officer at the university. Dr. Cooper requests her medical history context of Alice. After receiving it, he analyzes the information and provides initial advice to the ambulance personnel. Meanwhile, he performs initial actions to prepare the clinic for the treatment.

In this scenario, two context elements are used, including the current health status and medical history of Alice. We discuss how CAMID handles this situation:

- Current health status: This context is provided by inference from the biotic sensors available on Alice's smartphone. The local smart healthcare application infers this context and provides it to Dr. Cooper by the CAMID's publish-subscription dissemination mechanism.
- Medical history: Dr. Cooper uses CAMID's dissemination mechanism to acquire this context. He invokes CAMID's dissemination mechanism from local L-CAMID installed on his computer. Subsequently, L-CAMID invokes the context lookup service from the H-CAMID available on a server, and finally delivers Alice's medical history to Dr. Cooper.

As this is a critical scenario, the time elapsed for obtaining contextual elements is the most important factor. Among the above-discussed context elements, current health status is obtained locally, so elapsed time is negligible. However, the medical history context is available on the global layer and obtained through a series of interactions. For this, we perform an experiment to evaluate the response time for obtaining the medical history context in a real situation.
We established a global H-CAMID server containing several medical histories. We further installed the L-CAMID on the local laptop and invoked CAMID's context dissemination service for different context volumes. We iterated each invocation 100 times. Table 2 shows the mean values of context dissemination response time. As concluded, the response time is mainly affected by Internet capacity. In fact, the intrinsic delay of the CAMID has been acceptable for this critical scenario.

Table 1 Context dissemination response time

Global context volume (MB)	1	50
Response time (Seconds)	2.6	47.2

6. Discussion

In the following, we discuss how CAMID addresses the challenges previously identified for the architecture design of multiple-domain context-aware middleware for IoT environments.
Resource limitation of devices: CAMID considers the limitations of the computational devices (things) involved in the IoT environment using a two-level platform. On the one hand, users get services from the middleware by interacting with their devices, and on the other hand, the massive part of the middleware is installed on the distributed servers. In fact, the devices carry a light-weight middleware, which provides its services by calling methods from dedicated servers through Java RMI.
Need for distribution: CAMID supports a suitable distribution level in both layers. Devices hosting context providers or applications operate totally in a distributed manner, and servers exploit a distributed master-slave paradigm to keep themselves up-to-date.
Dynamic nature of the environment: The dynamic nature of the environment is addressed in the architecture by maintaining the general view of the environment in dedicated servers. Anytime a context provider is initiated, the corresponding information is propagated in the system. Moreover, the garbage collector component periodically removes the information about unavailable or useless context providers from the tables. Even the failure of a server does not cause serious issues because the environment's state is replicated on other servers. Finally, since smart applications obtain their required contextual information using a publish-subscription scheme from the context providers (rather than from a static context-store), they are notified of any change in the context.
Extensibility and scalability: CAMID provides easy-to-use services for facilitating extensibility. Any device could easily be part of the IoT environment by running an instance of the L-CAMID. Therefore, any user can insert a thing or smart application into the environment by simply developing it on the L-CAMID layer. Inserting new IoT domains and entities is also done using the corresponding services available in the L-CAMID. Designing CAMID in two totally distributed layers helps in achieving better scalability.
Need for context aggregation: H-CAMID performs context aggregation and provides a full view of the environment.
Mobility: Since all the inserted entities are primarily assigned to a domain as the "home domain", their mobility does not impose any problem in recognizing them.
Need for unique name allocation: The hierarchical naming scheme, which starts with the IoT domain id, together with the unique domain id allocation, ensures all the entities are recognized by a unique name all over the environment.
Global and transparent context distribution mechanisms: Applications can acquire their required context by submitting IoT domain id, entity name, and context type. Therefore, the context is transparently delivered to them using the publish-subscription paradigm supported by CAMID.

In summary, CAMID is suitable for open IoT environments in which context providers and context-aware applications can freely join and leave. Any user can develop a program for gathering a context from a sensor and modeling it or even inferring a high-level context from simple context types. Subsequently, the context is registered using the registration service and could be usable by other application developers, so the application developers are released from re-providing previously available context information.

7. Conclusion

In this paper, the open architecture of context-aware middleware for multiple-domain pervasive computing and IoT environment has been proposed to facilitate the development of smart applications. By running a light-weight middleware layer (L-CAMID), any device and thing can host a context provider or context-aware application. Therefore, extensibility and distribution are the main characteristics of CAMID that make it suitable for an open extra-large IoT-based environment. To address the limitations of the typical computational devices available in the environment, the heavy part of the middleware (H-CAMID) has resided on distributed dedicated servers to maintain a universal view of the environment and support the L-CAMID layer. CAMID also provides a uniform name allocation scheme to recognize entities uniquely all over the environment. Evaluation results show that CAMID fulfills the design requirements and acts well in a critical scenario.
In the current design of CAMID, an application developer should be aware of the entity name and domain-id of a context as it is registered in the system to be able to acquire the context using the publish-subscription method. As one of the future directions, we are working on designing a more flexible and easy-to-use query mechanism that helps IoT application developers find their contextual needs without strict information about the entity name and the corresponding domain-id. Moreover, in the current design of CAMID, some of the servers may encounter a heavy load due to receiving more requests from the H-CAMID layer. It emerges the requirement of a load balancing scheme to handle this situation.
Besides, privacy protection is a major concern for users participating in IoT environments. Another research direction is to design appropriate privacy protection services for CAMID in two stages and insert them into the corresponding layers. The privacy protection should be conceptually lying in the context registration and dissemination.

Compliance with Ethical Standards

The author has no relevant financial or non-financial interests to disclose.

Data availability

Data sharing is not applicable to this article as no datasets were generated or analyzed during the current study.

Code availability

https://github.com/The-J-J/CrowdBIG.git, Open source, 2022, developed in Java

References

[1] M. Weiser and J. Brown, "The Coming Age of Calm Technology," in *Beyond calculation: the next fifty years*, P. Denning and R. Metcalfe, Eds., New York, Copernicus, 1997, pp. 75 - 85.

[2] A. Zimmermann, A. Lorenz and R. Oppermann, "An operational definition of context," in *Proceedings of the 6th international and interdisciplinary conference on Modeling and using context*, 2007.

[3] H. Vahdat-Nejad, S. Ostadi Eilaki and S. Izadpanah, "Towards a Better Understanding of Ubiquitous Cloud Computing," *International Journal of Cloud Applications and Computing,* vol. 8, no. 1, pp. 1-20, 2018.

[4] H. Vahdat-Nejad, S. Izadpanah and S. Ostadi-Eilaki, "Context-aware cloud-based systems: design aspects," *cluster computing,* vol. 22, no. 5, pp. 11601-11617, 2019.

[5] H. Vahdat-Nejad, "Context-aware middleware: a review," in *Context in computing,* Springer, 2014.

[6] B. Bhushan , "Middleware and Security Requirements for Internet of Things," in *Micro-Electronics and Telecommunication Engineering,* Singapore, Springer, 2022, pp. 309-321.

[7] A. K. Dey, G. D. Abowd and D. Salber, "A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications," *Human-Computer Interaction,* vol. 16, no. 2, pp. 97-166, 2001.

[8] H. Chen, T. Finin, A. Joshi, L. Kagal, F. Perich and D. Chakraborty, "Intelligent Agents Meet the Semantic Web in Smart Spaces," *IEEE Internet computing,* vol. 8, no. 6, pp. 69 - 79, 15 November 2004.

[9] J. Euzenat, J. Pierson and F. Ramparany, "Dynamic context management for pervasive applications," *The Knowledge Engineering Review* , vol. 23, no. 1, pp. 21-49 , March 2008.

[10] K. Michalakis and et al, "A Context-Aware Middleware for Context Modeling and Reasoning: A Case-Study in Smart Cultural Spaces," *Applied Sciences,* vol. 11, no. 13, p. 5770, 2021.

[11] C. Drăgănescu, "Transport Oriented Framework for Context-Aware Services Management," *Advances in Science and Technology,* vol. 110, 2021.

[12] M. Román, C. Hess, R. Cerqueira, A. Ranganathan, R. H. Campbell and K. Nahrstedt, "A Middleware Infrastructure for Active Spaces," *IEEE Pervasive Computing,* vol. 1, no. 4, pp. 74-83, October 2002.

[13] T. Gu, H. K. Pung and D. Q. Zhang, "Toward an OSGi-Based Infrastructure for Context-Aware Applications," *IEEE Pervasive Computing,* vol. 3, no. 4, pp. 66-74, October 2004.

[14] A. Cadenas and e. a. , "Context Management in Mobile Environments: a Semantic approach," in *Proceedings of the 1st Workshop on Context, Information, and ontologies*, Greece, 2009.

[15] Y. Oh, J. Han and W. Woo, "A context management architecture for large-scale smart environments," *IEEE Communications Magazine,* vol. 48, no. 3, pp. 118 - 126, March 2010.

[16] F. Li, S. Sehic and S. Dustdar, "COPAL: An adaptive approach to context provisioning," in *proceedings of IEEE 6th International Conference on Wireless and Mobile Computing* , Canada, 2010.

[17] R. Baldoni and e. al., "An Embedded Middleware Platform for Pervasive and Immersive Environments for-All," in *Proceesings of the 6th Annual IEEE Communications Society Conference on Sensor, Mesh and Ad Hoc Communications and Networks Workshops*, Rome, 2009.

[18] R. Rocha and M. Endler, "Domain-based Context Management for Dynamic and Evolutionary Environments," in *Proceedings of the 4th on Middleware doctoral symposium*, Newport Beach, California, 2007.

[19] M. A. Razzaq and et al., "mlCAF: Multi-Level Cross-Domain Semantic Context Fusioning for Behavior Identification," *Sensors*, vol. 17, no. 10, p. 2433, 2017.

[20] P. Pradeep, S. Krishnamoorthy and A. V. Vasilakos, ""A holistic approach to a context-aware IoT ecosystem with Adaptive Ubiquitous Middleware," *Pervasive and Mobile Computing*, vol. 72, p. 101342, 2021.

[21] H. Vahdat-Nejad, K. Zamanifar and N. Nematbakhsh, "Context-Aware Middleware Architecture for Smart Home Environment," *International Journal of Smart home*, vol. 7, no. 1, pp. 77-86, 2013.

[22] H. Vahdat-Nejad, K. Zamanifar and N. Nematbakhsh, "A New Approach to Context Distribution in Large-Scale Environments," in *Proceedings of the International Conference on Trends in Information Technology and Applications*, Ajman, UAE, 2010.

[23] H. Vahdat-Nejad, K. Zamanifar and N. Nematbakhsh, "A New Community-Based Context Distribution Approach for Large-Scale Pervasive Systems," *International journal of ad hoc and ubiquitous computing*, vol. 14, no. 2, pp. 90-98, 2013.

[24] B. N. Schilit, N. Adams and R. Want, "Context-aware computing applications," in *Proceedings of the First Workshop on Mobile Computing Systems and Applications*, California, 1994.

[25] J. Pascoe, "Adding Generic Contextual Capabilities to Wearable Computers," in *Proceedings of the 2nd IEEE International Symposium on Wearable Computers*, Pittsburgh, 1998.

[26] A. K. Dey, "Understanding and using context," *Personal and Ubiquitous Computing*, vol. 5, no. 1, pp. 4-7, February 2001.

[27] J. Zhang, M. Ma, W. He and P. Wang, "On-demand deployment for IoT applications," *Journal of Systems Architecture*, vol. 111, p. 101794, 2020.

[28] E. Symeonaki, K. Arvanitis and D. Piromalis , "A context-aware middleware cloud approach for integrating precision farming facilities into the IoT toward agriculture 4.0," *Applied Sciences*, vol. 10, no. 3, p. 813, 2020.

[29] R. Zgheib, E. Conchon and R. Bastide, "Semantic middleware architectures for IoT healthcare applications," in *Enhanced Living Environments*, Springer, 2019, pp. 263-294.

[30] M. Mallegowda, P. Sarashetti and A. Kanavalli , "SOA-Based Middleware Framework for IoT Applications," in *Second International Conference on Sustainable Expert Systems*, Nepal, 2022.

[31] V. Rodrigues and et al., "HealthStack: Providing an IoT Middleware for Malleable QoS Service Stacking for Hospital 4.0 Operating Rooms," *IEEE Internet of Things Journal*, p. In press, 2022.

[32] L. Shen, "Multi-Software Architecture and Ubiquitous Multi-Center Internet of Things based Intelligent Library System," in *6th International Conference on Trends in Electronics and Informatics*, India, 2022.

[33] T. Gu, H. K. Pung and D. Q. Zhang, "A Middleware for Building Context-Aware Mobile Services," in *Proceedings of 59th IEEE Vehicular Technology Conference*, Milan, 2004.

[34] T. Gu, H. K. Pung and D. Q. Zhang, "A service-oriented middleware for building context-aware services," *Journal of Network and Computer Applications,* vol. 28, no. 1, p. 1–18, January 2005.

[35] H. v. Kranenburg, M. S. Bargh, S. Iaco and A. Peddemors, "A Context Management Framework for Supporting Context-Aware Distributed Applications," *IEEE Communications Magazine,* vol. 44, no. 8, pp. 67 - 74, 21 August 2006.

[36] M. J. Van Sinderen, A. T. Van Halteren, M. Wegdam, H. B. Meeuwissen and E. H. Eertink, "Supporting Context-aware Mobile Applications: an Infrastructure Approach," *IEEE Communications Magazine,* vol. 44, no. 9, pp. 96 - 104, September 2006.

[37] F. Liu and G. Heijenk, "Context Discovery Using Attenuated Bloom Filters in Ad-hoc Networks," *Journal of Internet Engineering,* vol. 1, no. 1, pp. 49-58, 2007.

[38] P. Pawar, A. Van Halteren and K. Sheikh, "Enabling Context-Aware Computing for the Nomadic Mobile User: A Service Oriented and Quality Driven Approach," in *Proceedings of IEEE Wireless Communications and Networking Conference*, Kowloon , 2007.

[39] C. Hesselman and e. al., "Bridging Context Management Systems for Different Types of Pervasive Computing Environments," in *Proceedings of the 1st international conference on MOBILe Wireless MiddleWARE, Operating Systems, and Applications*, Brussels, 2008.

[40] P. Pawar and e. al., "Bridging Context Management Systems in the Ad Hoc and Mobile Environments," in *IEEE Symposium on Computers and Communications*, Sousse, Tunisia, 2009.

[41] B. Guo, L. Sun and D. Zhang, "The Architecture Design of a Cross-Domain Context Management System," in *8th IEEE International Conference on Pervasive Computing and Communications Workshop*, Mannheim, 2010.

[42] B. Guo, D. Zhang and M. Imai, "Toward a cooperative programming framework for context-aware applications," *Personal and Ubiquitous Computing,* vol. 15, no. 3, pp. 221-233, March 2011.

[43] B. Guo, D. Zhang and M. Imai, "Enabling user-oriented management for ubiquitous computing: The meta-design approach," *Computer Networks,* vol. 54, no. 16, p. 2840–2855, November 2010.

[44] B. M. Patel , C. M. Bhatt, H. Vahdat-Nejad and H. B. Patel, "Smart city based on MQTT using wireless sensors," in *Protocols and Applications for the Industrial Internet of Things*, IGI, 2018, pp. 240-263.

[45] R. Kazman, G. D. Abowd, L. Bass and P. Clements, "Scenario-based analysis of software architecture," *The IEEE software journal,* vol. 13, no. 6, pp. 47- 55, 1996.

[46] L. Dobrica and E. Niemela, "A survey on software architecture analysis methods," *IEEE Transactions on Software Engineering,* vol. 28, no. 7, pp. 638- 653, 2002.

[47] P. Hu, J. Indulska and R. Robinson, "An autonomic context management system for pervasive computing," in *Annual IEEE International Conference on Pervasive Computing and Communications* , Hong Kong, 2008.

[48] A. Corradi, M. Fanelli and L. Foschini, "Implementing a Scalable Context-Aware Middleware," in *Proceedings of IEEE Symposium on Computers and Communications*, Sousse, 2009.

[49] K. Henricksen, J. Indulska, M. Ted and S. Balasubramaniam, "Middleware for Distributed Context-Aware Systems," in *International Symposium on Distributed Objects and Applications*, Cyprus, 2005.

[50] K. Michalakis, Y. Christodoulou, G. Caridakis, Y. Voutos and P. Mylonas, "A Context-Aware Middleware for Context Modeling and Reasoning: A Case-Study in Smart Cultural Spaces," *Applied Sciences*, vol. 11, no. 13, p. 5770, 2021.