Towards Enabling Context-as-a-Service for Social Assistive Robotics

Alexandru Sorici
University Politehnica of Bucharest
Bucharest, Romania
alexandru.sorici@cs.pub.ro

Andrei Olaru
University Politehnica of Bucharest
Bucharest, Romania
cs@andreiolaru.ro

Stelian Flonta
Technical University of Cluj-Napoca
Cluj-Napoca, Romania
Stelian.Flonta@aut.utcluj.ro

Enyedi Szilard
Technical University of Cluj-Napoca
Cluj-Napoca, Romania
Szilard.Enyedi@aut.utcluj.ro

Adina Florea
University Politehnica of Bucharest
Bucharest, Romania
adina.florea@cs.pub.ro

***Abstract*—Integrating assistive robots into global-scale management systems comes with a number of challenges and requirements which are pervasive to all applications enabling context-awareness. We argue that the hypermedia model and the agent-oriented paradigm help achieve the vision of Context-as-a-Service.**

We categorize challenges according to context processing concerns and use a scenario to exemplify how the proposed architectural principles help overcome the challenges. A security model is presented, enabling the correct management of information in a hierarchy of agents.

***Index Terms*—context management, context-awareness, context-as-a-service, hypermedia, software agents, social robotics**

I. Introduction

The observation of an exponential increase of internet-connected sensors and actuators has been a continuous remark for the past decade in the Internet-of-Things (IoT) community, as well as at the level of the everyday consumer of internet-enabled services. While true in terms of the increase of devices, the global-scale exploitation of the wealth of data brought forth by such devices still trends much behind.

Some of the most active and promoted domains of interest that make use of *consumer-oriented* IoT devices relate to Ambient Intelligence (AmI). Notable examples include smart environments (e.g. intelligent homes, offices or public venues), as well as eHealth. Active and Assisted Living (AAL) is, in the context of the steep rise of the elderly population, a much needed and well financed research and development domain which sees the use and interplay of many IoT devices and related technologies.

From the perspective of human computer interaction in AmI applications, a recent and growing trend is that of robotic interfaces. Social Assistive Robotics [1], in particular, refers to robots that are meant to assist people in a manner that focuses on social interactions (e.g. speaking, guiding, reminding, observing, entertaining). For the social interaction between human and robot, awareness of the surrounding medium and the current user activity or preferences are required on the part of the robot. As such, social assistive robotics applications fall under the category of consumers that need to interface with the IoT devices that are present in the smart environment.

While there has been steady progress in developing AmI applications, the vision held by the ISTAG group in its proposed scenarios [2], back in 2001, where IoT and Ambient Intelligence (AmI) meet, is still far from reach.

In the Maria scenario [2], for example, a single personal assistant, running on a smartphone, handles interactions with transportation, security, communication and smart home systems. The complexity of these interactions comes from the need to continuously and seamlessly switch between different *contexts*.

Context "is any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves" [3]. Another definition [4] states that context is the *dressing* of a focus, separating the focus of the user or the application from the "dressing" – data which is not vital, but which can improve reasoning on the focus. While these definitions look general, they also offer an explanation for the complexity of deploying the Maria scenario.

In a smart environment application, user focus may change rapidly from the current activity at home or at the workplace, to finding the nearest restaurant conforming to personal preferences given current position on a road, or to monitoring personal health parameters. The consumer of context is mobile and changes focus quickly, while the means to derive the required high-level context information may involve data-driven algorithms (machine-learned data), alongside knowledge-driven heuristics.

This research was supported by grant PN-III-P1-1.2-PCCDI-2017-0734.

[1]The final publication can be found on IEEE Xplore
A. Sorici, A. Olaru, S. Fonta, E. Szilard and A. M. Florea, *Towards Enabling Context-as-a-Service for Social Assistive Robotics*, 2019 22nd International Conference on Control Systems and Computer Science (CSCS), Bucharest, Romania, 2019, pp. 228-235, doi: 10.1109/CSCS.2019.00045.
https://ieeexplore.ieee.org/document/8745138

For a social robot that caters to a group of users (e.g. as guide in an office building, museum or exhibition venue), the change in its own context (e.g. the room it is in, the exhibition item next to which it is placed), as well as that of the users it is helping, is an essential element that needs constant monitoring.

When discussing the use of IoT devices for context-awareness in smart environments, a recent model of gaining access to the data of such devices is that of Sensing-as-a-Service [5]. The model tackles the issue of siloed systems, proposing the existence of several stakeholders: the *sensor owner* (e.g. an individual person, a state or a private organization), the *sensor publishers* (organizations implementing means for sensor data collection), the *extended service providers* (organizations that bring added value by analyzing and aggregating sensor data) and the *final consumer*.

The sensing-as-a-service vision carries with it implicit non-functional requirements for *openness* and *scalability*, since the model allows for and encourages organizations to play several of the mentioned roles at once, thus leading to the development of *context prosumers* (producer *and* consumer at the same time).

From a context management perspective, implementing the sensing-as-a-service model, as well as the ISTAG vision for AmI, brings about a number of challenges, which we group along the main functionality aspects of context management.

Representation and Reasoning. At global scale, a single context model or reasoning/inference technology cannot be expected for all application domains [6]. Mechanisms that facilitate conversion from one model to another, as well as on-demand execution of various context processing procedures have to be considered.

Context Provisioning. This perspective focuses on how context information is supplied to services that can add value to raw data through inference. The challenge in AmI and sensing-as-a-service lies in the fact that the input base can no longer be considered static. A means to *search for* and *select* the most appropriate/available context sources required for a consumer request is needed [7]. Furthermore, the information push behavior of a context producer must be controllable (e.g. altering frequency of updates or sending only on-change), while from a security perspective, the producer must possess means to perform access control.

Query and Dissemination. The large-scale nature of sensing-as-a-service and pervasive AmI means that there must be a means to handle queries/subscriptions with a large selection base. Handling of frequently overlapping queries, as well as result caching should be enabled where appropriate. Another challenge comes from the principle that context information should be consumed as *close* as possible to its production source, thereby increasing the chance of relevance. An organizational scheme is needed that facilitates *local* consumption of context information, while at the same time enabling a *structured dissemination* of context information for remote consumers.

Context Service Deployment. The challenge from the deployment perspective is to find the means to facilitate scalability, search and discoverability of context management services. An organizational scheme that facilitates a *single entry-point* for all context life cycle entities (producers, processors, consumers) is required. The deployment structure must address mobility of context consumers and, more importantly, the *shift* in their focus, which brings a change in the necessary context information.

Compounded with these core issues are two additional concerns which we want to address in this paper. One relates to the means by which a social robot is integrated as a consumer of a Context Management Service and how the information it gathers from the service is used in its internal reasoning cycle. The second one relates to the issue of privacy and control of access to sensitive context information (e.g. the location of a user, the availability status), which is of importance especially when such information is made available to autonomous robotic platforms.

The hypermedia model is an appropriate architecture for such systems, ensuring the development of open, scalable and evolvable systems, which answer well to the presented challenges. Moreover, the agent-oriented paradigm helps moving the development focus from a system-wide perspective to one that is centered on individual entities.

In Section II we explain why existing work towards context management middleware does not currently fulfill the requirements listed previously. Section III details our argument that the fundamental engineering techniques that sustain the Web (e.g. hypermedia-driven interactions, RESTful [8] protocols, knowledge graphs, publish/subscribe mechanisms) can be coupled with the organizational and behavioral principles stemming from the Multi-Agent System (MAS) literature to obtain architectural specifications for Context-as-a-Service (CaaS) systems supporting application areas as large-scale as AmI and Sensing-as-a-Service. We further discuss how social robots can act as both producers and consumers of context information in CaaS systems. In Section IV we describe a method for context encryption/decryption in a deployment that follows a hierarchical distribution of context management elements. Finally, section V concludes the paper.

II. Related Work

Recent survey papers [6], [9], [10] perform a good review of several of the existing context management middleware. While each solution covers many aspects that are essential for the implementation of a context management life cycle, most of the existing proposals either focus on specializing for a particular application domain (e.g. SeCoMan [11] – location, CoCaMAAL [12] – eHealth), or on the engineering effort for one of the focus requirements outlined in the introduction. CA4IOT [7], for example, defines an architecture and a reasoning process that enables a context-aware selection of the best sensors, with respect to a complex user query. It thereby offers a reference regarding the challenge of managing a large selection base for context input. However, the middleware focuses mostly on this aspect and the application scenario was limited to the domain of smart agriculture.

CONSERT [13] is a context middleware that uses the multi-agent programming paradigm to design autonomous management for each of the context life cycle entities. It further proposes a deployment mechanism that follows an explicit organizational scheme, which uses *Context Dimensions* (well-defined focus points of a consumer that dominate the rest of the perceived context information - e.g. location, activity, role in an organization) to connect the different instances of context agents. While conceptually it answers to the challenges enumerated under the query/dissemination and service deployment categories, it does not meet the requirements for context provisioning and multi-modal, on-demand reasoning capabilities. Furthermore, the middleware has not undergone any real-world scenario validation.

FIWARE[1] is an open-source cloud platform aiming to provide reliable means for developing open, collaborative and mature ecosystems of smart, context-aware, internet-scale applications. It comes close to the set of requirements outlined in the introduction.

The context management reference architecture of FIWARE[2] is based on the interaction between key Generic Enablers (GE) that facilitate the development of a context processing pipeline. The central element is the *Context Broker* GE, which defines the context model and allows for context information storage, update and look-up. The *Context Broker* can be connected to a series of other GEs which complement its functionality. An *IoT Broker* plays the role of an intermediary between raw information coming from IoT devices and the *Context Broker* where the information needs to be stored. To facilitate discovery of IoT devices and their capabilities, each *IoT Broker* communicates with an *IoT Discovery* GE. Short term reasoning and long term analysis of context events are enabled by the connection of the *Context Broker* to *Complex Event Processing* and *Big Data Analysis* GEs respectively.

A core feature of the FIWARE platform is its reliance on web-based standards for enabling the communication between GEs. The Open Mobile Alliance (OMA) NGSI-9 and NGSI-10 [14] are two specifications for RESTful interaction protocols that define means to semantically describe the capabilities of IoT devices (NGSI-9) and the exchange of context content itself (e.g. publishing, updating, querying – through NGSI-10).

The modular, GE-based architecture of FIWARE and its standardized communication interfaces render it a good candidate for the requirements presented in the introduction. In a demo implementation[3] the federation of several *Context Broker* GEs is showcased, exemplifying how both aggregation and load balancing of context information can be enabled.

However, while the structural elements are there, the FIWARE platform proposes no conceptual means of organizing the connection of its GEs, so as to enable large-scale, automated discoverability and dissemination of information. The *Context Broker* interconnection has to be manually specified and is predefined at development time. Furthermore, the existing *Context Broker* implementation does not handle complex (composite) query execution, nor can the *Complext Event Processing* GE manage on-demand context reasoning.

Nonetheless, the takeaway point of FIWARE is that the defined RESTful interfaces contain in their specification the potential to enable all the types of interaction outlined as required. Either upgrading the existing GEs, or delivering fresh implementations compatible with the NGSI-9 and NGSI-10 specifications has the potential to overcome the challenges.

In the field of social robotics, there are already projects that look to enable context-aware decision making of the robotic units. Many among these [15], [16] are looking into Human Activity Recognition to better understand the current context of the user(s) the robot is supposed to help. With respect to this, the robots act as context producers. However, the STRANDS project [16], for example, is looking into building systems that enable long-term autonomy of social robots. Ensuring the context-awareness (from both consumer and producer perspectives) of a mobile robotic unit is therefore essential for such a goal (being aware of both its environment, as well as the state of the users therein).

III. Hypermedia-driven CaaS

While the exact mechanisms and methods for context management are highly application-dependent, the multitude of entities and stakeholders in a global-scale system for the management of context information requires a suitable underlying architecture, which can answer to the outlined challenges and also fulfill the non-functional requirements of openness and scalability.

In the following we first lay out a set of architectural principles that have their grounding in the technologies that support the Web. We then enter into more details regarding proposed deployment and query handling mechanisms. The latter permits us to further explain how social robots are integrated as consumers of CaaS services, while also considering access control methods.

A. Architectural principles

The purpose of this architectural specification is to lead to the development of systems that support a large-scale *Context-as-a-Service* (CaaS) view, where context management can be offloaded to a network of *context life cycle entities*, each working under its own policy on context production, reasoning, or query management/dissemination.

Such an underlying architecture is the *resource-oriented* model of the Web and the *hypermedia-driven interactions* it supports (using the HTTP protocol), which we believe would contribute greatly to the implementation of the CaaS paradigm. Moreover, the *agent-oriented paradigm* can contribute to modeling individual entities in the CaaS ecosystem, by focusing on a perspective centered on individual participants, rather than on the system as a whole.

Hypermedia underpins the World Wide Web as a network of uniquely identifiable (by means of URIs) *resources* interconnected through *web services*. However, to truly exploit the

[1]FIWARE (https://www.fiware.org/, https://www.fiware.org/developers/catalogue/)

[2]FIWARE Context Management Architecture (https://goo.gl/Yvwv5A)

[3]FIWARE Lab Demo Platform (https://goo.gl/CbzMyt)

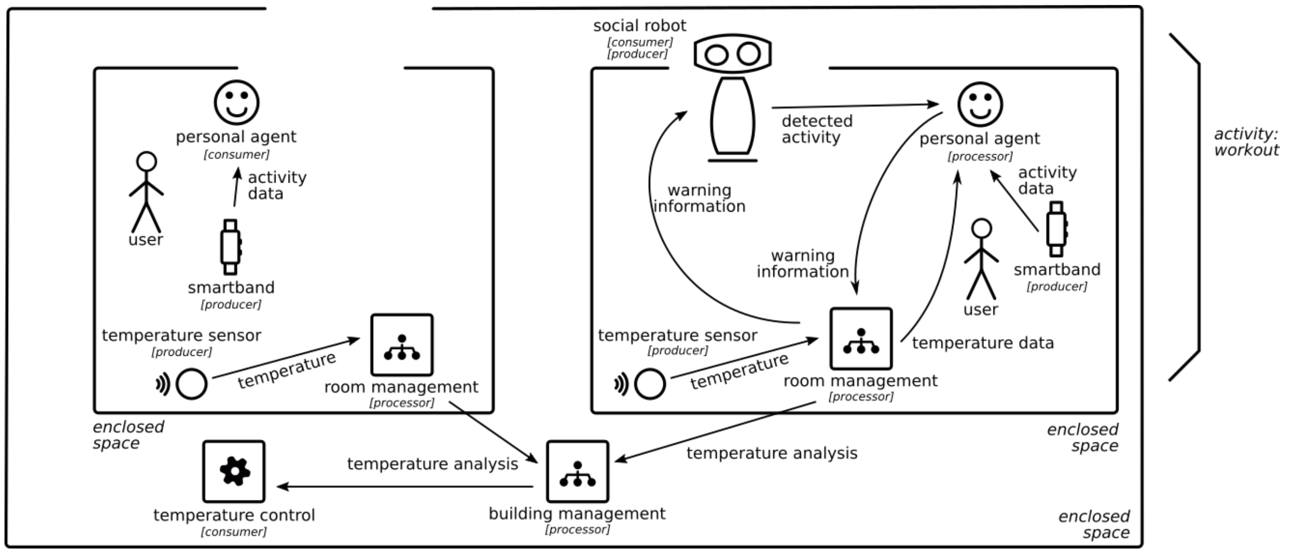

Fig. 1. An example of context management services enabling a scenario in which a social robot is both a producer of activity information and a consumer of information regarding warnings in its current enclosed space. The personal agent gathers activity information and environmental data to issue warnings about potential dehydration. Temperature readings are also used for the temperature control of the building.

resource-oriented nature for the modeling of context processing entities, their *state* (e.g. context production capabilities, query subscription results, active reasoning mechanisms) has to be explicitly (semantically) described and the means to *change their state* should be clearly identifiable, following the principle of *hypermedia as the engine of application state* (HATEOAS).

In terms of the **context service deployment** challenges, the use of a hypermedia environment and RESTful [8] interactions is not sufficient on its own to facilitate discoverability and efficient search / query propagation. In the architectural specification we envision, context life cycle entities must have a structured means by which to determine their *required* connections. We draw inspiration from the vision of Socio-Technical Networks (STNs) [17] and the proposed deployment organization scheme of CONSERT [13]. All URIs referring to connections between entities of an STN are *typed relations*, meaning that there is a clear semantic attached to them (e.g. ownership, membership, collocation). In CONSERT, *Context Dimensions* and *Context Domains* are concepts defining key *properties* and *values* that relate a context consumer to context sources that might be of interest to his current *focus*. They furthermore help organize the space of deployed context life cycle entities into a tree-like hierarchy, which facilitates their management and indicates clearly the context sub-model of which they are responsible.

In the example from Figure 1, where an elderly person, living in a care facility, is performing a physical exercise (workout) session in a room, there are two dimensions of context distinguishable: a *spatial* one and an *activity*-based one. From the spatial perspective, the ⟨`building manager`⟩ keeps track of a hierarchy of two domains – particular *rooms*, which are ⟨`includedIn`⟩ the *building* and where the two elderly users reside.

In our example, the care facility owns a ⟨`social robot`⟩, which is able to monitor the users (e.g. detect a defined set of activities carried out by a person, based on the robot's video input), as well as notify them verbally about specific events (e.g. a warning regarding their health state, a reminder for medication intake). In the context model, an activity has a place where it is carried out, a relation which is modelled semantically and explicitly by both the robot (which acts as a *context producer* by detecting the activity and where it is carried out), as well as the ⟨`personal agent`⟩ (which acts as a *context consumer*) on behalf of the user. By this relation, the ⟨`personal agent`⟩ is able to use a discovery mechanism through which it looks explicitly for a *context processor* responsible for the space in which the workout activity takes place. The ⟨`personal agent`⟩ is then able to launch a query for the current temperature in the room. The result of this query is forwarded to the ⟨`workout activity manager`⟩, where this *context processor* uses it, together with data from the ⟨`smart watch`⟩, to infer whether the user is subject to dehydration or not. In case the user is dehydrated the ⟨`personal agent`⟩ looks for means to inform the user of this fact. The agent knows that the ⟨`smartphone`⟩ on which it runs can notify the user through a text message. However, since being engaged in a workout activity means the user might not have the phone with them, the ⟨`personal agent`⟩ queries the ⟨`room`⟩ (and if needed, the ⟨`building manager`⟩) for context information about entities having the capability to notify the user verbally. Since the ⟨`social robot`⟩ publishes such a capability and is currently in the room, the ⟨`personal agent`⟩ opts to use the robot for notifying the user about drinking some water.

The critical observation regarding the *deployment perspective* in the given example is that the ⟨`personal agent`⟩ *is* or *becomes* aware of two *Context Domains* (the jogging activity and the agricultural field location) and is able to connect to the relevant *context processors* to retrieve context that "dresses" the current focus of the user (his well-being while working out). This is made possible by the explicit, *typed* spatial inclusion and current activity relations which relate the ⟨`personal agent`⟩ with the ⟨`workout activity manager`⟩ / ⟨`building manager`⟩. The physical runtime of the latter is not important (it may be a cloud-based deployment or a local server arrangement) as long as the typed relations enable discovery / search of the relevant URIs.

From a **query / dissemination perspective**, a *Context Dimension*-based organization of *context processors* facilitates query routing. In our example, a manner of handling queries similar to the one described by Mayer et al. [18] is proposed. Both the smart thermostat and the personal agent subscribe for temperature updates to the ⟨`building manager`⟩ *processor*. Being mobile consumers, they will request temperature updates for given locations (rooms) within the building. If temperature analysis is managed by different *processors*, queries sent to the processor for the entire building must be routed to the one corresponding to current consumer location. The *Context Dimension* / *Context Domain* based organization of the *processors* enables the same type of query routing mechanism as described in [18] (including *Context Domain*-based range queries). Furthermore, modeling a subscription query for temperature updates as a web resource, identifiable by a URI, enables incremental updates of the answers, load-balancing through web-based publish/subscribe implementations (e.g. WebSub[4]), as well as the use of caching mechanisms, since it is expected that temperature updates will be given on an on-change basis, rather then on a time-based one.

An *agent-oriented view* of the context life cycle elements in our example is of relevance from the **context provisioning perspective**. Software agents are entities that are able to act autonomously in their environment in order to achieve or maintain their goals. Among the main features of agents that are part of a multi-agent system are autonomy, reactivity, proactivity and social interaction.

The owner of the care facility is the one that deployed the temperature sensors and the ⟨`building manager`⟩ and ⟨`room manager`⟩ *processor(s)* which analyze the temperature. The ⟨`personal agent`⟩ may be able to discover the existence of the *processors* but this may not guarantee that he has the *right* to access their information. This use case can be viewed as a sensing-as-a-service instance, where the ⟨`building / room manager`⟩ is an *extended service provider* and the ⟨`personal agent`⟩ a *final consumer*. If we model/design the life cycle entities as *agents*, then access to the temperature data can be subject to an automated, agent based negotiation. The ⟨`building / room manager`⟩ *processor* can make use of goal-based policies, that require it to, for example, accept query requests only for a specific user activity context, such as the workout one. Apart from a semantics based access control, the web-based environment of the *context processor* agents enables them to make use of existing techniques for authentication and authorization.

The agent-oriented view is suitable for the multi-modal and on-demand **context reasoning perspective** as well. In our example, the ⟨`workout activity`⟩ *processor* may by default provide reasoning methods for user well-being analysis based on smart-watch data alone. When the ⟨`personal agent`⟩ is able to access additional room temperature information, it may request that the ⟨`workout activity`⟩ processor executes a knowledge-driven heuristic that includes temperature data alongside the previous smartwatch information to gauge the well-being state. Since such interactions between the *consumer* and *processor* context entities resemble a service request, more than a state change for the *processor*, technologies such as gRPC[5] can be used to enable them.

B. Deployment and Query mechanism

Social robots are expected to be able to interface with many IoT devices and to obtain the context information that is relevant to the current or an expected future user situation. For example, if the robot detects that the user is engaged in a form of physical exercises at home, it is convenient for it to be able to automatically subscribe to sources providing context information for relevant health and home monitoring parameters (e.g. heart rate, room temperature) such that it may proactively offer notifications or warnings to the user while he is carrying out the exercises.

Deployment. To this end, facilitation of context information discovery is an important concern, because it is expected that there will be many social robot types and many sources for health and home monitoring information.

We discussed previously that a structured organization of context information by means of typed relations is a necessary step to enable discovery, navigation and access control. To achieve this, we propose an explicit indexing mechanism based on the *Context Dimension* and *Context Domain* organization elements considered in [13]. All context life cycle entities are indexed under one or more *Context Domains*, that are structured along a *Context Dimension*. If no such index is provided, then the information provided / managed by the context life cycle entities is considered to be part of the `all` *Context Domain*.

In the example in Figure 1 there are three relevant *Context Dimensions* which relate to the *spatial localization* and their *current activity*, as well as to the *maintenance activity* carried out by the smart thermostat in the building. Consider the dimensions defined by the properties `locatedIn(User, EnclosedSpace)`, `engagedIn(User, Activity)` and `engagedIn(Application, MaintenanceActivity)`. Four *Context Domains* are defined based on these dimensions: an instance of `Activity` (`Workout`) for the user, an instance of `MaintenanceActivity` for the smart thermostat and two instances of `EnclosedSpace`, the `FacilityRooms` in which the users are located.

[4]WebSub W3C Recommendation (https://www.w3.org/TR/websub/)

[5]gRPC open-source universal RPC framework (https://grpc.io)

In keeping with the sensing-as-a-service principle, the owner of the temperature sensors can define under which *Context Dimensions* and *Context Domains* the information can be indexed. The indexing can be performed explicitly, or it may be inferred based on other existing typed relations. For example, assume sensor information is initially submitted to the ⟨`building manager`⟩ processor based on a `ownedBy(Sensor, SensorOwner)` *Context Dimension*, where the `SensorOwner` is a subclass of `User`. If there exists then context information that states that `managesActivity(SensorOwner, TemperatureMgmt)`, the reasoning policy of the ⟨`building manager`⟩ can lead it to infer that this information may be forwarded to all consumers who define an interest in the `engagedIn(Application, MaintenanceAcitivty)`, where the *Context Domain* for the MaintenanceActivity is defined by the `TemperatureMgmt` instance. A complete discussion of this mechanism is however out of the scope of this paper.

The essential part of the indexing mechanism is that context information comes clearly defined with a *scope* in which it is considered as relevant (e.g. temperature information for a person near a agricultural field only becomes relevant if the person is engaged in a workout activity). It furthermore means that, using *Context Dimensions* and *Context Domains* consumers can automatically search and find which information is available/relevant given the situation of the users who employ those context consumers.

Querying. The organization of context information in a structured manner allows exploitation of more efficient query/dissemination mechanisms, as well. In [19] we mention that *Context Domains* can, by default, be exploited in a *flat* structure (i.e. as different instances of the *object* part of a *Context Dimension*). However, if the *Context Domains* have relations which favor the construction of a hierarchy (e.g. `includedIn(EnclosedSpace, EnclosedSpace)`, `partOf(Activity, Activity)`) then the ensuing organizational structure can be exploited in terms of dissemination and querying.

In Figure 2 an example of a *Context Domain* hierarchy based on the `locatedIn(User, EnclosedSpace)` *Context Dimension* is presented. The object part of the *Context Dimension* allows for the existence of a hierarchy inducing relation (`includedIn(EnclosedSpace, EnclosedSpace)`). The latter is used to define the relationship between the *Context Domain instances*.

In compliance with the locality principles for context information consumption, a hierarchical organization allows a child ⟨`context processor`⟩ to forward information it receives to its parent, but not the other way around. In so doing, a type of *granularity* of the disseminated context information can be obtained. For example, a consumer who is subscribed to the ⟨`context processor`⟩ of an instance of a `CareFacility` may obtain information about the localization of people at the level of the *building*, as opposed to individual `FacilityRooms` (i.e. know if someone is in the building, but not in which room).

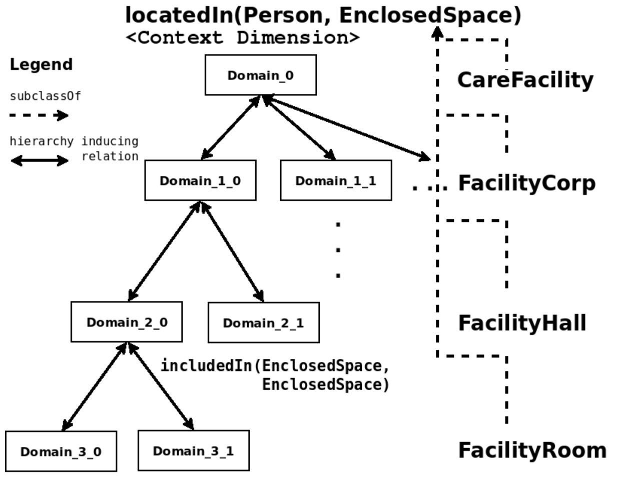

Fig. 2. *Context Domain* hierarchy example showing the *Context Dimension* and the relation that induces the hierarchy. Notice the `subclassOf` relation that may exist in between different levels of the hierarchy

From a query/subscription perspective, the *Context Domain* hierarchy enables both *exact domain*, as well as *domain range* queries [19]. This mechanism is similar to the one described in [18]. A query can have a *scope* from which it expects to receive an answer. In *exact domain* queries the scope is defined by a single *Context Domain* (e.g. the `Domain_3_0 FacilityRoom`). In a *domain range* query, the scope has upper and lower bounds, expressed in terms of the existing *Context Domain* hierarchy (e.g. any temperature information collected in between `Domain_0` and `Domain_3_0`). Note that *domain ranges* may be expressed between individual instances, as well as between the *types* of the instances (thereby increasing the range).

C. Social Robots as CaaS Prosumers

In our example, the social robot is the perfect example of a context *prosumer*, i.e. acting as both consumer and producer of context information. From the production perspective, we distinguish two types of context information. On the one hand, the robot acts as mobile sensor, which is able to detect actions and activities of a user by means of analyzing the video feed it receives. That is, the robot provides context information characterizing an external entity (the user). On the other hand, the robot provides context information characterizing itself, in the form of the *capabilities* that it has, such as being able to deliver voice notifications.

As a context consumer, the robot uses knowledge that the user it supervises has a new context focus (that of the workout, expressed through the ⟨`engagedIn(User, Activity`⟩ *Context Dimension*), such that it can now search for and subscribe to context information indexed under the corresponding *Context Dimension* and *Context Domain*.

It is worth noting that, from a technical perspective, the integration of a social robot as a context prosumer is not difficult if the CaaS system with which it is being integrated follows the hypermedia-oriented architectural principles outlined previously. The reason is that all modern high-level robotics frameworks contain libraries that facilitate communication over the web. The challenge from the robotics perspective is to facilitate the integration of context information as *events* that affect the robot reasoning life cycle. With respect to this, modeling the robot itself as an agent (more precisely, as a BDI [20] agent) presents an advantage, as the BDI paradigm specifically installs a reasoning life cycle where events *alter* the beliefs of the agent or inform the selection of the next action/decision to take.

IV. Access Controlled Context Dissemination

In a complex context management system, it is necessary that sensitive, personal or personally identifiable information [6] is only accessible to the indented recipients. For instance, in the scenario illustrated in Figure 1, the user in the room to the right (say, user A) will only want its activity data to be available to the user in the room to the left (say, user B), only if user A has explicitly permitted it. Otherwise, even if the data may circulate in the system (e.g. for routing purposes), it must not be readable for unauthorized entities.

Since many *Context Dimensions* are naturally hierarchical (such as spatial or activity-related context), an agent hierarchy is adequate for organizing a context-aware system [21].

In order to protect the context information managed by the agents, the data can be encrypted with partial keys, by that agent, with its own key. Although the agents have the means of communicating with each other, neither of them can decrypt the data belonging to the others, since the key that one agent possesses can decrypt only its own data. However, a node at a higher level can access the data of its "subordinate" agents and can decipher them with its own key, without needing all the partial keys from the subordinate agents.

Function trees. A function tree, where the nodes are functions, can be generated starting from a function of multiple variables. In the following, we will describe an algorithm for generating a function tree.

We choose the numbers $p_1, p_2, \ldots, p_l \in N^*$, for each $k \in \{1, 2, \ldots, l\}$ and we define the sets: $D_k \overset{\text{def}}{=} \overline{1, p_l} \overset{\text{def}}{=} \{1, \ 2, \ldots, p_k\}$. Additionally, for each $k \in \{1, 2, .., l\}$ we choose the functions $g_k : N^* \to N^*$. Let A be a finite cyclical group $(Z_q \ , \ \bullet \ )$. If $x_1, \ x_2, \ldots, x_l \ \in A$ distinct elements, we define the functions like so:

$F^0 : D_1 \times D_2 \times \ldots D_l \to A$

$F^0 \, (m_1, \ m_2, \ldots, m_l \ ) = {x_1}^{g_1(m_1)} {x_2}^{g_2(m_2)} \ldots {x_1}^{g_l(m_l)}$

$F^1_{s_1} : D_2 \times D_3 \times \ldots D_l \to A$

$F^1_{s_1} \, (\ m_2, \ldots, m_l \ ) = F^0 \, (s_1, \ m_2, \ldots, m_l \ ),$

where $s_1 \in D_1$

$F^2_{s_1, s_2} : D_3 \times D_4 \times \ldots D_l \to A$

$F^2_{s_1, s_2} \, (\ m_3, \ldots, m_l \ ) = F^0 \, (s_1, \ s_2, \ldots, m_l \ ),$

where $s_1 \in D_1, \ s_2 \in D_2$

$F^3_{s_1, s_2, s_3} : D_4 \times D_5 \times \ldots D_l \to A$

$F^3_{s_1, s_2, s_3} \, (\ m_4, \ldots, m_l \ ) = F^0 \, (s_1, \ s_2, s_3, \ldots, m_l \ ),$

where $s_1 \in D_1, \ s_2 \in D_2, \ s_3 \in D_3$

$\ldots$

$F^{l-1}_{s_1, s_2, s_3 \ldots s_{l-1}} : D_l \to A$

$F^{l-1}_{s_1, s_2, s_3 \ldots s_{l-1}} \, (\ m_l \ ) = F^0 \, (s_1, \ s_2, s_3, \ldots, s_{l-1}, m_l \ ),$

where $s_1 \in D_1, \ s_2 \in D_2, \ s_3 \in D_3, .., \ s_{l-1} \in D_{l-1}$

Finally, we get the following values that are elements of A: $F^l_{s_1, s_2, s_3 \ldots s_l} = F^0 \, (s_1, \ s_2, s_3, \ldots, s_l)$ where $s_1 \in D_1, \ s_2 \in D_2, \ s_3 \in D_3, \ldots, \ s_l \in D_l$.

Each function or value has, with respect to the notation, an exponent and an index. The exponent is a natural number from the set $\{0, 1, 2, .., l\}$, and the index is a vector $s \overset{\text{def}}{=} s_1, \ s_2, s_3 \ldots s_j \overset{\text{def}}{=} (s_1, \ s_2, s_3, \ldots, s_j) \in D_1 \times D_2 \times \ldots D_j \overset{\text{def}}{=} D$. Using the notations, we can build a tree:

- The root is the function F^0 at level 0.
- The functions and values are on the levels corresponding to their exponents.
- A function is the descendant of another, parent function, if their indices are:
  $(s_1, \ s_2, s_3, \ldots, s_j, \ s_{j+1} \ ) \in D_1 \times D_2 \times \ldots D_j \times D_{j+1}$
  $(s_1, \ s_2, s_3, \ldots, s_j) \in D_1 \times D_2 \times \ldots D_j$, respectively
- The leaves are values of F^0 .
- The tree has arcs oriented from the lower level towards the upper level.

It is important for the tree to be directional, because this is the way the access permissions are represented.

Figure 3 exemplifies this tree generation algorithm.

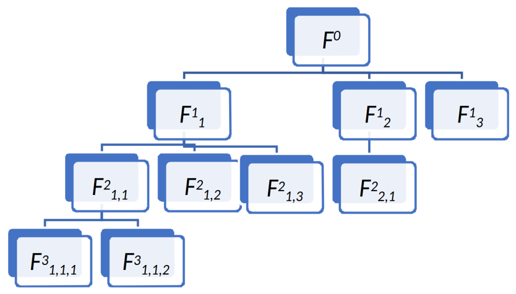

Fig. 3. A function tree.

If A is a cyclical group where the discrete logarithm problem is difficult to solve, knowing the value or formula for the descendant does not yield the formula for the parent. This fact results from the way the functions are defined. Vice versa, knowing the formula for the parent, one can immediately have the formula for the descendant, by simply replacing numbers in the formula of the parent. Thus, all nodes can be traversed, exploring the path from the root to the leaves.

A subtree composed of merging paths that start from the root and end in leaves is a system of private, hierarchical keys. By allocating these keys, in a bi-univocal fashion, to a system of agents, one can ensure access to the data held by an agent α by an agent β if and only if α is a descendant of β .

Therefore, to control access to the information stored by an agent community, one may use a hierarchical encryption

[6] PII(https://en.wikipedia.org/wiki/Personally_identifiable_information)

system with public keys. Starting from the access right of each agent, we build a tree of the agents, this tree having arcs oriented from the lower levels towards the higher levels, according to the access credentials. Based on the tree of agents, we generate a function tree, of the type described above. All information is encrypted / decrypted using the values from the leaves as private keys for the ElGamal algorithm [22].

If the leaves of the agent tree are not all on the same level, the tree is completed with conformity nodes as in Figure 4. A trusted center generates the keys for the algorithm and distributes them to the agents. This trust authority can be A^0 or another trusted entity.

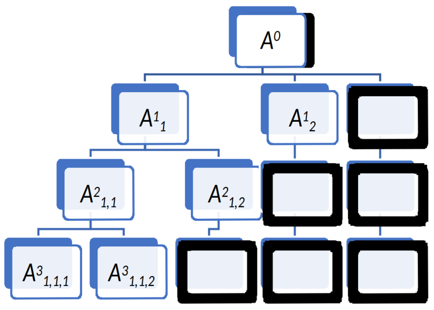

Fig. 4. A function tree with conformity nodes.

We generate a function tree, based on the agent tree. The values in the leaves are the private keys associated with agents. Each agent can encrypt with the ElGamal algorithm, using the public keys for each leaf. Accordingly, each agent can decrypt, using its own private key, all the information belonging to a subordinate agent.

It is notable that the key network can be easily replaced with a new one. Generating the tree means choosing the function F^0 from which we start generating all the keys, and defining the function F^0 using an algorithm with random variables, which can be done by a trust authority, even an agent.

V. Conclusions

We envision a global-scale CaaS deployment as a conglomerate of computing entities obtaining producing, processing, and consuming context. The multitude and diversity of entities inevitably leads to a highly *heterogeneous* system, that needs to be *loosely coupled*.

Not only does the hypermedia model naturally support openness and scalability, but it also answers well to the challenges of CaaS deployment, while also enabling easy conceptual integration with social robotics frameworks.

We also argue that context management systems would benefit from an agent-oriented approach to the implementation of their individual components. What we can take from the MAS domain and use in CaaS is not necessarily how entities can be implemented as agents, but the *agent-oriented perspective*. Moving from a system-wide view to an entity-centered helps designing components that are able to work in highly heterogeneous systems.

References

[1] D. Feil-Seifer and M. J. Mataric, “Defining socially assistive robotics,” in *Rehabilitation Robotics, 2005. ICORR 2005. 9th International Conference on*. IEEE, 2005, pp. 465–468.
[2] K. Ducatel, U. européenne. Technologies de la société de l’information, U. européenne. Institut d’études de prospectives technologiques, and U. européenne. Société de l’information conviviale, *Scenarios for ambient intelligence in 2010*. Office for official publications of the European Communities Luxembourg, 2001.
[3] G. D. Abowd, A. K. Dey, P. J. Brown, N. Davies, M. Smith, and P. Steggles, “Towards a better understanding of context and context-awareness,” in *International symposium on handheld and ubiquitous computing*. Springer, 1999, pp. 304–307.
[4] J. Brézillon and P. Brézillon, “Context modeling: Context as a dressing of a focus,” in *Modeling and Using Context*, ser. Lecture Notes in Computer Science, B. Kokinov, D. C. Richardson, T. R. Roth-Berghofer, and L. Vieu, Eds. Springer Berlin Heidelberg, 2007, vol. 4635, pp. 136–149.
[5] C. Perera, A. Zaslavsky, P. Christen, and D. Georgakopoulos, “Sensing as a service model for smart cities supported by internet of things,” *Transactions on Emerging Telecommunications Technologies*, vol. 25, no. 1, pp. 81–93, 2014.
[6] M. Knappmeyer, S. L. Kiani, E. S. Reetz, N. Baker, and R. Tonjes, “Survey of context provisioning middleware,” *IEEE Communications Surveys & Tutorials*, vol. 15, no. 3, pp. 1492–1519, 2013.
[7] C. Perera, A. Zaslavsky, P. Christen, and D. Georgakopoulos, “Ca4iot: Context awareness for internet of things,” in *Green Computing and Communications (GreenCom), 2012 IEEE International Conference on*. IEEE, 2012, pp. 775–782.
[8] R. T. Fielding and R. N. Taylor, *Architectural styles and the design of network-based software architectures*. University of California, Irvine Irvine, USA, 2000, vol. 7.
[9] C. Perera, A. Zaslavsky, P. Christen, and D. Georgakopoulos, “Context aware computing for the internet of things: A survey,” *IEEE communications surveys & tutorials*, vol. 16, no. 1, pp. 414–454, 2014.
[10] X. Li, M. Eckert, J.-F. Martinez, and G. Rubio, “Context aware middleware architectures: survey and challenges,” *Sensors*, vol. 15, no. 8, pp. 20 570–20 607, 2015.
[11] A. H. Celdrán, F. J. G. Clemente, M. G. Pérez, and G. M. Pérez, “Secoman: A semantic-aware policy framework for developing privacy-preserving and context-aware smart applications.” *IEEE Systems Journal*, vol. 10, no. 3, pp. 1111–1124, 2016.
[12] A. Forkan, I. Khalil, and Z. Tari, “Cocamaal: A cloud-oriented context-aware middleware in ambient assisted living,” *Future Generation Computer Systems*, vol. 35, pp. 114–127, 2014.
[13] A. Sorici, G. Picard, O. Boisser, and A. Florea, “Multi-agent based flexible deployment of context management in ambient intelligence applications,” in *International Conference on Practical Applications of Agents and Multi-Agent Systems*. Springer, 2015, pp. 225–239.
[14] S. Krco, B. Pokric, and F. Carrez, “Designing iot architecture(s): A european perspective,” in *Internet of Things (WF-IoT), 2014 IEEE World Forum on*. IEEE, 2014, pp. 79–84.
[15] P. Lasitha and S. Kodagoda, “Human activity recognition for domestic robots,” in *Field and Service Robotics*. Springer, 2015, pp. 395–408.
[16] N. Hawes and et al, “The STRANDS Project: Long-Term Autonomy in Everyday Environments,” *IEEE Robotics and Automation Magazine*, vol. 24, no. 3, pp. 146–156, 2017.
[17] A. Ciortea, A. Zimmermann, O. Boissier, and A. M. Florea, “Hypermedia-driven socio-technical networks for goal-driven discovery in the web of things,” in *Proceedings of the Seventh International Workshop on the Web of Things*. ACM, 2016, pp. 25–30.
[18] S. Mayer, D. Guinard, and V. Trifa, “Searching in a web-based infrastructure for smart things,” in *Internet of Things (IOT), 2012 3rd International Conference on the*. IEEE, 2012, pp. 119–126.
[19] A. Sorici, “Multi-agent based context management middleware in support of ambient intelligence applications,” Ph.D. dissertation, Saint-Etienne, EMSE and Bucharest, UPB, 2015.
[20] A. S. Rao, M. P. Georgeff *et al.*, “Bdi agents: from theory to practice.” in *ICMAS*, vol. 95, 1995, pp. 312–319.
[21] A. Olaru, A. M. Florea, and A. El Fallah Seghrouchni, “A context-aware multi-agent system as a middleware for ambient intelligence,” *Mobile Networks and Applications*, vol. 18, no. 3, pp. 429–443, June 2013.

[22] T. ElGamal, “A public key cryptosystem and a signature scheme based on discrete logarithms,” *IEEE Transactions on Information Theory*, vol. 31, no. 4, pp. 469–472, 1985.