ALPACA: A Decentralized, Privacy-Centric and Context-Aware Framework for the Dissemination of Context Information

Florian Dorfmeister, Sebastian Feld, and Claudia Linnhoff-Popien
Mobile and Distributed Systems Group
Ludwig-Maximilians-Universität München
Munich, Germany
Email: {florian.dorfmeister, sebastian.feld, linnhoff}@ifi.lmu.de

***Abstract*—With the ongoing rise of smartphones as everyday mobile devices and their steadily increasing amount of sensing and communication capabilities, we are on the brink of a subtle, widespread adoption of context-aware computing techniques into our daily lives. Focusing on functionality and performance, the majority of existing architectures for managing context information typically deploy central components for collecting, analyzing and distributing its users' up-to-date data. However, preservation of users' privacy needs remains a crucial factor for such systems' acceptableness. Inspired by existing works on privacy in context-aware applications and the authors' beliefs in the necessity to put users back in control, this article adopts a privacy-centric perspective and presents ALPACA: A novel approach for modeling and managing a user's rich context information in a user-centric and privacy-preserving way fit for a multitude of different usage scenarios. To this end, this article offers a general conceptual mapping of a user's privacy needs to distinct layers. Based on this conceptualization we introduce a privacy-centric approach for modeling this information. Additionally, we propose a context-aware mechanism for the definition of context-dependent release triggers in order to enable fine-grained control over the disclosure of sensitive information. Finally, we present the components of the proposed system architecture, explain how they interact with each other and discuss how our framework can be integrated into a modern mobile operating system.**

Keywords-context modeling, context-awareness, privacy-centric design, context-dependent privacy policies, context obfuscation.

I. INTRODUCTION

Both the acquisition of a user's current context, e.g., by applying activity classification or other reasoning techniques to her smartphone's sensor readings as well as approaches for effectively modeling and managing context information are active areas of research. At the same time, however, also the leakage of sensitive personal data is an issue of substantial interest on both a legal, social, and technical level. The ongoing scandalization of agencies' large-scale data collection on the Internet hopefully affords an appropriate, albeit unpleasant, opportunity to sensitize the public for these privacy issues. The latter get additionally tightened as, due to the popularity of smartphones, we are heading towards a full supply of small electronic devices with broadband Internet access, extensive sensing and computing capabilities. The privacy problems naturally exist regardless of extensive eavesdropping, as the number of applications that silently collect context information and send it to the Internet is legion. Privacy-aware users usually face an all-or-nothing option at install time and, once approved, there is no control over an application's usage of personal or context information at all. In order to contribute to the tackling of some of the technical aspects of these problems, we originally published the concepts and design of the *Layered Architecture for Privacy-Assertions in Context-Aware Applications* (ALPACA) in [1]. The main goal of this privacy-centric framework for managing context information is to turn a user's mobile device into a personal data vault that only the user has full access to as well as to enable fine-grained access control mechanisms for the dissemination of sensitive information. In this article, we will provide an updated and more thorough view at the different components and present new additions to the framework. In particular, we will go into detail on our ontology-based modeling approach and the context-aware trigger mechanism, which allows for the definition of context-dependent privacy policies. We will give more insight into our system's key components, communication protocol, and entity authentication and outline its applicability to the Android operating system and different usage scenarios.

One can think of many beneficial use cases for context-aware applications, such as proactive route planning services taking into consideration the current traffic volume and a user's appointment schedule, smart mechanisms automatically adjusting a phone's audio profile based on the user's current occupation or context-aware online social networking, e.g., buddy finder apps notifying the user about friends in proximity [2], [3]. In addition, there are applications such as the *SmartBEEs* context-aware business platform [4], which do not act based on a single-user or peer-to-peer basis, but leverage the combined knowledge of multiple users' current contexts and their surroundings' state, e.g., for business process optimization. In order to prevent unintentional disclosure of personal data, users must yet be able to control what kind and resolution of context information applications are able to collect at any time. Enabling the acquisition of a user's context information with the help of her smartphone's sensors and eligible reasoning mechanisms in return also enables spying on this person. Thereby "one person's sensor is another person's spy", as [5] puts it. For the enabling of high levels of service quality, the algorithms used for context acquisition typically aim at maximizing resolution, freshness, and accuracy of their findings. When talking about preserving a user's privacy, however, different and partly even contradicting objectives are to be

pursued. For example, in many situations it might be perfectly sound to deliberately reduce the resolution of a piece of context information before sharing it with others in order not to reveal too much. Many approaches for managing context information focus on the generation, modeling, processing, and efficient distribution of the latter as well as the realization of context-aware applications built thereon. Most of these works take a primarily functional view on their system and try to maximize parameters such as classification accuracy, availability, and scalability. Some works also argue that not only the usability and utility of context-aware applications are paramount for a wide acceptance, but also the establishment of appropriate privacy mechanisms, which make users feel safe and comfortable within such ubiquitous computing environments. Concordantly, in this work we propose an integrated approach for giving the full control over the release and granularity of sensitive context information to the data's owner, comprising a novel ontology-based and privacy-centric context model as well as a mechanism for defining context-dependent access control policies and the corresponding system architecture.

The remainder of this article is structured as follows: In Section II, we will give our problem statement and a definition of sensitive context. Section III reviews related work on privacy in context-aware applications and presents a comprehensive list of requirements for privacy in context-aware applications. In Section IV, we present a general conceptualization of a user's privacy needs, which is the foundation of our privacy-centric modeling approach presented in Section V. In Section VI, our context-aware release triggers for the definition of a user's privacy policies will be introduced. Our framework's system architecture and communication protocol are described in Section VII. After discussing our approach in Section VIII, we conclude.

II. Problem Statement

This work focuses on the modeling and management of a mobile user's rich context information in a privacy-preserving way. We found our understanding of context on the general definition by Abowd et al., declaring context to be "*any information, that can be used to characterize the situation of an entity* [...] *relevant to the interaction between a user and an application*" [6]. Strictly following a user-centric approach, we assume a user's smartphone to be the primary source of information about her current situation. With this work focusing not only on functionality, but on protecting a mobile device user's privacy, we complement the given definition to characterize *sensitive context* as follows: *Sensitive context is any information available through a user's device, that can be used by any entity to infer the situation of the user regardless of any interaction between the user and any application*. Given this definition, we aim at designing a generic and integrated solution for the management and context-dependent access control of a mobile device user's static and dynamic context information. Consequently, we consider it essential to primarily focus on putting the user in full control over the acquisition, release, and resolution of her personal data.

From a privacy-centric point of view, on the one hand, the less data is going to leave the user's mobile device, the better. Yet in order to enable a diversity of context-aware applications, there is usually a need for communicating one's current context information to other parties. For privacy reasons, however, we argue that there must not be any party but the user herself able to access or control her complete context information at any point in time, thereby ruling out any solutions based on a trusted third party approach. On the other hand, providing central reasoning components with some carefully selected context information seems nonetheless desirable in some situations, e.g., in order to allow for the efficient realization of multi-subject context-aware applications. Existing frameworks typically comprise four consecutive stages to enable context-awareness, i.e., context acquisition, context modeling, context exchange, and reasoning [7]. In order to protect a user's context privacy along this chain, an integrated approach should hence comprise the following requirements:

1) An *abstract conceptualization* of a user's privacy needs, that models context on easily comprehensible layers according to different groups of requesters.
2) A *formal model* of a user's context, that already integrates some core aspects of privacy of context information and provides the ability to easily integrate current and future privacy and security methods into context-aware applications.
3) An effective mechanism for controlling the dissemination in form of *user-defined privacy policies*. These policies have to consider the user's as well as the requesting entity's context and should be evaluated in a context-aware manner.
4) An *integrated system architecture*, that is suitable for the above mentioned use cases and all kinds of context requesters, i.e., applications running exclusively on the user's device, peer-to-peer, and third party services while always emitting only the minimum amount of information required.

In the next sections, we propose a solution to all of these requirements: The privacy layers described in Section IV incorporate an abstracted view on a user's privacy needs. Our *Context Representations* model is concerned with the formal representation of a user's context information (Section V). We design a mechanism for context-dependent privacy preferences in Section VI and eventually put the pieces together with our privacy-centric and context-aware framework for the dissemination of context information in Section VII.

III. Related Work

This section presents related work on privacy mechanisms for context-aware applications. Several different categories of approaches can be found in literature. Some of them rely on trusted third party (TTP) solutions for efficient context dissemination, whereas other systems adopt a peer-to-peer (P2P) based approach in order to avoid such central points of attack. In addition, there are rule languages for defining access control based on contextual information as well as different obfuscation techniques for adequately reducing the richness of a user's context information before their release.

A. Context management frameworks and tools

A common architectural model for the realization of location-based and context-aware applications is the use of a TTP acting as some kind of middleware for the aggregation

of its users' context information [8]. It is necessary for all of the system's participants to fully trust this component. The CoPS architecture introduced in [9] implements such a central privacy service. While allowing for different granularities of context items, it does not permit the definition of context-dependent access rules. Another TTP-based approach called CPE [10] enables the definition of context-dependent privacy preferences, but lacks mechanisms for releasing information in different granularities. With a focus on context-dependent security policies, the CoBrA platform [11] deploys the Rei policy language [12] in order to enable the definition of access control rules depending on a user's current context. Beyond that there is much literature on different techniques for the obfuscation of contextual information. As an example, [13] presents another centralized approach focussing on context ownership and offering obfuscation mechanisms for several kinds of context information based on SensorML process chains, obfuscation ontologies, and detailed taxonomies describing dynamic granularity levels. In contrast to these systems, purely P2P-based approaches such as [14] get along without any central component. As a major drawback, such architectures can hardly be efficiently deployed in applications depending on up-to-date context information of a whole group of users at the same time. Behrooz et al. presents CPPL, which is a policy language that can be used to define the access rules for a user's context information in a context-dependent way [15]. Noticeable, the rules that can be set up here are not only context-dependent, but the engine responsible for policy enforcement itself is context-aware, i.e., it maintains an up-to-date set of active policies according to the user's current context in order to minimize the number of policies that have to be considered when processing an incoming request for context information. However, the policy mechanism has been designed to only depend on the context owner's content and its social relationship to the context requester. We argue, however, that a comprehensive policy approach should be able to also take the requester's current context into account as well. Consequently, it is not clear how a requester's context could be transferred to the context owner in a sensible and privacy preserving way, too.

B. Privacy mechanisms for context-aware applications

Based on the above mentioned works [9]–[15] and further literature on privacy in context-aware applications, in this section we identify and complement a set of different techniques and requirements for realizing different aspects of privacy. Naturally, an integrated approach for modeling and managing context should incorporate all of these mechanisms. In the following, we will list and briefly explain the most important of these requirements.

1) Fine-grained control: Above all, a minimum set of **access control** operators such as $grant$ and $deny$ has to be available in order to be able to define different requesters' access rights. Additionally, users should be in a position to define their privacy preferences in a **context-dependent** way. For example, Xie et al. [16] show that a user's willingness to release any of her context information to others depends on a number of different factors, such as time, companion and emotion. Considering the nuances in privacy preferences among different users, Bokhove et al. also argue for the need of **fine-grained** and **expressive** privacy mechanisms [17].

2) Adjustable quality-of-context (QoC): **Variable granularity** and other QoC-related aspects can be used to reduce a piece of context information's precision or accuracy [18]. As an example, consider a user reading her e-mails. Different granularities of her currently modeled activity context might contain `read-e-mail`, `computer-work`, `office-work` and `working`. Moreover, credibility issues, such as **intentional ambiguity** and **(white) lies**, can be used in order to lower the confidence or validity of a piece of context information. Notice that such behavior is common in our everyday actions in the offline world, too, e.g., not answering telephone calls in order to conceal presence or availability. **Adjustable freshness** and **temporal resolution** are other means for intentionally reducing a piece of information's quality, e.g., regarding its age, capturing time or temporal validity. It can hence be used in a similar way as intentional ambiguity to obfuscate a user's context.

3) Consistency and completeness: We define **consistency** of a user's privacy preferences as another important requirement, stating that a context requester must not be able to retrieve ambiguous, e.g., contradicting pieces of context information. Considering that completely denying a request for a piece of a user's context information might itself reveal much, we define **completeness** to be the principle of answering any request with a plausible response. This is important, as it is often more privacy-preserving to give an imprecise yet plausible reply than to refuse a request.

4) Notifications and logging: **Notifications** can optionally be sent to a user upon a request of her context information [9]. Additionally, requests should be **logged**. These techniques allow for users being well informed about usage of their personal information [17] and can be used as a social means able to contain the intentional abuse of contextual information.

5) Symmetry: Another important concept adapted from the offline world is **symmetry**, stating that a certain party has to reveal just as much of its own information as it requests [19]. In connection with context-aware applications, this especially applies to the realization of peer-to-peer-based scenarios.

6) Anonymity and pseudonymity: Other concepts for privacy in context-aware applications are **anonymity**, **pseudonymity**, and **k-anonymity** [20] relating to a user's identity not being known by a system, only being known by a pseudonym, and a user being indistinguishable from k-1 other users, respectively.

In order to seamlessly protect a user's privacy, it also seems beneficial to closely band together the acquisition, modeling and management of a user's context information. More general requirements are extensibility, e.g., a system's ability to automatically integrate new types of context sources and usability.

IV. A Layered Conceptualization of Privacy

We will now introduce a practical conceptualization of privacy as a general mapping from different groups of requesters of context information to distinct privacy layers. These layers resemble what we believe to be a good compromise about a privacy-aware user's sensation of different levels of information accessibility and reduction of complexity. To keep

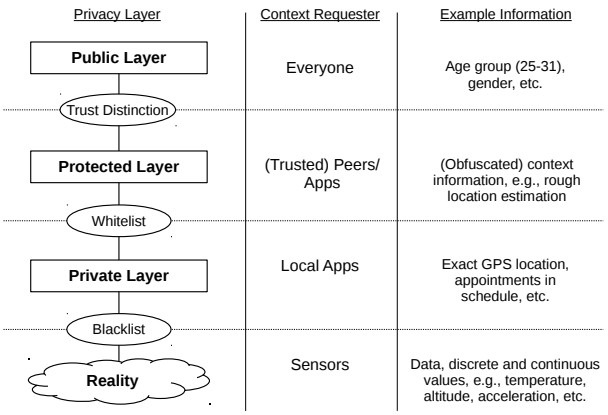

Figure 1. The four different privacy layers defined in ALPACA and their most likely audiences as well as some example context items for each layer.

things small and simple, there are only two roles defined: the *context owner*, who is the user whose context is to be protected and the *context requester*, which might be any application, service or peer requesting access to the user's context.

As shown in Figure 1, we define four logical layers that can be mapped to a user's privacy needs, as well as different gateway mechanisms controlling these layers' permeability. At the bottom layer we put *reality*, possibly containing more information than any types of sensors and reasoning mechanisms will ever be able to capture. Obviously, there is no need to implement anything on this layer. However, it still has to be considered as this layer constitutes what is to be reflected by any context modeling approach. Users might feel uncomfortable knowing that each of their smartphone's sensors is recording data all the time, e.g., in some situations or at certain locations a user might not want her smartphone's microphone to record audio, and hence this sensor should be turned off automatically. Users should thus be able to set up a context-dependent blacklist for defining which sources of context information should be turned off. Every time the user's context changes, this blacklist has to be re-evaluated.

The next layer is the *private layer*, holding all the information a user wants to have available for herself, i.e., context-aware services and applications running exclusively on her mobile device. Consider for example locally run apps, which adapt their appearance and behaviour according to the user's current context. In order for such services to be responsive and proactive, this layer enables access to the most fresh and sophisticated context representations. Naturally, these high-resolution representations are probably not intended for everyone else as well. Thus, the trigger mechanism described in Section VI is used as a context-dependent whitelist for the release of certain representations to the upper layers.

Context information which pass this whitelist enter the *protected layer* and might hence be available for some other entities, too, such as trusted services and peers. For example, a user might be reluctant to share her current whereabouts with everyone, but maybe with some of her friends in her spare time or with her employee during working hours. Naturally, the number and composition of context representations available on this layer will hence change dynamically based on the user's privacy preferences and current context.

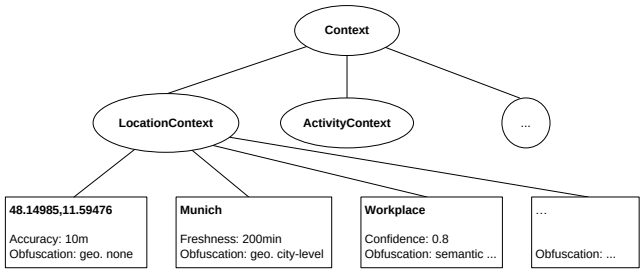

Figure 2. An example context tree with its second-level *Context* nodes linking to an arbitrary number of representations, each indicating its level of obfuscation based on the semantics of the underlying information.

Additionally, a user might be willing to share some kind of information about herself with anyone, meaning that these information are available on the *public layer*. This might, e.g., be true for information that are somehow obvious anyway, such as personal profile data containing the user's gender or age group. However, a user might still define notifications to be displayed when these kinds of information are being requested. That said, notice that our system's user is of course not forced to abide by these layers in the way we just described, but rather can individually choose which level of visibility fits her own situation by the use of appropriate release triggers.

V. Modeling Context Using ALPACA-CoRe

A privacy-oriented approach for modeling a user's context should inherently focus on the enabling of fine-grained access control mechanisms over these dynamic and sensitive data. Applying the privacy techniques listed in Section III to the modeling part of our framework implies that at least the concepts of variable granularity, intentional ambiguity, white lies, freshness, and consistency should directly be integrated into our model. Our basic concept is that a user should be able to maintain different versions of her current context information that can be shared with different context requesters, as the latter might not appear equally trustworthy to the context owner. Accordingly, *Context Representations* (CoRe) model is designed to inherently store several heterogenous versions of a user's context information in parallel, each serving a different purpose for different groups of context requesters. Notice, that there have been some considerable modifications to the original version presented in [1].

A. Representations of context information

Approaches for context modeling (cf. [21] for a comprehensive survey) typically adopt a hierarchic understanding of context, which can be illustrated using a tree-based structure. In such a hierarchic scheme, the root node aggregates all categories of a user's context information. At the second level, distinctions between the basic types of context information are made, such as a user's current time, location, or activity. Each of the tree's second-level nodes may be a parent to a hierarchy of an arbitrary number of nodes representing the corresponding category in a different way. To preserve a user's privacy, the simultaneously possible representations of a user's current location might differ, e.g., in their spatial resolution, accuracy and freshness, or in case of a white lie maybe even validity. Apart from the creation of such hierarchies being a

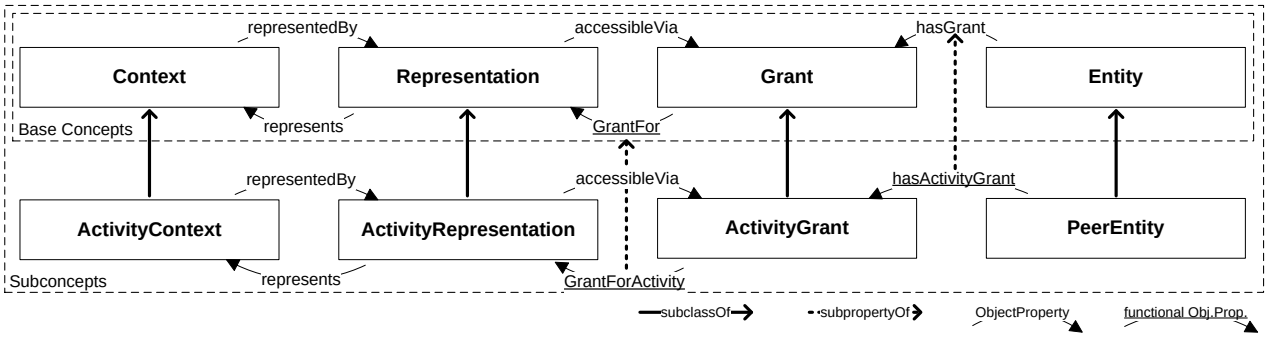

Figure 3. The cornerstone concepts and properties of the CoRe model: *Context*, *Representation*, *Grant*, and *Entity* as well as corresponding example subclasses.

non-trivial task [13], it is thus not possible to stick to such taxonomies in general, due to contemporaneous representations of the same kind of context differing in more than one dimension. In contrast to existing context models such as MUSIC [22], which use the term "representation" in order to label the data formats used for communication (such as XML or JSON), we hence suppose distinct representations of context information to possibly differ from each other on a semantic level, independent from any encoding. As an example, consider the three different representations of the user's current location in Figure 2: The instance on the left holds the current GPS position fix of the user. In contrast, the one in the middle only states the user's location on a city level, while the third uses a non-geographic, symbolic location identifier that cannot be mapped to a geographic one – at least not without any further knowledge about the user. The acquisition of these representations is not within the scope of this work. However, we require this process to be performed by sensing, reasoning, and context obfuscation services running on the user's device. The different representations stored in the model can be marked to be accessible by different groups of requesters. Which representation is to be released to whom might depend on the privacy level assigned to the requester and on context information itself: Services running exclusively on the user's mobile device are likely to be allowed to access the user's context information with the highest resolution available, whereas peers and third party services might only be able to retrieve some kind of adequately obfuscated representations.

B. Components of the CoRe model

Following an OWL-based modeling approach, our context model consists of the base classes shown in Figure 3, i.e., *Context*, *Representation*, *Grant* and *Entity*. Each of the first three has subclasses for the different categories of context, such as *ActivityContext*, *ActivityRepresentation* and *ActivityGrant*, respectively. As stated above and indicated by the *representedBy* property, a certain subclass of context may be described by multiple contemporaneous instances of the according subclass of *Representation*. The model's basic structure was inspired by the ASC model by Strang et al. [23] enabling service interoperability based on a shared understanding of and transformation rules for different, yet logically equivalent scales. There, the interrelation of different scales such as a mile and kilometer scale are described along with transformation rules that can be used to translate from one scheme to another. Quite the contrary, our CoRe approach aims at modeling different representations of the same kind of context information, which according to the privacy mechanisms listed in Section III do not necessarily have to share a similar or at least consistent meaning at all. In fact, there must not exist any transformation rules which allow for a trivial conversion, e.g., from a low-resolution representation to a high-resolution one.

As an enabler for the definition of privacy preferences based on our model, an instance of *Representation* can be made accessible to a group of context requesters via subclass instances of *Grant*. This can be modeled by using the corresponding subproperty of the functional *GrantFor* property as depicted in Figure 3. A context requester can be any *Entity* requesting some of the user's context information and can be referred to individually, by group membership, or by dynamic constraints on context. In order to allow for privacy policies that also depend on the requester's context, such as "friends at my location", the CoRe model also stores the known contexts of all active requesters. As claimed by the requirement for consistency, however, a requesting entity must never be granted access to more than one representation of the same type of context information at the same time, as this might result in an ambiguous response. In order to integrate this understanding of consistency into the CoRe model all subproperties of the *hasGrant* property are marked as *functional*, too. Constricting the modeled relations by the ontology's *functional property* constraint as described allows for dynamic consistency checking at runtime based on the built-in OWL reasoning capabilities. Hence, a user's privacy policies do not have to be tested statically for any inconsistencies, which instead can automatically be detected by an inconsistent state of the ontology. Likewise, we provide the *GrantFor* property with a *minCardinality* constraint of 1, stating that there must also be at least one *Representation* for each *Grant*. This can be used to realize some kind of garbage collection that is able to automatically detect and remove an obsolete *Grant*. Notice, that the inverses of these properties are not constrained like that, e.g., one instance of *Representation* can be accessible via several *Grant* instances simultaneously and a *Representation* can exist even if there is no corresponding *Grant*.

In order to be able to automatically assess the resolution, validity, and sensitivity of a *Representation* instance, each of them provides information about its *ObfuscationLevel*. As shown in Figure 4, each type of context can have an arbitrary number of obfuscation levels and each *Representation* belongs to exactly one of these. Considering the great differences in semantics that different types of context information and obfuscation techniques might display, a generic labeling ap-

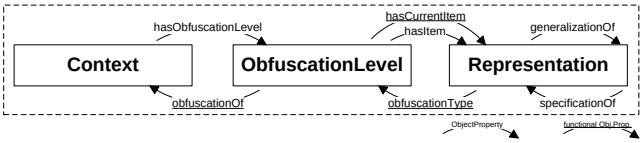

Figure 4. Each instance of *Representation* describes a certain type of *Context* using a corresponding scheme, i.e., a certain instance of *ObfuscationLevel*.

proach for these levels seems unfeasible. Hence, each subclass of *Context* is expected to define its own obfuscation scales, probably provided by the service used for creating the corresponding representations. Within a single scale there might be a naturally defined hierarchy, such as the granularity of location. Instances of the same subclass of *Representation* can hence be marked to be a *generalizationOf* one another, which is a *transitive* property. This can be used to model that a context requester who is granted access to a high-resolution representation can also access low-resolution ones, e.g., if she only asks for the latter. This does not violate the consistency requirement, as conflicting responses caused by contradicting representations cannot arise. In addition, each *ObfuscationLevel* has a functional *hasCurrentItem* property indicating which of the corresponding *Representation* instances is currently valid and should thus be used for the evaluation of the user's privacy policies. This is necessary as for each *ObfuscationLevel* there might be a multitude of *Representation* instances stored in the model in parallel, some of which are not valid any more, but still remain in the CoRe model, e.g., to enable adjusting the freshness of released information.

In this section, we have described a novel, privacy-centric approach for modeling a user's context information, which can be used to manage multiple representations of a user's context information intended for different context requesters. In the next section, we will focus on how context-dependent privacy policies can be realized based on our model.

VI. Context-Dependent Privacy Policies

In order to protect a user's context information from unintended leakage, our system takes a highly restrictive, whitelist-based approach: Apart from fully trusted apps running exclusively on the user's device, explicitly stating that some information should be accessible by a certain requester is the only possibility for a user to share any piece of context information. In this section, we will introduce a suitable mechanism for the realization of corresponding privacy policies, show example usage, and demonstrate our framework's capability to preserve consistency by automatically detecting and resolving conflicting policy definitions at runtime.

A. Trigger-based release of context information

The ALPACA framework comprises a new mechanism for the definition of flexible, fine-grained, and context-based access control rules on behalf of the context owner. In order to facilitate the definition of both static and context-dependent privacy policies, the context owner must be able to manage her contacts in groups comparable to current online social networking (OSN) services. To this end, *Release Triggers* (RTs) can be set up as privacy policies for controlling the release of a user's context information. Behrooz et al. differentiate between situational context, that can be used to evaluate privacy policies and sensitive context, that is to be protected by the policies [15]. In contrast, we argue that all of a user's context information is likely to be both relevant in the policy definitions and worthy of protection. The RT concept hence enables the deployment of *context-dependent* privacy preferences based on and intended for the rich context information stored in the CoRe model. It can be used for dynamically whitelisting the release of a certain *Representation* of context information to a context requester.

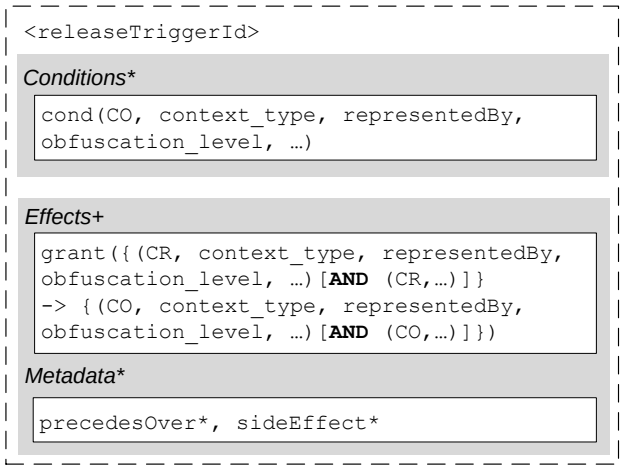

Figure 5. The logical structure of the release trigger mechanism: *Conditions* and *Effects* allow for the definition of context-dependent privacy preferences.

The logical structure of a trigger and its subcomponents are outlined in Figure 5. For identification purposes, every trigger is assigned a unique name. It further consists of sets of *Conditions* and *Effects* as well as a *Metadata* section. The *Condition* part holds all prerequisites posed on the context owner's (CO) current context for the trigger to become active. Here, several conditions can be specified per trigger, which are evaluated as a whole using the logical `AND` operator. The required conditions may base on *Representations* of any context type and *ObfuscationLevel* that are available in CoRe. This allows the definition of conditions based on any desired granularity as well as any combinations of context types. The *Effects* part states what kind of *Representation* is to be released upon a certain entity (CR) requesting context information. In order to differentiate between context requesters in the *Effects* part, entities may either be referred to explicitly or implicitly via group membership, context properties, or any combination thereof. The latter allows for referring to the requester's context in a trigger definition, too, so that the final release of context information can depend on the requesting entity's context. In the *Metadata* part, e.g., information about the relative importance of a trigger can be defined, such as precedence rules used for conflict resolution.

If a trigger fires, the corresponding instance of *Representation* will be made accessible to a context requester. In terms of the concepts of the CoRe model, this is achieved by the creation of a new *Grant* instance. The latter will be linked to the *Entity* characterized in the release trigger's *Effects* part as well as with the described *Representation*. Which instance of *Representation* is to be released to the requester

can be defined by the context owner in terms of the desired obfuscation level, maximum or minimum accuracy, confidence, and freshness properties. Again, several effects can be defined within the scope of a single trigger. However, it is important that each release of a *Representation* by any *Entity* is managed by a dedicated *Grant* instance, as the access to individual representations may depend on different conditions on the requester's context. From [9] we have adapted the idea that accessing a piece of context information by a certain requester might have side effects such as notifying the user, as we agree on that being a proper means suitable for keeping the user informed and containing data abuse. If desired, a user can hence also specify which *SideEffect* should be activated when a certain kind of *Representation* is accessed using a certain *Grant*. The metadata section can be used to define the relative importance of a trigger compared to others, which is needed for resolving conflicting policy definitions. Both the side effects and precedence rules will also be added to the CoRe model.

B. Context-aware policy evaluation

Apart from allowing for context-dependent privacy policies, our approach can be considered *context-aware*, too, as the policy evaluation takes into account the user's current situation in order to reduce the number of rules that have to be evaluated upon an incoming request for context information. To this end, the evaluation of a user's release triggers is performed in two stages, thereby fulfilling the requirement for context-awareness of privacy policies stated in [15]. In the first stage, a release trigger is set active if all of its *Conditions* are met. As already explained, these conditions only depend on the context owner's locally known context and can hence be evaluated proactively to any context request. The up-to-date subset of active triggers is managed directly in the CoRe model and is adjusted accordingly with each change of the user's context. As a result, only active triggers have to be evaluated upon an incoming request for context information, i.e., during the second stage. This procedure can be naively implemented as follows: If the local context changes, all *Grant* instances are removed from the model. In the next step, all triggers are removed from the context owner's active trigger subset. Notice, that in this state no one but *private layer* entities has access to the context information. Now the first, context-aware stage of the trigger evaluation starts: Based on the currently modeled context, all triggers' *Conditions* will be matched in order to decide which triggers enter the active set. This set then holds all triggers that can potentially fire upon an incoming request for the user's context information. The second stage is a lazy rule evaluation performed in reaction to an actual request. Consequently, this evaluation only has to be performed on the filtered subset of active triggers instead of the user's complete policy set, which has the potential to improve the response time of the system. The active triggers will now be evaluated and in the case a trigger fires, a corresponding *Grant* instance as well as any *SideEffects* are added to the model.

To enable context-awareness in an actual implementation of the trigger mechanism in a rule language, the triggers' logical structure must be split into two parts: The *Conditions* part on the one hand and the *Effects* and *Metadata* part on the other are thus to be defined in two seperate rules, as illustrated by the example trigger definition in Figures 6, 7, and 8.

C. Example of a release trigger

We will now demonstrate the interplay of the CoRe model and our release trigger mechanism by means of a simple example scenario: Alice, who is the context owner, has set up a pair of release triggers defining which information she is willing to share with her colleagues. Her co-worker Bob is the context requester, who currently is at work. The full description of the first trigger can be seen in Figure 6 as pseudo-code. This

```
Trigger1 {
  Condition:
    Entity:          Alice
    Context:         TimeContext
    representedBy:   worktime
  Condition:
    Entity:          Alice
    Context:         LocationContext
    representedBy:   atWork
  Effect:
    Context:         LocationContext
    Obfuscation:     building-room
    ---
    Entity:          coworker
    Context:         LocationContext
    representedBy:   atWork
  Metadata:
    precedesOver:    Trigger2
}
```

Figure 6. Schematic definition of an example release trigger, *Trigger1*, depending on the context owner's current time and location given in pseudo code.

trigger defines that given it is worktime and Alice's current location is found to be her workplace, co-workers who are at work themselves are allowed to see in which room she currently resides in. The actual implementation of the trigger happens, as already mentioned, by outsourcing the condition part into an own rule. A second rule describes the effects and metadata part of the trigger. The corresponding sub-rules are shown in Figures 7 and 8, respectively. The syntax used here takes a generic format assuming rule evaluation in forward-chaining mode. The `makeTemp` creates a blank node in the model. For the sake of brevity, the definition of any SideEffects and usage of exact RDF syntax have been left out here.

The conclusion part of the *Conditions* rule

```
[shareRoomWithColleagues:
  (Alice hasContext ?t)
  (?t type TimeContext)
  (?x obfuscationOf ?t)
  (?x hasCurrentItem worktime)
  (Alice hasContext ?l)
  (?l type LocationContext)
  (?y obfuscationOf ?l)
  (?y hasCurrentItem atWork)
  -> (Alice hasActiveTrigger Trigger1)]
```

Figure 7. The context-aware *Conditions* part of *Trigger1* as a separate rule.

shareRoomWithColleagues shown in Figure 7 simply states that Trigger1 is to be added to Alice's set of active triggers. For this to happen, all conditions defined in the rule's body part must match the current state of the model. Just as required, the conditions of this rule exclusively depend on Alice's context information. So even though the overall result of the release trigger depends on the context of a requesting co-worker, too, shareRoomWithColleagues can be evaluated pro-actively to any incoming request. As soon as it is working hours and Alice arrives at work, Trigger1 will hence be set active. In contrast, as depicted

```
[Trigger1:
  (Alice hasContext ?l)
  (building-room obfuscationOf ?l)
  (building-room hasCurrentItem ?lr)
  (?lr type LocationRepresentation)
  (?cw isA coworker)
  (?cw hasContext ?cwl)
  (?cwl type LocationContext)
  (?cwl representedBy atWork)
  -> makeTemp(?g)
     (?g createdBy Trigger1)
     (?g GrantForLocation ?lr)
     (?cw hasLocationGrant ?g)]
```

Figure 8. *Effects* and *Metadata* parts of *Trigger1* as a separate rule.

in Figure 8, the Trigger1 rule itself takes into account the current context of the context requester. However, it only has to be evaluated when a context requester sent a request for her context information with his own context piggybacked. The first four triplets define that this rule targets the currently valid representation of Alice's location context on the *building-room* obfuscation level. The remaining lines of the rule's body state that the matching representation shall be made accessible for any co-worker who is at work: A new instance of *LocationGrant* is added to the model and linked to the chosen representation and matching entities. If there were any *SideEffects* that should be performed upon this *Grant* being used, it would also be stated here and the corresponding individuals are added to the model.

The release trigger shown in Figure 9 only relates to Alice's time context and does not pose any requirements to her whereabouts. In contrast to the first one, hence, this rule also matches when she is on a business trip or on a day off or simply late for work. Consequently, Alice decided to only let her colleagues know about her location on a city-level in this case. Assume that Bob requests her current location during the working hours: In case Bob is not at work himself, none of the two triggers will release her location information to him. If Bob is at work and Alice is not, only the second trigger matches the current situation and a *Grant* instance to her city-level location will be linked to Bob. When she is at work, however, both triggers match the situation. Accordingly, both rules fire and Bob is granted access to two different instances of *LocationRepresentation*, which clearly violates the requirement for consistency. How the CoRe model is capable of automatically detecting and resolving such conflicting policy definitions will be explained in the next section.

```
Trigger2 {
  Condition:
    Entity:          Alice
    Context:         TimeContext
    representedBy:   worktime
  Effect:
    Context:         LocationContext
    Obfuscation:     city
    ---
    Entity:          coworker
    Context:         LocationContext
    representedBy:   atWork
}
```

Figure 9. Schematic definition of an example release trigger, *Trigger2*, depending on the context owner's current time only.

D. Dealing with inconsistent privacy preferences

When making usage of context-dependent access control policies, it might happen that two or more (possibly contradicting) rules match a given situation. We define a set of privacy policies to be conflicting, if they produce an ambiguous set of access grants for a context requester. A set of *Grant* instances is ambiguous, if any *Entity* is granted access to more than one *Representation* instance of the same subclass of *Context* at the same time. This situation is prone to harm a user's integrity, which becomes evident when considering two contradicting representations, such as a user's true and fake location information: If a requester is allowed to access several representations of the same subtype of *Context* at once, the result is not only ambiguous, but also likely to negatively influence the context owner's respectability due to being caught lying. Considering the structure of the release trigger mechanism, such situations might occur when a context requester matches the entity description in more than one active trigger's *Effects*. In order to automatically detect such situations at runtime, we use the underlying ontology's built-in reasoning capabilites to dynamically check the CoRe model's state for consistency. Constraining all subproperties of the *hasGrant* and *GrantFor* properties to be *functional* and setting the *minCardinality* constraint of the *GrantFor* property to 1 as described in Section V-B allows for the following assertions:

1) Any instance of *Entity* is allowed to be linked to at most one instance of a given subclass of *Grant*.
2) Any *Grant* can only be linked to exactly one instance of the corresponding subclass of *Representation*.

If the rule evaluation produces a state that conflicts with these requirements, checking the model for consistency will result in an error. This check is performed each time the rule evaluation finishes. In order to solve conflicting policy definitions, the *precedesOver* property can be set for a release trigger. In an actual implementation, the first time an inconsistent state is detected the context owner can be notified about the conflict and the triggers that caused it. The user then can decide which of the triggers involved should precede over the other. The decision will be stored in a *precedesOver* property of the winning trigger. The next time the same inconsistent state arises, inferior *Grant* instances can be automatically removed from the model, as every *Grant* instance links to the trigger it

has been created by.

In this section, we introduced the concept of *Release Triggers* for the context-based definition of a user's privacy preferences. The trigger mechanism has been designed to consider both the context owner's and optionally also the context requester's context to decide which kind of representation is to be released to the requester. The evaluation of these access control rules is performed in a context-aware way, which takes into account a user's context in order to minimize the number of rules that have to be evaluated upon incoming requests for context information. Eventually, we explained how conflicting policy definitions can dynamically be detected and resolved. In the next section, we will outline our framework's core components and overall system architecture.

VII. ALPACA System Architecture

In the previous sections, we introduced the privacy-centric CoRe approach for modeling a user's rich context information. Additionally, the RT mechanism enabled a context-dependent definition of a user's privacy policies. These aspects will now be joined together. For this purpose the ALPACA system is presented in the following together with a description of the communication occuring between the different components and context requesters. Finally, we sketch how our framework can be integrated into a modern mobile operating system such as Android.

A. Managing acquisition and release of context information

Our framework's key component is the *Privacy Manager* (PM), which is responsible for managing all access to the user's context information by enforcing the privacy policies set up by the context owner. To this end, the user is both able to control what information enters the CoRe model by blacklisting the acquisition of certain types of context information as well as to decide which kind of information is released to whom by whitelisting access to context information for trusted requesters (cf. Section IV). The overall system architecture of ALPACA and the logical placement of the PM is depicted in Figure 10. Serving as a gatekeeper, the PM is the only component able to directly access and update the information stored in the context model.

For the acquisition of a user's up-to-date context information, we assume a number of different hardware and software sensors for low-level context recognition as well as services capable of high-level context reasoning to be available on a mobile device. In addition, we expect obfuscation techniques that can be used for adjusting the granularity or validity of a certain type of context information to be implemented as well. The corresponding sensors and services are responsible for capturing a user's context, transforming it into *Representation* instances of the corresponding subclass of *Context* and for providing the required meta data, such as *Freshness*, *Accuracy*, and *ObfuscationLevel*. Each time one of these context sources produces a new representation, it is reported to the PM. After authenticating the source, the PM will update the CoRe model by adding the *Representation* and setting the *hasCurrentItem* property of the respective obfuscation scale to link to the new *Representation*. As these operations alter the state of the context model, the PM now re-evaluates the context-dependent acquisition blacklist, turns corresponding sensors

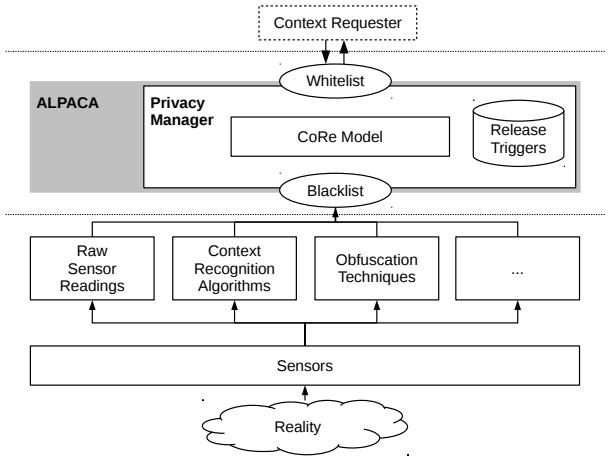

Figure 10. The *Privacy Manager* is the core component of the ALPACA framework and acts as an exclusive interface to the modeled context information for both context sources and context requesters.

and reasoning services on or off, respectively, and re-evaluates the *Conditions* of the context owner's release triggers' in order to update the active set.

Moreover, the PM also controls the release of a user's context information to a context requester. As described in Section VI, this is realized on a per-request basis using lazy rule evaluation due to accessibility and granularity of a user's context information possibly depending on the requester's current context, too. A context requester can utilize most of the representations' meta data stored in the CoRe model to specify the characteristics of the requested *Representation*, such as context type, freshness, and accuracy. The PM will consider these requirements and look up the best matching accessible items. In the following, the basic communication flow within context requests issued by different context requesters will be explained.

B. Communication flow and request handling

The ALPACA framework aims at providing a generic solution for the management of a user's privacy preferences and the dissemination of context information fit for all kinds of usage scenarios. One can identify three different categories of context requesters, that can be mapped to the privacy layers introduced in Section IV as follows:

- *Local applications* running exclusively on the user's device and not sending any data off the device can be allowed to access a user's *private layer* context.
- *Third-party services* communicating context information to remote entities are to be placed on the *protected layer* or *public layer*. Distinct services might yet be granted access to different representations depending on the user's trust in the service provider.
- *Peer users* requesting the user's context information can be placed on the *protected layer*, too. Which representation they are allowed to see might depend on both the requesting peer's identity and context.

In order to enforce the privacy policies set up by the context owner, in each case the PM first verifies which entity is requesting context information and which of the above categories

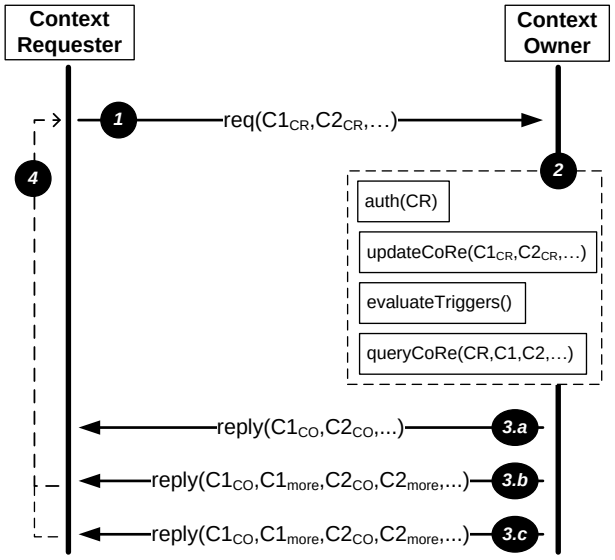

Figure 11. The ALPACA framework's turn-based communication protocol deployed for the exchange of context information between peer users.

the requester belongs to. In case the requesting entity is found to be a fully trusted *local application*, no further checks have to be performed: The PM will look up the *Representation* instances in the CoRe model matching the criteria specified in the request and return them to the context requester. Apart from context-aware applications merely consuming context information, also services performing context reasoning in order to infer high-level context fall in this category. The latter can hence be regarded both as requesters and sources of context information.

If a context requester is a *third-party service*, the PM will query the model to learn if there are any *Grant* instances related to the requester. Hence, if the context owner has set up release triggers that grant the requesting entity access to one or more *Representation* instances, the PM again looks up the most appropriate ones and returns them to the requester. However, if a context requester is not explicitly granted access to any *Representation*, a mediocre representation defined by the user will be returned: To protect a user's privacy, this *Representation* should be highly obfuscated, i.e., of considerably reduced granularity, outright cheated, or simply "*N/A*". Both personal context-aware applications that send data over the Internet as well as multi-subject context aware applications belong to this category. In order to efficiently cater for different needs of context-aware applications, the PM supports request/response-based and publish/subscribe-based communication patterns. Both local and remote requests for a user's context information may hence contain a flag indicating whether the requester is asking for a single reply or continuous updates. In the latter case, a desired update interval as well as a context-dependent break condition can be set by the context requester. As long as there is a corresponding *Grant* instance and the break condition is not yet met, the PM will supply the requester with the requested information.

Due to a user's privacy policies possibly depending on the requesting entity's context as well, the context owner might require her *peer users* to communicate their own context when requesting her context information. Figure 11 outlines the communication flow during a request for context information issued by a peer user. As a means for protecting the requesting entity's privacy, too, we introduce a turn-based protocol for the exchange of context information. Our protocol, which also aims at fulfilling the requirement for symmetry in P2P-based usage scenarios, works as follows: *(1)* The context requester (*CR*) issues a request for some of the context owner's (*CO*) context information. The *CR* implicitly specifies which types and granularities of context information he is interested in by piggybacking equivalent information about himself on the request. *(2)* The *CO*'s PM analyzes the request by authenticating the *CR* and adding the contained context information about the *CR* to the CoRe model. At this step, naturally, the *CO*'s release triggers are re-evaluated based on the updated state of the model. The PM then looks up any *Grant* instances related to the requesting entity based on the model's current state. Additionally, the PM also queries the CoRe model for any active triggers that might match the *CR*, e.g., based on the *CR*'s identity or static group membership. Based on this information, for each context subtype the PM learns if the *CO* is willing to release any *Representation* instances to this requester. The next steps are hence to be performed for each type of context information separately: Depending on whether a corresponding *Grant* instance has been found or not, the PM either replies with the respective *Representation* or with the subtype's highly obfuscated standard representation. *(3.a)* If both no *Grant* instances and no active triggers have been found, the protocol ends here. *(3.b)* Otherwise, if the *CR* is explicitly granted access to a *Representation*, the PM looks up the corresponding instance matching the granularity specified in the request. In case the requester asked for a lower granularity representation than the one that is granted to him by the respective *Grant*, the PM automatically adjusts the granularity by selecting the matching *Representation* using the CoRe model's transitive *generalizationOf* property. Given that the *CO*'s maximum granularity for this situation is not yet reached the PM will notify the *CR* about this. *(3.c)* Additionally, there might be active triggers related to the *CR* that have not fired yet, as they depend on the requester's higher resolution context. If so, the PM also informs the *CR* about the availability of representations of higher granularities that possibly will be released after learning more about the *CR*'s respective context information. In order to reveal as little as possible, however, the *CO*'s PM only indicates the existence of higher resolution information, but does not betray the maximum granularity available. *(4)* Upon reception of the *CO*'s response, the *CR* updates the local CoRe model and decides about entering another round of the protocol in order to learn more details about the *CO*'s context. Notice, that based on the received *CO*'s context the *CR*'s set of active triggers might have beeen updated, too. If so, the *CR* selects a corresponding higher granularity *Representation* in order to issue a new request for the *CO*'s context information and the protocol is repeated until either the *CO* or *CR* reaches a maximum level of granularity and decides to quit.

We hence enforce the symmetry requirement by demanding that the release and resolution of a user's context information directly correlates with the information the requester has to share. Notice, however, that there is no guarantee that each party will always learn just as much about the other one as it

conveyed about itself. At each step in the protocol an entity can decide to quit the protocol and not share any further information, possibly depending on the other party's context information just learned. However, the turn-based nature of the protocol tries to alleviate this problem. The communication protocol can be applied to the example from Section VI-C as follows: According to the release triggers that Alice has set up, she only shares her location information with her colleagues during her working hours. If her own location context is known to be *atWork* and so is Bob's, however, she has decided to even let him know in which room she is located. Her mobile device of course knows what time it is and where she is at, yet the current location of Bob must be known as well. Assume that Bob already knows she is at work, so he directly queries her location on a room level. Following the symmetry requirement, he therefore has to tell her the same information about himself. When the request arrives at Alice's PM, Bob's location context can be reasoned to be *atWork*, hence her room-level location information can be released to him. However, in case Bob is not sure about here being at work, he probably would just formulate the request with his own location being reported as *atWork*, which results in Alice's PM stating the same and informing Bob about higher resolution context being available. Notice, that in step *(2)* it is also possible that the requester's context can be used to calculate new local context at the context owner's device, e.g., learning about a peer user's location might lead to a context source being able to detect proximity to that user, which might result in new triggers being added to the context owner's active set.

C. Entity Authentication

Different sources and requesters of context information vary in terms of placement in the operating system and network location, trustworthiness, liability to impersonation attacks, and frequency of requests to the PM. Consequently, adequate means of authentication have to be considered.

Both third-party services and peer users requesting a user's context information from the PM can be authenticated based on public key certificates. Hence, the authenticity of networked requests from these kind of context requesters can be checked by means of digital signatures. In order to protect the user's context information from eavesdropping, the requester's public key can also be used for content encryption. Obviously, the PM only accepts requests for the user's *protected layer* context from entities that are already known. To this end, the PM holds a repository for all those entities' certificates that the context owner is possibly interested in sharing her protected context information with. For identification and authentication of third-party services and multi-subject context-aware applications one can simply rely on a standard public key infrastructure (PKI) and the validity of a given application's certificate, respectively. The exchange of certificates between peer users can be realized by deploying a protocol similar to the process of becoming friends in our privacy-preserving OSN *Vegas* [24]: Users that know each other in real life can perform an out-of-band key exchange using self-signed certificates in order to bypass the need for a PKI. Common difficulties related to certificate revocation can be avoided by these self-signed certificates being valid for a limited time span only and key renewal on a regular basis. Consequently, in order to reliably check the origin and authenticity of an incoming request for context information the PM simply tries to verify the request's digital signature using the information stored in its local certificate repository. Once authenticated, the PM maps the requester to the corresponding entity and queries the CoRe model for related *Grant* instances.

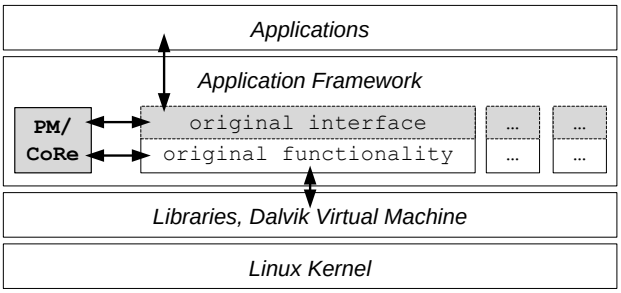

Figure 12. Placement of the *Privacy Manager* in the Android software stack.

Due to its computational overhead, however, the deployment of public key cryptography is overkill for sources and requesters of context information running locally on the user's mobile device and frequently interacting with the PM, e.g., services performing context reasoning. Instead, an HMAC-based approach could be used for lightweight integrity checking and entity authentication. In order to prevent low-level data leakage the problem of securing communication channels between different components on the mobile device itself will have to be further investigated. Initial thoughts on how the PM could blend into a mobile platform's software stack will be presented in the next section.

D. Integration into Android

We will now outline how our framework can be integrated into a modern smartphone software stack such as Android. As shown in Figure 12, the latter is based on a Linux kernel and uses the *Dalvik Virtual Machine* to execute applications written in Java and converted from bytecode to the more lightweight `.dex` (*Dalvik Executable*) format. For security reasons, each application runs in a dedicated sandbox process using its own instance of Dalvik. Apart from standard libraries in C and Java, Android ships with an *Application Framework* that can be used by developers to access a mobile device's base functionality, such as telephony and window management, and also to request context information, e.g., the current GPS position fix. With the PM working as a gatekeeper responsible for handling all requests for updating and querying the user's context information, this layer of the Android software stack is where our framework's key component is to be placed.

In order to effectively protect a user's context information, applications must no longer be able to access the original functionality of the *Application Framework* directly. Applications must yet still be able to make usage of the well-established programming interfaces provided by the Android SDK in order to request desired information. Consequently, even the existence of the PM should be transparent to applications. This can be achieved by exchanging the standard implementation of all the respective interfaces for proxy code, which redirects corresponding requests to the PM along with information about the context requester's identity. Based on the current context and the privacy layer the requester belongs to, the PM then

can decide which kind of information is to be released to the requesting entity. Apart from this kind of legacy functionality, however, applications and services might be especially designed to work with the ALPACA framework, e.g., peer-based context-aware applications, which directly register with the PM in order to increase performance by calling optimized interfaces. For context acquisition the PM can simply run the original interfaces' code for infering the user's current context, feeding this information into CoRe, and updating the set of active release triggers. Obviously, also different kinds of covert channels that can be used by applications to collect information without explicitly requesting it will have to be identified and eliminated. In addition, the PM must also be able to monitor whether applications behave as expected, i.e., act in accordance to the corresponding privacy layer. For example, context requesters must not be able to trick a user into believing that an application is to be placed on the *private layer* while still sending data off the device. Hence, the PM must not only be in a position enabling it to manage access to context information, but must also be able to control what happens to this information afterwards. To this end, we are currently investigating the applicability of approaches for information-flow tracking, such as *TaintDroid* [25], and plan to extend ALPACA with the necessary functionality.

VIII. Discussion

Strictly following a privacy-centric point of view, all decisions in the design process have been directed at offering users a maximum level of control over the release of context information. Hence, we will now check our framework against the requirements stated in Section II and dicuss the pros and cons of our privacy-centric approach for modeling and managing a user's context information.

In a first step, this article introduced an abstract conceptualization of privacy needs regarding context information. To this end, four different privacy layers and appropriate inter-layer gatekeeping mechanisms have been identified, which render a user's privacy preferences to be effectively manageable by a software system. These layers provide a good compromise about information accessibility, flexibility, and reduction of complexity: A blacklist-based approach can be used to control what kind of context information should be acquired by the user's device. For the release of context information, however, a whitelist-based approach is pursued, argueing that a privacy-aware user wants to feel rather safe than sorry. Consequently, each access to context information has to be explicitly granted by the context owner. The next goal was to find a formal representation of a user's context information that inherently features general aspects of privacy and is easily extensible with regard to the integration of novel privacy and security methods. Based on an abstraction of context in terms of semantically different representations of the same type of information, we designed the ontology-based CoRe model. The latter is able to meaningfully store multiple versions of a user's context in parallel, which can be used to serve different responses to distinct groups of variably trustworthy context requesters. In addition, we presented an effective mechanism for controlling the dissemination of context information in form of user-defined privacy policies. For this purpose, the ALPACA framework utilizes a two-stage rule-based approach for the definition of fine-grained and context-dependent release triggers on behalf of the context owner. The triggers' conditions, effects, and affected context requesters can be specified based on the rich context information modeled in CoRe, which allows for great expressivity and flexibility. Consistency of a user's privacy policies can be detected at runtime by means of ontology-based reasoning on our model's current state. However, using ontology-based reasoning on a per-request basis might become a performance issue once the information modeled in CoRe becomes extensive, which has to be investigated in our future work. Finally, we introduced the *Privacy Manager* as our framework's key component and described the interplay of the different components, which have been integrated into a system architecture suitable for all kinds of context requesters, e.g., ranging from local applications to peer-to-peer and third-party services. By omitting a TTP, there is no single point of trust or failure. Instead, each user keeps the complete control over the release of her context information. Multi-subject context-aware applications can still be realized, yet in each situation each application is only able to learn the amount of information the context owner is willing to release.

In order to offer maximum levels of universality and extensibility, we intentionally decoupled our framework from the acquisition of context information. It is hence independent from existing sensors, reasoning mechanisms, and context obfuscation techniques implemented on the user's device. The latter can be used to realize the concepts of variable granularity, plausible deniability, etc. as required. New sensors and inference algorithms can simply register themselves at the PM without requiring the need for modifications to any of the ALPACA components. In a practical implementation, the PM can simply inform the user about new types or sources of context information being available.

Due to its restrictive nature, however, our approach inevitably comes at the expense of usability and out-of-the-box functionality: All context information the context owner wants to be accessible by other entities in certain situations have to be explicitly released by manually setting up appropriate release triggers. It is thus not clear whether a majority of users is willing to adopt such a whitelist-based system, e.g., facing peer pressure and their own reluctance towards manually configuring the release of context information. The deployment of obtrusive notification mechanisms is also likely to be perceived as being disturbing, such as alerting the context owner about the release of context information on a per-request basis. However, we argue that there is a clear necessity to give users full control over the release and granularities of their context information even if usability is constrained. Moreover, the general privacy awareness of mobile users has to be trained in order to prevent unnoticed large scale data leakage. Nevertheless, user-friendly mechanisms will have to be developed in order to increase the likeliness of a broad acceptance among a wide user base. For example, a layered policy approach as proposed in [17] could be applied, which allows both for simple privacy settings suitable for the mostly unconcerned and pragmatic users and fine-grained policy-definitions for fulfilling the needs of the privacy-aware. Furthermore, it has been shown in [16] that a personalized recommendation of most likely privacy policies based on a user's current context, previously learned preferences and group correlation is feasible, too, which presents another interesting direction for future research.

IX. Conclusion

This article presented ALPACA, an integrated framework for modeling and managing a user's context-information in a privacy-centric way. To this end, we introduced a practical conceptualization of privacy in context-aware applications with regard to who is requesting information from the user's mobile device. On the basis of this abstracted view we presented our ontology-based CoRe model, which can be used for maintaining multiple, semantically different representations of the same class of context information fit for differently trustworthy groups of context requesters. In order to enable the fine-grained definition of a user's privacy preferences a context-dependent, whitelist-based trigger mechanism has been created. Eventually, we described our framework's key components as well as its system architecture and a turn-based communication protocol used for the exchange of context information between different entities.

We are currently working on a prototype implementation of our system allowing us to conduct a user study for evaluating the usability of our whitelist-based release mechanism. As for our future work, we aim at finding mechanisms capable of ensuring consistency over several consecutive context requests by a single entity. Furthermore, we are looking for a way to implement incentive-based mechanisms for the optional release of additional data to third-party services in a privacy-preserving way. Future work will also be directed at finding new obfuscation mechanisms for different types of context information and integrate them into ALPACA. Finally, we want to investigate techniques increasing usability, such as an automatic mapping of applications to the privacy layers of our framework.

References

[1] F. Dorfmeister, S. Feld, C. Linnhoff-Popien, and S. Verclas, "Privacy-centric modeling and management of context information," in CENTRIC 2013, The Sixth International Conference on Advances in Human oriented and Personalized Mechanisms, Technologies, and Services, 2013, pp. 92–97.

[2] A. Amir, A. Efrat, J. Myllymaki, L. Palaniappan, and K. Wampler, "Buddy tracking - efficient proximity detection among mobile friends," Pervasive and Mobile Computing, vol. 3, no. 5, pp. 489–511, 2007.

[3] A. Küpper and G. Treu, "Efficient proximity and separation detection among mobile targets for supporting location-based community services," SIGMOBILE Mob. Comput. Commun. Rev., vol. 10, no. 3, pp. 1–12, Jul. 2006, last accessed on 2014-06-16. [Online]. Available: http://doi.acm.org/10.1145/1148094.1148096

[4] F. Dorfmeister, M. Maier, M. Schönfeld, and S. Verclas, "Smartbees: Enabling smart business environments based on location information and sensor networks," in 9. GI/KuVS-Fachgespräch "Ortsbezogene Anwendungen und Dienste". Springer, 2012, pp. 23–37.

[5] M. Ackerman, T. Darrell, and D. J. Weitzner, "Privacy in context," Human–Computer Interaction, vol. 16, no. 2-4, pp. 167–176, 2001.

[6] G. D. Abowd et al., "Towards a better understanding of context and context-awareness," in Handheld and ubiquitous computing. Springer, 1999, pp. 304–307.

[7] P. Makris, D. Skoutas, and C. Skianis, "A survey on context-aware mobile and wireless networking: On networking and computing environments' integration," Communications Surveys Tutorials, IEEE, vol. 15, no. 1, pp. 362–386, First 2013.

[8] A. Solanas, J. Domingo-Ferrer, and A. Martínez-Ballesté, "Location privacy in location-based services: Beyond ttp-based schemes," in Proceedings of the 1st International Workshop on Privacy in Location-Based Applications (PILBA), 2008, pp. 12–23.

[9] V. Sacramento, M. Endler, and F. N. Nascimento, "A privacy service for context-aware mobile computing," in Security and Privacy for Emerging Areas in Communications Networks, 2005. SecureComm 2005. First International Conference on. IEEE, 2005, pp. 182–193.

[10] M. Blount et al., "Privacy engine for context-aware enterprise application services," in Embedded and Ubiquitous Computing, 2008. EUC'08. IEEE/IFIP International Conference on, vol. 2. IEEE, 2008, pp. 94–100.

[11] H. Chen, T. Finin, and A. Joshi, "An intelligent broker for context-aware systems," in Adjunct proceedings of Ubicomp, vol. 3, 2003, pp. 183–184.

[12] L. Kagal, T. Finin, and A. Joshi, "A policy language for a pervasive computing environment," in Policies for Distributed Systems and Networks, 2003. Proceedings. POLICY 2003. IEEE 4th International Workshop on. IEEE, 2003, pp. 63–74.

[13] R. Wishart, K. Henricksen, and J. Indulska, "Context privacy and obfuscation supported by dynamic context source discovery and processing in a context management system," in Ubiquitous Intelligence and Computing. Springer, 2007, pp. 929–940.

[14] W. Apolinarski, M. Handte, D. Le Phuoc, and P. J. Marrón, "A peer-based approach to privacy-preserving context management," in Proceedings of the 7th international and interdisciplinary conference on Modeling and using context, ser. CONTEXT'11. Berlin, Heidelberg: Springer-Verlag, 2011, pp. 18–25, last accessed on 2014-06-16. [Online]. Available: http://dl.acm.org/citation.cfm?id=2045502.2045505

[15] A. Behrooz and A. Devlic, "A context-aware privacy policy language for controlling access to context information of mobile users," in Security and Privacy in Mobile Information and Communication Systems, ser. Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering, R. Prasad et al., Eds. Springer Berlin Heidelberg, 2012, vol. 94, pp. 25–39.

[16] J. Xie, B. P. Knijnenburg, and H. Jin, "Location sharing privacy preference: Analysis and personalized recommendation," in Proceedings of the 19th International Conference on Intelligent User Interfaces, ser. IUI '14. New York, NY, USA: ACM, 2014, pp. 189–198, last accessed on 2014-06-16. [Online]. Available: http://doi.acm.org/10.1145/2557500.2557504

[17] W. Bokhove, B. Hulsebosch, B. Van Schoonhoven, M. Sappelli, and K. Wouters, "User privacy in applications for well-being and well-working," in AMBIENT 2012, The Second International Conference on Ambient Computing, Applications, Services and Technologies, 2012, pp. 53–59.

[18] K. Sheikh, M. Wegdam, and M. Van Sinderen, "Quality-of-context and its use for protecting privacy in context aware systems." Journal of Software (1796217X), vol. 3, no. 2, 2008.

[19] A. Kofod-Petersen et al., "Implementing privacy as symmetry in location-aware systems," in Proceedings of the International Workshop on Combining Context with Trust, Privacy and Security (CAT 2008), G. Lenzini, B. Hulsebosch, S. Toivonen, and J.-M. Seigneur, Eds, vol. 371, 2008, pp. 1–10.

[20] L. Sweeney, "k-anonymity: A model for protecting privacy," International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems, vol. 10, no. 05, pp. 557–570, 2002.

[21] C. Bettini et al., "A survey of context modelling and reasoning techniques," Pervasive and Mobile Computing, vol. 6, no. 2, pp. 161–180, 2010.

[22] R. Reichle et al., "A comprehensive context modeling framework for pervasive computing systems," in Proceedings of the 8th IFIP WG 6.1 international conference on Distributed applications and interoperable systems, ser. DAIS'08. Berlin, Heidelberg: Springer-Verlag, 2008, pp. 281–295, last accessed on 2014-06-16. [Online]. Available: http://dl.acm.org/citation.cfm?id=1789074.1789105

[23] T. Strang, C. Linnhoff-Popien, and K. Frank, "Cool: A context ontology language to enable contextual interoperability," in Distributed applications and interoperable systems. Springer, 2003, pp. 236–247.

[24] M. Dürr, M. Maier, and F. Dorfmeister, "Vegas – a secure and privacy-preserving peer-to-peer online social network," in Privacy, Security, Risk and Trust (PASSAT), 2012 International Conference on and 2012 International Confernece on Social Computing (SocialCom), Sept 2012, pp. 868–874.

[25] W. Enck et al., "Taintdroid: An information-flow tracking system for realtime privacy monitoring on smartphones," in Proceedings of the 9th USENIX Conference on Operating Systems Design and Implementation, ser. OSDI'10. Berkeley, CA, USA: USENIX Association, 2010, pp. 1–6, last accessed on 2014-06-16. [Online]. Available: http://dl.acm.org/citation.cfm?id=1924943.1924971