PerManOnt: Permission Management Ontology for Smart Devices

S Vishal[1,*,†], Ayon Chakraborty[1,†] and P Sreenivasa Kumar[1,†]

[1]*Indian Institute of Technology Madras, India*

Abstract

The rapid growth of smart devices across various domains has led to an increase in the collection of personal information from these devices. Such data may be exploited for targeted advertising, personalizing content or for even more intrusive purposes such as identity theft and cyberbullying. Although smart devices offer users options to manage their privacy settings, each application typically requests multiple permissions and a smart device usually has many applications. Managing these permissions places a heavy cognitive burden on users, making it error-prone. There is also a need for the privacy management to be customizable based on the context in which the user is using the device.

In this paper, we provide a comprehensive ontology, PerManOnt, that conceptualizes a permission management system aiming to reduce the cognitive burden on users and improve expressivity of the permission decisions of the users. PerManOnt is an extension of an existing ontology for permission management and aims to make the system more evolvable. We analyze a real-world dataset and find that 90% of the users' privacy behaviour can be expressed with an average of approximately 4 preferences per user specified as per the proposed methodology. We also present the design of a permission management system, called PerManOnt Permission Manager (PerManOnt-PM), that utilizes the ontology. This system aims to convert the users' self-reported privacy preferences into actual context-based privacy settings as per the internal representation of the PerManOnt ontology.

Keywords

Smart Devices, Permission Management System, Applied Ontology

1. Introduction

Smart devices have become indispensable tools in our modern lives. Over the years, our reliance on these devices has increased rapidly. They have become an integral part of our daily lives, reshaping how we communicate, work, learn and entertain ourselves. However, this also means these devices collect and store a lot of personal information. Smart device applications often request access to our location, contact lists, camera, microphone, and even biometric data, such as fingerprints or facial recognition data. This data can be exploited for targeted advertising, personalized content, and even more intrusive purposes such as identity theft or cyberbullying.

Usually, the device's user is given options to control how their information is shared and stored. This can be done by stating a set of privacy settings representing their privacy concerns. For example, in Android, these settings correspond to 'Allow all the time', 'Ask every time', 'Allow when in use' and 'Don't allow'. However, such privacy self-management techniques are a burden on the cognitive abilities of the user and hence error-prone. Our cognitive limitations may sometimes inhibit us from making informed and rational decisions, especially when such decisions become increasingly complex.

Each application requests multiple permissions and typically a smartphone will have many applications. Selecting 'Allow all the time' will grant the permission for all subsequent uses, which is not ideal. The user is allowed to change the permission setting later but this feature is rarely used by the users[1]. Meanwhile, 'Ask every time' option prompts the user for permissions when it is requested every time. However, users do not pay attention or fail to understand prompts[2]. Also, on average users get 35

Proceedings of the Joint Ontology Workshops (JOWO) - Episode XI: The Sicilian Summer under the Etna, co-located with the 15th International Conference on Formal Ontology in Information Systems (FOIS 2025), September 8–9, 2025, Catania, Italy

*Corresponding author.

†These authors contributed equally.

cs22s010@cse.iitm.ac.in (S. Vishal); ayon@cse.iitm.ac.in (A. Chakraborty); psk@cse.iitm.ac.in (P. S. Kumar)

permission requests per hour of smartphone usage, which means manually answering them will lead to prompt fatigue[3]. This heavily increases the cognitive burden on users for managing all of their privacy settings.

Further, users' decisions are susceptible to various biases. For example, most users tend to focus on certain permissions and end up accepting other permissions even if they are unreasonable[4]. In fact, most users do not pay attention to permissions during the installation of an application[5]. Users struggle to infer the responsibilities of permission groups and accordingly make mistakes due to wrong assumptions[6]. Users also experience warning fatigue when they receive too many warnings at install time, and sometimes even runtime requests are accepted because users do not switch their attention from the task to reviewing the request[2].

Current research works try to reduce this cognitive burden by automating the user decisions. These works vary from locally trained models, recommendation systems based on collaborative filtering, clustering algorithms to identify similar user profiles. However, automating these privacy decisions also takes away the users' control. Moreover, research suggests users' self-reported privacy concerns do not always align with their privacy behavior[7] and hence using user behavior alone to predict user decisions may not fully address the problem.

Moreover, the user's security concerns will also depend on their surroundings. For example, a user might find it very helpful to use a voice assistant when they are in their office. The same user might feel unsafe about using it at home. In such a situation, the user needs to grant microphone permission to the voice assistant when they are at the office but deny the microphone permission when they are at home. In these cases, a context-aware permission sharing scheme would be helpful. However, doing so manually by the user would also place a heavy cognitive burden on the user. Some of the current works also found contextual cues to be helpful in predicting users' privacy preferences[8][9].

Even though the above concerns are applicable to all smart devices, we will be focusing on Android smartphones for the rest of the paper to make the explanations concrete.

In this paper, we aim to provide an enhanced ontology that conceptualizes a permission sharing system which is both context-aware and gives more expressive power to the user. This conceptualization also tries to make up for user's cognitive limitations by allowing users to vary how general or specific their preferences are. This ontology is an extension of a task-specific ontology created by Johansson et al.[10] called Android Permission Sharing Ontology (APSO). APSO ontology conceptualizes each privacy setting as `Preferences`. These Preferences determine whether to grant or deny permissions for applications in a particular context. However, we find that the APSO system is not stable under change and hence is not scalable. This is explained further in Section 3.2. We propose to appropriately extend the ontology to devise ways to automatically detect conflicts and correlations among the given preferences and notify the user accordingly.

In summary, The contributions of our work are as follows:

- We provide a comprehensive ontology called PerManOnt that conceptualizes a permission management system which is both context-aware and improves the expressivity of the users' privacy concerns and evaluated this expressivity of our conceptualization using a real-world dataset.
- We propose the design of a system called PerManOnt-PM which can convert the user's self-reported privacy preferences into actual privacy settings in the new conceptualization provided by PerManOnt and hence reduce the cognitive burden of the user in creating and managing his/her privacy settings.

2. Motivation

Almuhimedi et al.[11] showed that educating users about the sensitive information being shared was successful in making 95% of participants reassess their permissions, with 58% of those participants further restricting some of their permissions. Shen et al.[6] found that only a very small percentage (6.1%) of users can accurately infer the scope of permission groups from the system-provided information. Felt et al.[2] showed only 17% of users paid attention to permissions during a given installation and

only 3% of participants were able to fully explain the meaning of a permission. Tahaei et al.[12] found that developers sometimes requested multiple permissions due to confusion about the scope of certain permissions or third-party library requirements. Furthermore, 57.9% of end users do not know how to revoke permissions for an application.

Another work by Alsoubai et al.[4] identified four profiles that reflect the different privacy management strategies, perceptions, and intentions of Android users, going beyond the binary decision to share or withhold information via mobile apps. They identified four unique profiles: Privacy Balancers (49.74% of participants), Permission Limiters (28.68% of participants), App Limiters (14.74% of participants) and Privacy Unconcerned (6.84% of participants). Inspired by the observations of this study, researchers tried to identify user profiles by using clustering algorithms to predict user privacy decisions[13][14][3].

Other works try to reduce cognitive burden by automating the privacy decisions. Some are trained locally to predict users' privacy preferences[8][9]. These works require intensive user input which they acquire by prompting the user continuously. This creates prompt fatigue for the user. Other works recommend privacy decisions to users based on collaborative filtering[15][16][17][18]. These works depend on users sharing their privacy behavior to a central server which is a privacy risk. Most of these works either deal with sharing privacy decisions to a centralized server or are trained locally. Training locally requires extensive user input through prompts, which can lead to prompt fatigue. On the other hand, sharing privacy decisions to a centralized server poses serious security risks — if the server is compromised or trained with spurious data. Additionally, centralization expands the attack surface and increases both system complexity and operational costs. Also, ML models are stochastic in nature and cannot deterministically predict outcomes. This might become an issue if the predicted privacy decision does not align with user intent. Automating these decisions also takes away the users' control over their privacy. While works like [9] do allow decisions to be reviewed and changed, this once again increases the cognitive burden of the user. Moreover, unlearning a data point is non-trivial for these models. Research also suggests that user's self-reported privacy concerns do not always align with their privacy behavior[7], popularly known as the privacy paradox. Since, most of these works try to mimic user data and considering users do not understand permissions completely, these models only increase the mistakes already made by the users. To the best of our knowledge, there is no existing work that tries to take the users' self-reported privacy concerns into consideration when trying to come up with solutions that reduce the cognitive burden of the user in making privacy decisions.

While there is a need for reducing cognitive burden on the user for making privacy decisions, automatically predicting preferences may not be an optimal solution. Hence, there is a need to provide a system that makes it easier for the user to make these privacy decisions while also giving him/her complete control and flexibility to create and modify these decisions. The user should also be able to make privacy decisions based on his/her context.

The contributions of our work (listed at the end of Section 1) are motivated by the above considerations.

3. Related Work - Android Permission Sharing Ontology (APSO)

3.1. Overview

Johansson et al.[10] created a task-specific ontology that increases Android users' options in making permission-sharing decisions while limiting the effort required to make these decisions. This ontology enables users to create general or specific permission-sharing preferences and specify when a preference should be considered based on contextual information. Users can request insights into their preferences using SPARQL[19] queries. Figure 1 shows a graphical representation of key concepts in the ontology.

The concept `Preference` is used to encapsulate user preferences. By conceptualizing a permission-sharing preference as a concept, they capture knowledge about its owner, action, target and context. The context of the user is conceptualized using the concept `DescriptiveContext`. The context of the user encompasses everything external to the application that can change over time and to which applications must adapt. This context can be based on time, location or social surroundings. For

example, a user might have different preferences when they are at work versus while traveling. In this case, ‘At_Work’ is a descriptive context and ‘Traveling’ is a different descriptive context. The concept `Preference` is linked with this `DescriptiveContext` using the `onDescriptiveContext` role. The concept `PreferenceObject` is used to represent the permission or permission groups that the preference regulates. A specific preference entity is linked to a permission or a permission group using the role `onPreferenceObject`. The concept `TargetContext` is used to represent the application or a group of applications that a preference is trying to regulate. A specific preference entity is linked to an application or a group of applications using the role `onTargetContext`.

Target contexts, descriptive contexts and preference objects can be used to identify when a specific preference will become active. These concepts together are called conditional contexts. In general, a conditional context refers to any information that can occur in a user’s actual context, which can be used to decide when a preference is active.

The ontology has provision to use application categories as target context instead of the individual applications. The role `hasPart` is used to link the application categories to its individual applications. The same `hasPart` role is also used to link permission groups to its individual permissions. The inverse of the `hasPart` role is the `partOf` role.

The concept `PreferenceAction` is used to specify what action needs to be executed when the preference is active. This action could be one of ‘Share’, ‘Not_Share’, ‘Prompt_User’ or ‘Prompt_Group’. If the action is ‘Share’, then the permission is shared with the requesting application. If the action is ‘Not_Share’, then the permission will not be shared with the requesting application. If the action is ‘Prompt_User’, then the user will be prompted to allow or deny the permission at runtime. If the action is ‘Prompt_Group’, then the user group will be prompted to allow or deny the permission at runtime. A specific preference entity is linked to an action using the role `onPreferenceAction`.

The concept `Policy` organizes a person’s or group’s permission sharing preferences and enables users to establish individual or shared preferences. A policy is linked with all of its preferences with the `hasPreference` role.

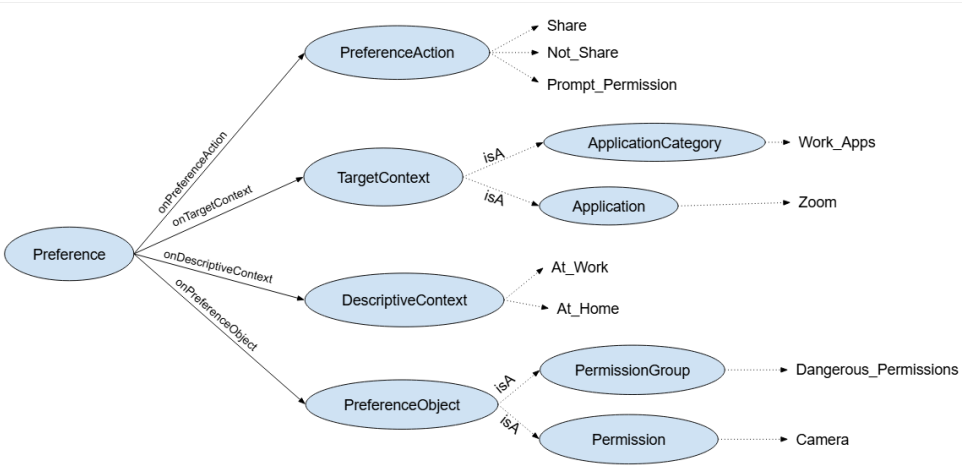

Figure 1: This figure shows a graphical representation of the Android Permission Sharing Ontology (APSO)

When an application requests a permission, SPARQL queries are used to identify preferences that match the conditional context of this request. The `PreferenceAction` of the corresponding Preference is then returned. For example, Mary wants to share all permissions when she is at work with the apps Zoom and Authenticator. In this case, the preference has the descriptive context ‘At_Work’ which indicates that she is at her workplace during working hours. The target contexts are the applications Zoom and Authenticator which can be grouped into an Application Category called ‘Work_Apps’. The preference object is essentially all the permissions which can be indicated by a permission group containing all the permissions (called ‘Universal_Object’). Now, when the Zoom application requests the Camera permission at work, this preference becomes active and the permission is shared with the

Table 1
Competency Questions for the PerManOnt Ontology

Basic Functionalities:
• What action needs to be performed when an application asks for a permission when the user is in a particular context?
• What preference is responsible for a particular action when the application requests a permission?
• Which applications can access a particular resource and in which context?
• Which permissions are accessible to an application and in which context?
PerManOnt Functionalities:
• When a new preference is added, what preferences conflicts/correlates with it?
• What are the overlapping components in conflicting/correlated preferences when adding a new preference?
• Which preference will be executed when multiple Descriptive Contexts are active?

Zoom application.

There might be cases where more than one preference becomes active for a particular conditional context. In such cases, the data property `priority` is used to identify which preference should be active. The preference with the higher priority value becomes active.

3.2. Limitations of APSO

APSO system shares many similarities with a firewall rule base. It defines rules to perform an action based on the context. Similarly in a firewall rule base, rules are defined to perform an action based on source and destination IP addresses. They also share the same issues as firewall rule bases. Firewall rule bases are known for having evolvability issues. As the rule base grows, inserting a rule in the correct order becomes increasingly cumbersome. Order sensitive rules are identified as one of the key reasons for this non-evolvability of firewall rule bases[20]. By using the data property *priority* to order the preferences, the APSO system also becomes order dependant. As the number of preferences increases, it becomes increasingly difficult to insert a new preference with the correct *priority* value just like how inserting a new firewall rule will require the user to analyze the entire rule base to specify the correct order. Also, when the actions of a preference need to be changed or removed, all other preferences that also perform the same action (for the same context) need to be changed. This leads to evolvability issues. In Normalized Systems theory [21], this is known as combinatorial effect i.e., the effect that occurs when the impact of a change is proportional to the nature of the change and the system's size. Such a system is unstable under change.

Moreover, the current ontology[10] does not specify how the descriptive context is identified. According to this work, the context could be location, time or surroundings. However, there may be other contexts that can affect a user's preferences. For example, users might want apps to stop allowing background permissions when the battery percentage is low. In such cases, battery percentage can also be a context. The descriptive context is currently identified as predefined individuals. However, there is potential for the descriptive context to be conceptualized in the ontology itself.

4. PerManOnt

In this section, we extend the APSO ontology and propose a new ontology called PerManOnt that can be used to improve privacy management for users, reduce the overall cognitive burden and also increase the expressivity of users' privacy decisions.

Recall that user preferences are captured by the concept `Preference`. Each preference has a conditional context and a preference action. The conditional context consists of a descriptive context that specifies the user's surroundings, a preference object representing the set of permissions that this preference regulates and a target context indicating the set of applications the preference is concerned with. The `PreferenceAction` concept specifies what action should take place (one of 'Share', 'Not Share', 'Prompt User', 'Prompt Group') when the conditional context becomes active.

As more and more preferences are added, it will inevitably lead to conflicts and redundancies. The previous work attempted to resolve this by adding a data property called `priority`. This would mean that users should be aware of the priorities of all the preferences when adding a new preference. This would impose a significant cognitive burden on the user. This also makes the system non-evolvable. Hence, drawing inspiration from the series of works by Haerens et al.[20][22][23], we try to make the preferences disjoint from each other.

We also provide a more generic conceptualization of Descriptive Contexts. Table 1 gives the competency questions that will be used to evaluate the ontology.

In Section 4.1, we provide a conceptualization of descriptive contexts. In Sections 4.2, 4.3 and 4.4, we propose using SWRL[24] rules to identify these conflicts and redundancies among preferences.

4.1. Conceptualizing descriptive contexts

In the APSO ontology, the `DescriptiveContext` concept has fixed instances like 'At_Home' and 'At_Work'. It does not specify how the user's context is identified or how a particular `DescriptiveContext` can be defined by the user. It only indicates that it could be based on location, time or social surroundings. However, there is a need to provide additional ways of capturing the user's context.

In general, the context can be thought of as consisting of more than one specific situation such as being at work place, running on low battery, etc. Different sensors and other settings on the device can help us identify these specific situations. Hence, we conceptualize the specific situations as context indicators and propose a concept called `ContextIndicator` as a collection of specific context indicators that are available in the system. For example, 'At_Home', 'At_Work', 'Traveling', 'Low_Battery' etc.. are all members of this concept. We expect that each of these context indicators will be set to active/inactive by a subsystem that continuously monitors the various sensor values of the smart device. For the purpose of this paper, we assume the existence of such a subsystem and leave further study/research on that to our future work. We also define a concept called `CurrentContext`. `CurrentContext` will contain all the `ContextIndicators` that are active at that point in time. The `ContextIndicators` can then be used by the user to define their personalized `DescriptiveContext`. The `DescriptiveContext` can be defined as a collection of one or more `ContextIndicators`. Note that this is a generalization of the existing framework as the current approach assumes that only one of the context indicators is active at any point and this is not realistic. In practice a smart device can be in multiple contexts such as {At_Work, Low_Battery}, {At_Work, In_Meeting} and our framework allows it. The `DescriptiveContext` will be linked to its `ContextIndicators` by the `hasContextIndicator` role. A particular `DescriptiveContext` becomes active if all the `ContextIndicators` that are part of that `DescriptiveContext` are in the `CurrentContext`. Inorder to indicate whether a `ContextIndicator` is active, we add a boolean data property `isActive` which becomes true when the context is active. For example, when the user's device is in low battery while traveling, they might need to restrict the use of certain resources that might accelerate battery drainage. In this case, 'Low_Battery' and 'Traveling' may have already been identified as `ContextIndicators`. The user can then define a `DescriptiveContext` that becomes active if both 'Low_Battery' and 'Traveling' are in the `CurrentContext`. The user can then use this `DescriptiveContext` to create a `Preference` to deny certain permissions when the device is in this `DescriptiveContext`.

Note that this also allows two or more `DescriptiveContexts` to be active at the same time. This might introduce new conflicts among `Preferences`. To avoid this, we first identify `ContextIndicators` that can potentially be active at the same time. For example, 'At_Work' and 'At_Home' can never be active simultaneously whereas 'At_Work' and 'Low_Battery' can be active simultaneously. We introduce a symmetric role `canNotOccurWith` to link such `ContextIndicators`. For example, 'At_Home' depends on the location sensor, whereas 'Low_Battery' depends on the battery status of the device. These two sensors are not dependent on each other and therefore can occur simultaneously, but not 'At_Home' and 'At_Work', as they both depend on the location sensor and specify different locations. If two `DescriptiveContexts` do not have any `ContextIndicators` that have a `canNotOccurWith` role

between them, they can potentially be active at the same time. Once such `DescriptiveContexts` are identified, the user can then define the priority among those `DescriptiveContexts`, which is conceptualized using the data property `level`. In this way, the user does not need to order the entire set of `DescriptiveContexts` but only those that can potentially be active simultaneously.

4.2. Conflicting preferences

If a request matches more than one preference and they have different actions specified by the user, then we say that there is a conflict between the preferences. Previously, these conflicts were resolved using a data property called `priority`. The preference with the highest priority value was made active and the action specified in that preference was executed. However, as mentioned in Section 3.2, this approach increases the cognitive burden of the user. Therefore, we propose using SWRL rules to identify such conflicting preferences and notify the user, who can then take the necessary action to resolve the conflict.

For example, suppose there is a preference A that restricts background permissions for all applications and there is another preference B that allows all permissions when the requesting application is developed by Microsoft. Then, when a Microsoft app does request for a background permission, it is unclear if preference A becomes active or preference B becomes active. And since both have *different* actions, these are conflicting preferences.

To detect such conflicts, we can extend the APSO ontology and add a property called `conflicts` to indicate whether two preferences conflict with each other. We define `conflicts` as a symmetric property. We can use a SWRL rule to add a `conflicts` role between two preferences if they share the same descriptive context, have applications that are common in both their target context and have permissions that are common in both their preference objects but have different actions.

When a new preference is added or an existing preference is modified, this SWRL rule is used to identify any conflicts with the existing preferences. When such conflicting preferences are identified, they will be notified to the users, who can then decide the necessary action. A SPARQL query can be used to identify overlapping components in the conflicting preferences and display them to the user. The user can then decide to keep or discard these components from either or both of the preferences. The preference will only be added or modified when the conflict is resolved. This keeps the system stable when adding new preferences. The rule that detects a pair of conflicting preferences can be specified using SWRL as given in Rule 1 of Table 2.

For instance, assume there is a preference named 'Henry_1' that shares all permissions with apps from the application category 'CC_Microsoft_Apps' in all descriptive contexts. Suppose we try to add another preference named 'Henry_2' that does not share 'Access_Background_Locations' with any application under any descriptive context. Then 'Henry_1' and 'Henry_2' are said to conflict with each other. Consequently, these two preferences will have a 'conflicts' role between them. The conflict occurs when the applications in the category 'CC_Microsoft_Apps' requests for the 'Access_Background_Locations' permission. After identifying this conflict, the user can then change 'Henry_2' preference to restrict that permission only to a category of applications that contains all the applications except those in 'CC_Microsoft_Apps'.

4.3. Correlated preferences

If a request matches more than one preference and they have the same actions specified by the user, then we say that there is a correlation between the preferences.

For example, if there is a preference A that allows Location permission for all applications, then having a preference B that allows the Location permission to a Map application would be redundant. Here, preference A and preference B are said to be correlated.

We can extend the APSO ontology and add a symmetric role called `isCorrelatedWith` to indicate that one preference is correlated with another. We can use a SWRL rule to add a `isCorrelatedWith` role between two preferences if they share the same descriptive context, have applications that are

Table 2
List of SWRL Rules used in this ontology for reasoning

Rule Number	Rule
Rule 1 (Conflicting preferences)	Preference(?x1) ∧ Preference(?x2) ∧ onDescriptiveContext(?x1,?y1) ∧ onDescriptiveContext(?x2,?y1) ∧ onTargetContext(?x1,?y2) ∧ onTargetContext(?x2,?y22) ∧ onPreferenceObject(?x1,?y3) ∧ onPreferenceObject(?x2,?y33) ∧ Application(?z1) ∧ hasPart(?y2,?z1) ∧ hasPart(?y22,?z1) ∧ Permission(?z2) ∧ hasPart(?y3,?z2) ∧ hasPart(?y33,?z2) ∧ onPreferenceAction(?x1,?y4) ∧ onPreferenceAction(?x2,?y5) ∧ differentFrom(?y4,?y5) -> conflicts(?x1,?x2)
Rule 2 (Correlated preferences)	Preference(?x1) ∧ Preference(?x2) ∧ differentFrom(?x1,?x2) ∧ onDescriptiveContext(?x1,?y1) ∧ onDescriptiveContext(?x2,?y1) ∧ onTargetContext(?x1,?y2) ∧ onTargetContext(?x2,?y22) ∧ onPreferenceObject(?x1,?y3) ∧ onPreferenceObject(?x2,?y33) ∧ Application(?z1) ∧ hasPart(?y2,?z1) ∧ hasPart(?y22,?z1) ∧ Permission(?z2) ∧ hasPart(?y3,?z2) ∧ hasPart(?y33,?z2) ∧ onPreferenceAction(?x1,?y4) ∧ onPreferenceAction(?x2,?y4) -> isCorrelatedWith(?x1,?x2)
Rule 3 (Conflicting preferences based on dependencies)	Preference(?x1) ∧ Preference(?x2) ∧ onDescriptiveContext(?x1,?y1) ∧ onDescriptiveContext(?x2,?y1) ∧ onTargetContext(?x1,?y2) ∧ onTargetContext(?x2,?y22) ∧ onPreferenceObject(?x1,?y3) ∧ onPreferenceObject(?x2,?y33) ∧ hasPart(?y2,?z1) ∧ hasPart(?y22,?z1) ∧ hasPart(?y3,?z2) ∧ hasPart(?y33,?z3) ∧ dependsOn(?z2,?z3) ∧ onPreferenceAction(?x1,?y4) ∧ onPreferenceAction(?x2,?y5) ∧ sameAs(?y4,Act_Share) ∧ differentFrom(?y4,?y5) -> conflicts(?x1,?x2)

common in both their target contexts and have permissions that are common in both their preference objects and have the same actions. When adding a new preference, this SWRL rule is used to identify such correlated preferences. When such correlated preferences are identified, they will be notified to the users, who can then decide the necessary action. A SPARQL query can be used to identify overlapping components in the correlated preferences and display them to the user. The user can then decide to keep or discard these components from either or both of the preferences. The preference is only added or modified when these correlations are resolved. This keeps the system stable when removing a preference as no request matches more than one preference. The rule that detects a pair of correlated preferences can be specified using SWRL as given in Rule 2 of Table 2.

4.4. Conflicts based on dependent permissions

There may be cases where permission A is dependent on another permission B for it to function. If access to permission B is restricted, this will create a conflict.

An example of this is ACCESS_COARSE_LOCATION permission, which determines location using WiFi and mobile data. This means, an application trying to use this permission will also require the INTERNET permission. Now, if there is a preference that allows coarse location access but there is another preference that does not allow internet access for the same context and application, this will create a conflict.

To address this, we add a `dependsOn` role to represent such dependencies. We use a SWRL rule, which will add a `conflicts` role if there are two preferences sharing the same descriptive context, have common applications in their target context and some permissions of preference A depend on permissions in preference B while the preference actions of A and B conflict with each other. Preference action of A could be 'share/prompt_user' and preference action of B could be 'not_share/prompt_user'. The corresponding SWRL rule is given in Rule 3 of Table 2.

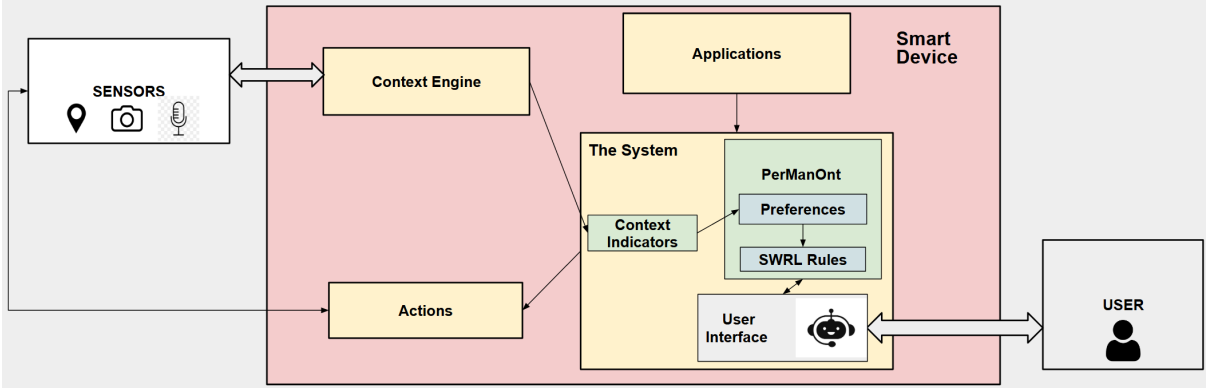

Figure 2: Displays the architecture for the proposed PerManOnt-PM system

4.5. Size Comparisons of the two systems

Let m be the number of permissions and n be the number of applications. The total number of unique preferences possible in the APSO system for a particular `DescriptiveContext` will be all possible combinations of application categories and permission groups. Since, there is no mechanism to identify overlapping or conflicting components in these preferences, the total number of preferences will be the product of all possible non-empty subsets of applications and all possible subsets of permissions, and the number of action choices (grant/deny/prompt). This is given by $3\,(2^m - 1)\,(2^n - 1)$.

Since, the PerManOnt ontology only allows disjoint preferences to exist, the total number of preferences possible in the PerManOnt system for a particular `DescriptiveContext` is given by $3.m.n$. This is significantly lesser to manage compared to the APSO ontology.

5. Design and Implementation

We implemented the operational version of this ontology by using the ontology editor called Protégé[25]. The ontology is designed using RDF, RDFS, OWL. We utilized the TBox, RBox and ABox axioms that were used in the previous work[10] and we have added the concepts and roles that are introduced in our work. The PerManOnt ontology is publicly available in the URI - https://w3id.org/permanont.

As part of the future work, the Permission Management Ontology will be used to build an application, the architecture for which is shown in Figure 2. A task-oriented user-interface component will be used to formulate user preferences by interacting with the user. There will be a set of default preferences which will simulate the default Android behavior. The user will be allowed to create their preferences in addition to that. The formulated user preferences are then checked for conflicts/redundancies with the existing preferences using the PerManOnt ontology. The preferences are then added to the knowledge base of the ontology. The context engine automatically identifies user contexts based on sensor data of the user's device. These contexts are used as `ContextIndicators` in the PerManOnt-PM system. The user will employ this system to create and manage preferences for different applications. These preferences will then be used to respond to permission requests from applications, with the necessary actions executed to access both sensitive and non-sensitive resources.

6. Evaluation

The evaluation of our proposed approach involves two components - to evaluate the consistency of the ontology and then to evaluate if the proposed approach actually improves the permission management for the users. In order to evaluate the ontology, we use the Pellet reasoner in Protégé[25] and found our ontology to be consistent. We also used the FOOPS![26] tool to check if the ontology is in compliance with the FAIR principles. We use the competency questions in Table 1 to validate the ontology. We

used SPARQL queries[1] to validate whether the ontology can answer these competency questions. We found that the ontology was able to satisfy the functional requirements that were captured in these competency questions. Although the amount of data involved may impose a cognitive burden on the user, it is not expected to be computationally expensive.

In order to evaluate the effectiveness of our approach in capturing users' behavior, we make use of the dataset collected in the NGI Trust project COP-MODE[27]. The dataset contains information about the permission requests, the users' response to the requests and contextual data regarding the phone state and the user context, which includes information such as background and foreground running apps, network status, semantic location, and devices in neighborhood. The data was collected through several campaigns occurring between end of July 2020 and end of March 2021 in Portugal. The dataset comprises 93 users and 2180302 permission requests, from which 65261 were manually answered by the users, with an average of 701.73 answered requests per user. From the 65261 permissions, 66% were accepted and 33% denied.

In order to prove that our conceptualization of the system does make the permission sharing system more expressive, we show that with a relatively small set of user preferences (specified as per the PerManOnt framework), we will be able to capture the essence of users' privacy behavior in this real-world dataset.

We analyze the dataset to see if we are able to extract a set of preferences for each user that will allow us to model the user's behavior. First, we map the available information into instances of the ontology. Then, we use these instances for each user to formulate preferences based on their responses to permission requests. To formulate these preferences, we construct a decision tree for each user with the features available (such as requesting application, permission requested, semantic location, timestamp) in the dataset[2]. Decision trees are chosen as they are interpretable and it is easy to adapt the paths in the tree to the preferences in our ontology. We interpret each decision path in the decision tree as a preference with the internal nodes in the path as the context in which the preference is active. Since, decision trees give us distinct decision paths, it is bound to give us disjoint preferences (hence no conflicts or correlations). If a decision path is not able to convincingly predict an action (grant/deny with the accuracy threshold being $> 90\%$), we select *prompt* as the action. We formulated such preferences for each of the 93 users. We also added a default preference for all the users which will prompt the user when a permission is requested. This default preference will be executed if none of the other preferences match the request. We exclude this default preferences in our evaluations since it is also the default behavior.

In summary, we are able to model 90% of the user behavior with the formulated preferences (excluding the default preference). On average, only 4.13 preferences were required for each user to express their permission sharing behavior in our conceptualization. These modeled the permission-sharing behavior of the user with 94.57% accuracy.

Hence, an average of ≈ 4 preferences were enough to express a user's permission sharing behaviour. At least one contextual feature was present in 44.53% of these identified preferences, which corroborates the fact that contextual cues are important in privacy decision making. In contrast, a traditional permission sharing system will require the user to manage each permission for each application separately for each context. Or the user will have to be prompted always for the permission request. Moreover, our approach allows users to change their behavior and correspondingly change their preference. This level of flexibility and customizability is not available in ML based approaches.

7. Conclusions and Future Work

In this paper, we recognized the increasing cognitive burden on users regarding privacy management and analyzed an existing ontology-based approach to address this problem. We identified certain limitations of that ontology and extended it to address those limitations. The extended ontology is

[1] https://raw.githubusercontent.com/CS22S010/PerManOnt/refs/heads/main/PerManOnt_Competency_Q.txt
[2] https://cop-mode.dei.uc.pt/dataset

called PerManOnt and it has several novel additions that can facilitate the detection of conflicts and redundancies in permission settings. It also includes a new notion called *context indicator* and provides a generalized notion of the descriptive context of a smart device. We evaluated the effectiveness of such a system in expressing users' permission sharing preferences using a real-world dataset and obtained encouraging results.

We also propose the design of a permission management system, PerManOnt-PM, based on the proposed ontology. This system aims to convert the users' self-reported privacy preferences into actual context-based privacy settings as per the internal representation of the PerManOnt ontology and also detect conflicts and redundancies among the settings. As part of our future work, we propose to implement PerManOnt-PM and carry out usage trials.

Declaration on Generative AI

During the preparation of this work, the author(s) used ChatGPT in order to: grammar and spelling check, paraphrase and reword. After using this tool/service, the author(s) reviewed and edited the content as needed and take(s) full responsibility for the publication's content.

References

[1] P. Andriotis, G. Stringhini, M. A. Sasse, Studying users' adaptation to android's run-time fine-grained access control system, Journal of information security and applications 40 (2018) 31–43.

[2] A. P. Felt, E. Ha, S. Egelman, A. Haney, E. Chin, D. Wagner, Android permissions: User attention, comprehension, and behavior, in: Proceedings of the eighth symposium on usable privacy and security, 2012, pp. 1–14.

[3] A. Brandão, R. Mendes, J. P. Vilela, Prediction of mobile app privacy preferences with user profiles via federated learning, in: Proceedings of the Twelfth ACM Conference on Data and Application Security and Privacy, 2022, pp. 89–100.

[4] A. Alsoubai, R. Ghaiumy Anaraky, Y. Li, X. Page, B. Knijnenburg, P. J. Wisniewski, Permission vs. app limiters: profiling smartphone users to understand differing strategies for mobile privacy management, in: Proceedings of the 2022 CHI Conference on Human Factors in Computing Systems, 2022, pp. 1–18.

[5] N. Alshomrani, S. Furnell, Y. He, Assessing user understanding, perception and behaviour with privacy and permission settings, in: International Conference on Human-Computer Interaction, Springer, 2023, pp. 557–575.

[6] B. Shen, L. Wei, C. Xiang, Y. Wu, M. Shen, Y. Zhou, X. Jin, Can systems explain permissions better? understanding users' misperceptions under smartphone runtime permission model, in: 30th USENIX Security Symposium (USENIX Security 21), 2021, pp. 751–768.

[7] S. Kokolakis, Privacy attitudes and privacy behaviour: A review of current research on the privacy paradox phenomenon, Computers & security 64 (2017) 122–134.

[8] K. Olejnik, I. Dacosta, J. S. Machado, K. Huguenin, M. E. Khan, J.-P. Hubaux, Smarper: Context-aware and automatic runtime-permissions for mobile devices, in: 2017 IEEE Symposium on Security and Privacy (SP), IEEE, 2017, pp. 1058–1076.

[9] P. Wijesekera, J. Reardon, I. Reyes, L. Tsai, J.-W. Chen, N. Good, D. Wagner, K. Beznosov, S. Egelman, Contextualizing privacy decisions for better prediction (and protection), in: Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems, 2018, pp. 1–13.

[10] E. A. Johansson, APSO: an Android Permission-Sharing Ontology, Master's thesis, University Of Oslo, 2022.

[11] H. Almuhimedi, F. Schaub, N. Sadeh, I. Adjerid, A. Acquisti, J. Gluck, L. F. Cranor, Y. Agarwal, Your location has been shared 5,398 times! a field study on mobile app privacy nudging, in: Proceedings of the 33rd annual ACM conference on human factors in computing systems, 2015, pp. 787–796.

[12] M. Tahaei, R. Abu-Salma, A. Rashid, Stuck in the permissions with you: Developer & end-user perspectives on app permissions & their privacy ramifications, in: Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems, 2023, pp. 1–24.
[13] O. R. Sanchez, I. Torre, Y. He, B. P. Knijnenburg, A recommendation approach for user privacy preferences in the fitness domain, User Modeling and User-Adapted Interaction 30 (2020) 513–565.
[14] B. Liu, M. S. Andersen, F. Schaub, H. Almuhimedi, S. A. Zhang, N. Sadeh, Y. Agarwal, A. Acquisti, Follow my recommendations: A personalized privacy assistant for mobile app permissions, in: Twelfth symposium on usable privacy and security (SOUPS 2016), 2016, pp. 27–41.
[15] B. Rashidi, C. Fung, T. Vu, Dude, ask the experts!: Android resource access permission recommendation with recdroid, in: 2015 IFIP/IEEE international symposium on integrated network management (IM), IEEE, 2015, pp. 296–304.
[16] Y. Zhao, J. Ye, T. Henderson, Privacy-aware location privacy preference recommendations, in: Proceedings of the 11th international conference on mobile and ubiquitous systems: Computing, networking and services, 2014, pp. 120–129.
[17] J. Xie, B. P. Knijnenburg, H. Jin, Location sharing privacy preference: analysis and personalized recommendation, in: Proceedings of the 19th international conference on Intelligent User Interfaces, 2014, pp. 189–198.
[18] Q. Ismail, T. Ahmed, A. Kapadia, M. K. Reiter, Crowdsourced exploration of security configurations, in: Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems, 2015, pp. 467–476.
[19] E. Prud'hommeaux, A. Seaborne, Sparql query language for rdf. w3c recommendation, w3c, Retrieved November 16 (2008) 2009.
[20] G. Haerens, P. De Bruyn, Using normalized systems to explore the possibility of creating an evolvable firewall rule base, management 14 (2019) 15.
[21] H. Mannaert, J. Verelst, P. De Bruyn, Normalized systems theory: from foundations for evolvable software toward a general theory for evolvable design, 2016.
[22] G. Haerens, H. Mannaert, Investigating the creation of an evolvable firewall rule base and guidance for network firewall architecture, using the normalized systems theory, International Journal on Advances in Security 13 (2020) 1–16.
[23] G. Haerens, Ontological analysis of the evolvability of the network firewall rule base, in: Proceedings of the 20th CIAO! Doctoral Consortium, and Enterprise Engineering Working Conference Forum 2020, co-located with 10th Enterprise Engineering Working Conference (EEWC 2020), Bozen/Bolzano, Italy, September 28th, October 19th and November 9th-10th, 2020, 2020, pp. 1–15.
[24] I. Horrocks, P. F. Patel-Schneider, H. Boley, S. Tabet, B. Grosof, M. Dean, et al., Swrl: A semantic web rule language combining owl and ruleml, W3C Member submission 21 (2004) 1–31.
[25] M. A. Musen, The protégé project: a look back and a look forward, AI Matters 1 (2015) 4–12. URL: https://doi.org/10.1145/2757001.2757003. doi:10.1145/2757001.2757003.
[26] D. Garijo, O. Corcho, M. Poveda-Villalón, Foops!: An ontology pitfall scanner for the fair principles., in: ISWC (Posters/Demos/Industry), 2021.
[27] R. Mendes, A. Brandão, J. P. Vilela, A. R. Beresford, Effect of user expectation on mobile app privacy: a field study, in: 2022 IEEE international conference on pervasive computing and communications (PerCom), IEEE, 2022, pp. 207–214.