# Capturing policies for fine-grained access control on mobile devices

Prajit Kumar Das, Anupam Joshi and Tim Finin
Department of Computer Science and Electrical Engineering
University of Maryland, Baltimore County
{prajit1, joshi, finin}@umbc.edu

***Abstract*—As of 2016, there are more mobile devices than humans on earth. Today, mobile devices are a critical part of our lives and often hold sensitive corporate and personal data. As a result, they are a lucrative target for attackers, and managing data privacy and security on mobile devices has become a vital issue. Existing access control mechanisms in most devices are restrictive and inadequate. They do not take into account the context of a device and its user when making decisions. In many cases, the access granted to a subject should change based on context of a device. Such fine-grained, context-sensitive access control policies have to be personalized too. In this paper, we present the MITHRIL system, that uses policies represented in Semantic Web technologies and captured using user feedback, to handle access control on mobile devices. We present an iterative feedback process to capture user specific policy. We also present a policy violation metric that allows us to decide when the capture process is complete.**

## I. INTRODUCTION

Mobile devices are a ubiquitous commodity. Both Android [1] and Apple [2] ecosystems claim to have more than a billion active devices today. However, unlike iOS, Android is open source. This allows both black hat hackers (who violate computer security for personal gain) as well as white hat hackers (ethical hackers), ample opportunity to study strengths and weaknesses of the system. Due to advantages of being open-sourced, we use Android as our primary research prototyping platform.

As of February 2016, Statista [3] estimated number of apps in Google Play Store at two million. Android apps are also available through other outlets like Amazon App Store and Samsung Galaxy Apps [4]. As the number of Internet-connected mobile users and number of apps and app-stores increase, not only are mobile devices and apps used more often, they also carry confidential data thus making them a potential target for attacks. A 2014 McAfee Labs report [5] predicted that mobile technologies would see an escalation of attacks, due to openly available mobile malicious source code. Malware are not the only threats faced by mobile users. A Google taxonomy provides us with additional threats that users face through Potentially Harmful Apps (PHA) [6], such as Billing Frauds, Spyware, Hostile Downloaders, Privilege Escalators, Ransomware, Rooting apps.

At a personal level, mobile devices offer a lucrative target to developers with malicious intent, due to their usage in e-commerce as a payment device at a point of sale (e.g., Google Wallet, Apple Pay). Also because they serve as a potential second factor authentication device and contain highly personal data like emails, messages and sensor data. At a corporate user level, the rise in adoption of Bring-Your-Own-Device (BYOD) principle, into organizational policy, allow employees' personal devices to be used both within and outside of corporate firewalls. Such devices at times store confidential industrial data thus creating an incentive for developers with malicious intent.

The BYOD scenario motivates a key challenge in access control for mobile devices. Depending on user context, access rights can change. For instance, it might be permissible to send some data over the corporate VPN, but not have it uploaded to Facebook. It might be OK to use camera generally, but not inside the company facility. Reporting GPS locations to a platform provider (e.g., Google, Apple) might be fine in general, but not when inside a sensitive compartmented information facility (SCIF). The current "permit once" model followed by most mobile OSs are inadequate for handling such context-dependent access control tasks.

In light of such potential problems, we submit that there is a need for fine-grained, dynamic and context-driven access control policies to protect the privacy and security of a user and her data. Although significant work in access control has been carried out by the research community [7], [8], [9], the state-of-the-art has stopped at generalizing access control policies to grouped policy profiles. Consequently, we present **MITHRIL**[1] access control system. Three key contributions of MITHRIL includes policy representation, user-preferred and specific policy capture and policy enforcement. We use a working prototype built for Android, that allows us to capture a policy *P'* that denotes a user-preferred and specific, context-sensitive policy, starting from a policy *P* that denotes an initial generic policy, applicable to a user's profile category.

We use Semantic Web technologies like the Semantic Web Rule Language [10] (SWRL), to represent our access control policy rules. We use the Platys ontology [11] written using the Web Ontology Language (OWL) [12], to model user context. MITHRIL combines information about users' context, requested information and requester info as antecedents in policy rules that allows us to express complex rule conditions. Our policy makes a closed world assumption for access

[1] MITHRIL is a reference to a precious, lightweight and extremely strong silvery metal from the Lord of the Rings which protected its wearer, Frodo, from life threatening dangers: http://lotr.wikia.com/wiki/Mithril

control, in order to reduce the complexity of the system. As in, when the context, requested information and requester combination does not match an appropriate policy rule we fall back to a deny by default access control mechanism.

For MITHRIL we have a user policy control module that uses a *Violation Metric* (hereafter denoted as VM metric), to determine state of the policy capture system. Using the metric allows us to determine, if we are closer to a stable and personalized user access control policy or not. The VM metric also allows us to determine transitional state for MITHRIL. We use an iterative model for capturing users' preferred policy that is guided by a hierarchical context ontology for location, activity generalizations. We begin the process by using an initial policy and observe all violations of said policy that happens on the mobile device. We use a periodic user feedback process that allows us to evolve from an initial user policy to a specific user policy. Finally, we use a policy enforcement module that intercepts any data request made by an app and manages access control decisions.

The main contribution of our work is the design and development of the MITHRIL system, which has the following features:

- An access-control policy representation technique using an ontology to model high-level semantic context on a mobile device.
- A framework for policy capture and using our VM metric to determine transitional state for MITHRIL.
- Access control decision handling and policy enforcement.

The rest of the paper is organized as follows. We start with a discussion of the related work in Section II. Following that we present our system's overview, define relevant terminology and present our assumptions in Section III. We present our use case scenarios in Section IV. Section V provides the details of our iterative process for capturing user-specific policy on a mobile device and discusses a possible way to generate more policies. We present our evaluation methodology and a discussion on our experimental results Section VI. Finally, we conclude the paper with a summary and discussion on possible future directions for the work in Section VII.

## II. RELATED WORK

Access control research has broadly focused on policy representation and policy execution [13], [14], [9] with work done both by the open source and research communities. Some prominent examples of open source solutions include XPrivacy (which needs a rooted phone), Privacy Guard (available on CyanogenMod, a custom Android ROM) and the PDroid application (which requires a rooted device). However, none of these systems can take into account user context for their policies. As explained earlier, this is a critical requirement for a mobile user today and our system uses such context driven policies. Access control examples from research community include work done by Conti et. al. [7] (CRePE), Enck et al. [8] (TaintDroid) and Jagtap et al. [11] (Preserving Privacy in Context-Aware Systems). CRePE was one of the earliest systems that described security policy enforcement based on the context of a mobile phone. Our system extends the approach taken in CRePE by introducing a more expressive context model and using inference to compare a user's current context to a context description in a policy.

TaintDroid was another research effort where data flow on an Android device was studied to determine when sensitive data left the system via an untrusted application. TaintDroid studied some contextual feature for determining privacy breaches but that included context of an app itself, i.e. "if it is accessing location, is this app a location provider?". The user context is a significant driver for privacy needs and was ignored by TaintDroid. Such a solution was proposed by our group in past work done by Jagtap et al. and Ghosh et. al. [11], [15]. The focus of these works was constraining data flow based on user context. They used predefined policies, which could be obtained from system admins or industry experts. However, there is also a need to allow individual users to modify, extend and create policy rules to match their needs and preferences. MITHRIL helps users do that by identifying potential faults in current policy and capturing policy modifications.

The state-of-the-art in research on policy capture stops at determining generalized privacy profiles [16], [17], [18], [19]. These works conclude that it was possible to create privacy "profiles" applicable to user categories on mobile devices with reasonable accuracy. When it comes to defining their own rules, it was observed by [17] that users were not good judges of how well a rule meets their true needs or preferences. However, in their other work they showed that with enough "privacy nudges", explaining how their location was being shared, users could be guided into modifying their preference. We argue that given a set of policy violations and a hierarchical context model, users would be able to define their preferred policy. We focus on using context generalization and specialization with assistance from our Platys ontology [11] driven context model, and combining that with user feedback to reach an individual user's preferred specific policy.

## III. SYSTEM OVERVIEW

The system architecture of MITHRIL is shown in Figure 1. MITHRIL contains four main components: a policy enforcement module, a policy decision module, a policy store module and a user policy control module. MITHRIL sits between application layer and framework layer. It takes as input a request for data or component access. It's output contains requested data or access to a component or an exception stating that data or component is unavailable. To represent access constraints, MITHRIL uses an attribute-based access control (ABAC) model [20], where the attributes represent user context, requested resource and requester meta-data. We use ABAC as it provides us the flexibility of having any number of attributes to be added to our rule.

MITHRIL has two operating modes: OBSERVER and ENFORCER. In observer mode the system does not enforce access control policies, but simply notes all violations of current policy rules. In this mode, feedback requested from user

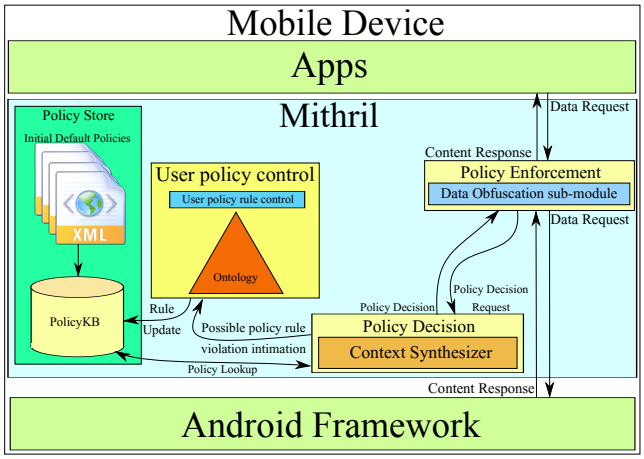


Fig. 1. Mithril sits between apps and Android framework and comprises four main modules: policy enforcement, policy decision, policy store and policy control

periodically, on the recorded violations, in order to capture their 'preferred policy'. The frequency for feedback is a system setting that is adjustable by user or system admin. After an initial round of policy capture and user interaction MITHRIL moves to ENFORCER mode, in which it enforces current applicable policy rules. The transition between the two modes is determined using a predefined, but adjustable, threshold for the VM metric.

### A. Relevant Term Definitions

Following are some terms and definitions that we use in this paper.

*Definition 1:* CONTEXT has been defined by Dey and Abowd [21] as: *"[...] any information that can be used to characterize the situation of an entity (i.e., identity, location, activity, time). An entity is a person, place, or object that is considered relevant to the interaction between a user and application, including the user and applications themselves."*

We have used above-mentioned definition of context and extended it further in our previous work [15] to generalize or specialize location and activity context.

*Definition 2:* A POLICY, consists of a set of RULES (also referred to as POLICY RULES in this paper), that define access control for data. A policy is applicable for a USER-CATEGORY or specific user.

*Definition 3:* A RULE, also called policy rule in our system, is a Semantic Web Rule Language (SWRL) [10] rule represented as `antecedent` $\Rightarrow$ `consequent`.

MITHRIL uses context-sensitive privacy policy rules defined in the Semantic Web Rule Language (SWRL) [10] to handle access control on mobile devices. Sharma et.al. [22] created a showed how to represent security rules using OWL and the ABAC model. SWRL was a proposal for a rule extension for OWL (W3C member submission) [10] and therefore we use it for our access control policy representation. The abstract syntax for SWRL rules follow the Extended Backus-Naur Form (EBNF) notation which, while are useful for XML and RDF serializations. Antecedent(s) must hold for a consequent to apply. Multiple antecedents in a rule are defined as a conjunctions of atoms. The consequent atom states whether the access is allowed or denied.

*Definition 4:* A policy rule VIOLATION, is recorded when antecedent(s) of a rule and a consequent defines a certain behavior for an app and an access pattern that contradicts this behavior is observed.

Assuming our rule is "Do not share **camera** resource with **social media** applications at **work**". A violation is recorded if for some reason the camera is accessed by a social media app at work. During the feedback cycle this violation will be marked as a TRUE violation (hereafter denoted as *TV*), if the user agrees that this behavior is indeed unexpected. For example, if the camera was accessed by Instagram, a social media app, when at work and the user did not expect this observed behavior, then we have a true violation captured. A violation is considered to be a FALSE violation (hereafter denotes as *FV*), if the user expected observed behavior. For example, the camera might have been accessed at work by Instagram, but the user initiated it while at lunch in the cafeteria. Using the *TV* and *FV* frequencies we compute our Violation Metric (VM). Our VM metric, computed as follows, helps us determine if MITHRIL is ready to transition from OBSERVER mode to ENFORCER mode:

$$VM = \frac{TV}{FV + TV}$$

This VM metric computes the "Precision" of our policy capture system, as in the ratio of true positives and sum of true and false positives. Our true violations are the "true positives", which signifies that the default policy *P* and the user's preferred policy *P'* were the same and *NO* modifications to the original policy will be required. On the other hand false violations or "false positives" are situations when the default policy *P* and the user's preferred policy *P'* differ and we need to capture change in current policy. A high value of the VM metric signifies we are closer to a user's "personalized" policy.

### B. Assumptions

Throughout this paper, we use a running example to explain the working behavior of our system. The example is simply used for clarity purposes and our working prototype is not limited to this example. Our example uses a policy applicable to users in a "graduate student" category. We also assume that our users work for confidential research and therefore protecting their personal and professional data is of critical importance. We assume that they start from an initial policy *P*, that they are allowed to modify to better protect their own data. We use Android mobile devices for our system implementation as Android is open source and thus we can modify the system behavior with respect to policy implementation or policy violation detection. A system grounded in an OWL ontology and that uses an OWL-DL reasoner has some limitations. OWL-DL, for example, doesn't allow us to draw conclusions based on our not knowing some fact. This means that once

$$
\begin{aligned}
&resourceRequested\,(?r, Camera) \wedge \\
&requestingApp\,(?app) \wedge \\
&hasAppType\,(?app, SocialMedia) \wedge \\
&User\,(?u) \wedge \\
&userLocation\,(?u, ?l) \wedge \\
&hasLocationType\,(?l, UniversityLab) \\
&\rightarrow \\
&AccessLevel\,(Deny)
\end{aligned}
$$

Fig. 2. Simple rule for controlling social media camera access

we have deleted a rule as presented in the next section, we will not be able to infer the decision for said rule. However, our system uses a closed world assumption and defaults to a default deny policy when a policy decision cannot be made about a certain request. One final assumption for our prototype was that we do not focus on context extraction from sensors, rather our high-level semantic context is already available to the system.

### C. Policy Store

The policy store module in MITHRIL has a knowledge base containing the currently applicable policy for the user-category of the mobile device's user. The user chooses an applicable user-category, when the system starts for the first time and the initial policy for said user category is then downloaded on the mobile device. Research conducted by [16], [17], [18], [19] have established that it is possible to fairly accurately create privacy profiles applicable to user categories on mobile devices. We use such a predefined initial policy, driven by user category classification defined in our ontology, as a starting point for our system.

The storage module takes as input a requester app's information and information about the requested resource and searches the policy knowledge base for the applicable policy rules and returns the same to the policy decision module. The second task that the policy storage handles is updating a policy rule as requested by the user policy control module. Now, let us take a look at how rules are represented in MITHRIL.

*Rule Representation:* Rules, in our system, are expressed using SWRL. A more abstract representation may be considered as a triple (U, C, Q) which contains: U, that represents user's context, that is the user whose data is represented in the system. C is requester metadata, that is the app which is requesting component access or user data. Q represents the data request in form of a query. The consequent of a rule defines the action to be taken. We define some use cases in detail in the following section but for now we present an example rule where, we have an app that belongs to the social media category. Let us take a look at a rule from our policy called GRADSTUDENTPOLICY for graduate students, called SOCIALMEDIACAMERAACCESSRULE. The rule states that, while the student is in a university building, social media apps are not allowed to access camera on her mobile device. The rule is shown Figure 2.

$$
\begin{aligned}
&resourceRequested\,(?r, Camera) \wedge \\
&requestingApp\,(?app) \wedge \\
&hasAppType\,(?app, SocialMedia) \wedge \\
&User\,(?u) \wedge \\
&userTime\,(?u, ?t) \wedge \\
&timeAfter\,(?t, 0900) \wedge \\
&timeBefore\,(?t, 1700) \wedge \\
&userDayOfWeek\,(?u, ?d) \wedge \\
&hasDayType\,(?d, weekday) \wedge \\
&userActivity\,(?a) \wedge \\
&hasActivityType\,(?a, Advisor\_Meeting) \wedge \\
&userpresenceInfo\,(?p) \wedge \\
&hasPresenceType\,(?p, Advisor) \wedge \\
&userLocation\,(?u, ?l) \wedge \\
&hasLocationType\,(?l, UniversityLab) \\
&\rightarrow \\
&AccessLevel\,(Deny)
\end{aligned}
$$

Fig. 3. Rule with higher granularity, for controlling social media camera access

Example of a higher granularity rule can be seen in Figure 3, which has more conditions incorporated. In plain terms we are now stating that instead of just being applicable in a university building, we "Do not allow camera access to "Social Media" apps when the time of day is between 9AM and 5PM and it is a weekday and the user is at university lab location in presence of her Advisor and has a meeting scheduled with her Advisor".

### D. Policy Decision

The policy decision module receives as input, a request meta-data from policy enforcement module. The current context is obtained using a context synthesizer sub-module. The context synthesizer keeps user context facts updated using an OWL-DL reasoner and a context ontology to infer high-level and semantically rich context. A similar technique for context inference from low level sensor information was explored in [23]. We use the Platys ontology [11] to semantically represent user context. We use classes defined in the Platys ontology to define hierarchical context models that enables us to generalize or specialize over user context. An example of how this is used is shown in section V.

We use a knowledge-base on the phone that stores facts about apps including app categories. The facts are extracted from various sources like the Android Marketplace [24] and the DBpedia ontology [25]. The facts include meta-data like app manufacturer, download count, maturity rating, user rating, developer country of origin, number and category of permissions requested by the app etc. The facts about user context and apps are stored in form of RDF triples, which helps us query the knowledge-base for properties like app types or location types. These information enables the inference mechanism as the rules are stated in terms of the properties of apps and user context.

The final piece of information needed to make a decision are the rules for the current request meta-data, which are provided by the policy storage module. A requester, resource tuple can have multiple policy rules applicable based on contextual conditions. Once rules are obtained, using context and app facts from knowledge-base a specific rule applicable is inferred by an OWL-DL reasoner. The consequent of a chosen rule is the applicable action. If action is deny then a data request is marked as a possible violation of current policy rules.

In observer mode, the violation meta-data, which consists of a request meta-data along-with an applicable rule and user context is forwarded to User Policy Control module and no response is sent to policy enforcement module. In enforcer mode however, action inferred by reasoner is simply returned to policy enforcement module to manage access to requested resource.

### *E. Policy Enforcement*

For policy enforcement, MITHRIL has to be the system admin. We achieve the goal of inserting ourselves in between applications and Android framework and acting as an admin by using a custom ROM. Our solution is similar to some seen in the open source world like the XPrivacy [26] solution which is an XPosed [27] module. Android has a process called Zygote which is similar to the Linux init process. Just like in Linux every application starts as fork of Zygote. This process is started by an /init.rc script when the phone is booted. Zygote is started with /system/bin/app_process, which loads all required classes and invokes initialization methods. Upon installation, XPosed copies an extended app_process executable to /system/bin. This adds an additional jar to the classpath and calls methods and inside that method, XPosed can act in Zygote's context. An XPosed module can thus act as the initiating process for every application and thus is able to control its behavior. Our access control implementation for MITHRIL emulates the behavior of XPosed and is installed on a phone with a custom ROM CM13 [28] which is a fork of Android 6.0.1 (Marshmallow). Our app acts with root privileges, which is required for controlling other apps.

The policy enforcement module receives as input, data requests from apps and serves them with data as dictated by the "action" returned by policy decision module. In observer mode, policy enforcement module does enforce access control on the mobile device. In this mode it simply passes data request tuples consisting of a requested component name or type of data and a requester name (henceforth referred to as: request meta-data) to policy decision module. In enforcer mode, it passes on a request meta-data but expects policy decision module to provide an "action". If the action is to allow access, it simply makes a request to the Android framework for the data and returns the same to the requesting app. If action is to deny access, it prohibits request from going any further.

### *F. User Policy Control*

Finally, we take a look at the user policy control module. This module is of key importance in this paper and will be discussed in detail in Section V but we provide a brief overview here. As we have explained before, MITHRIL starts with an initial policy for a particular user category as defined by the occupation chosen by the user at installation time for MITHRIL. We collect user's identity and some basic profile information. This information includes user's identity, work location, home location, occupation category defined by our ontology etc. Using the policy control module we capture a user's preferred policy. We use an ontology to define contextual information using a hierarchical context model. We use Location and Activity generalization as was shown in our group's previous work [29] and discussed in Section V.

## IV. USE CASE SCENARIOS

The use cases that we will discuss represent the possible scenarios we envision in our policy violation and user feedback process. We used CM13, a fork of Android 6.0.1 (Marshmallow) for creating the application that allows us to capture user feedback. Before Android Marshmallow, we had a permission model of install-time permission acquisition for data access allowed to an app. In Marshmallow we saw the launch of run-time permission acquisition model. Point to note here is that we now have Android 7.0.1 (Nougat) out in the wild but from an access control model perspective, nothing has changed, so using CM13 is okay for now. However, we still do not have context-sensitive, fine-grained and dynamic access control in Android. In our running example, we have an initial policy for graduate students, i.e., GRADSTUDENTPOLICY that contains a few rules like the following:

- SOCIALMEDIACAMERAACCESSRULE: Do not share camera resource with social media applications at work
- SOCIALMEDIALOCATIONACCESSRULE: Do not share location with social media applications at work
- TOOLAPPSNETWORKACCESSRULE: Do not share network information with Tool apps
- PRODCUTIVTYAPPSIDENTITYACCESSRULE: Do not share identity with productivity apps

In our example scenario we will use the SOCIALMEDIACAMERAACCESSRULE for explanations. We are assuming that the user is a graduate student at UMBC's Computer Science department. In the OBSERVER mode, we mentioned earlier, MITHRIL captures violations of the current applicable policy. Now imagine that the user takes a picture at the university cafeteria during lunch hours and uploads to Instagram. Our system is able to use the Platys ontology to determine that university cafeteria is "part_of" the university and therefore the user is at 'Work'. Using our app knowledge-base we are also able to determine that Instagram is a Social Media app. Since we have a rule that states that social media applications are not allowed location access at 'Work', we detect this as a violation of applicable policy. In the next user feedback cycle we present all such "violations" of the initial policy to the

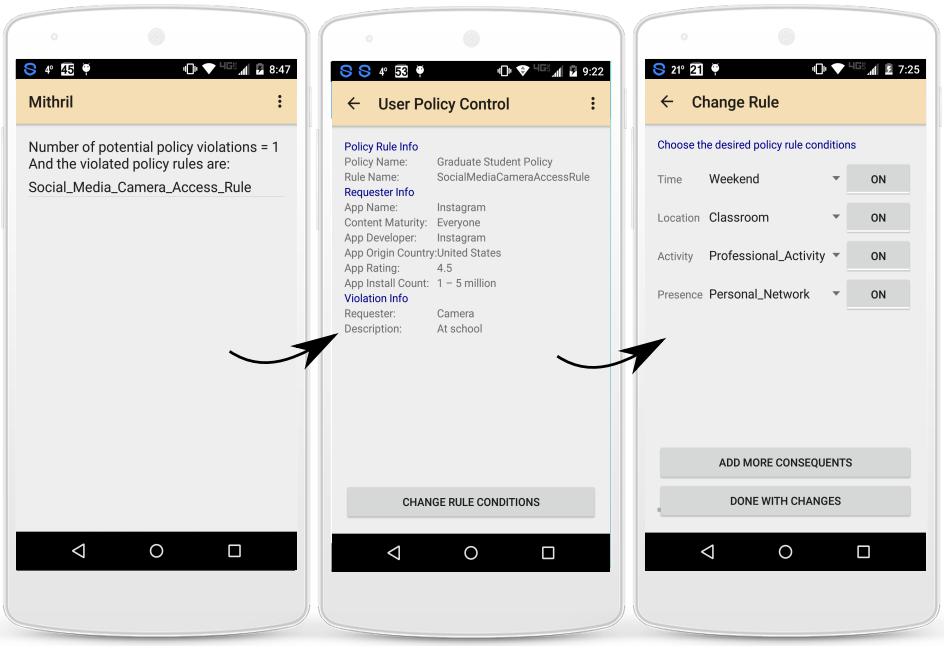


Fig. 4. Transitions shown for prototype app

user. At this juncture we study the five use case scenarios that can happen in our system.

### A. Use case - True Violations:

"Rule is good, keep it". User is presented with a violation and the user determines that this was a **TRUE** violation. As stated before, this type of violation signifies that user did not expect observed behavior and the policy requires no change. In this case, the response we capture is used as a confirmation of the rule as being true. We will not ask the user about this rule again unless some sort of system wide reset happens. In this scenario we will enforce this rule as-is in ENFORCER mode. This use case signifies MITHRIL will now make "Do not share camera resource with social media applications at work" a quasi-permanent policy. By quasi-permanent we mean that the policy is not going to change unless there is an explicit system reset performed to go back to the initial policy applicable to the user category. This could happen if some static user profile information is changed that was collected at the install time.

### B. Use case - False Violations:

The rest of the use cases state situations when policy modifications are required but we still have some variations in how modifications are carried out. The rest of the use cases explain these situations.

USE CASE FV-1: "Rule is not required, delete it". User is presented with a violation and the user determines that this was a **FALSE** violation requiring deletion. This scenario indicates that the user expected this behavior and thus the policy rule that causes current observation to be determined as a violation is no longer applicable. Similar to the above use case we will not ask the user about this rule again unless some sort of system wide reset happens to the original default policy. In this scenario we will delete this particular rule and not enforce it in ENFORCER mode.

USE CASE FV-2: "Rule requires antecedent generalization, modify it". User is presented with a violation and the user determines the policy rule to be **FALSE** violation but an imprecise rule. That is the observed behavior although might not have been unexpected but the current rule does not clearly define the user's preferred policy. As a result, observed behavior cannot be clearly stated as a violation of user's policy. An example of such a scenario would be, our rule stated was "Do not share camera resource with social media applications at work". Observed behavior was Instagram, a social media app, was used at University Cafeteria. The cafeteria is inferred as a work location as it is part of the University. However, the user expects to use their mobile to take pictures during lunch. Therefore, the rule requires more conditions like a temporal restriction or a more precise location restriction or an activity restriction. Such restrictions would mean modification would be required for the rule antecedents or new antecedents would have to be added to the rule. As such, at this point we can have four different outcomes. Since the user determines the rule as imprecise, they are allowed to modify the rule. Modifications could include changing a specific contextual antecedent into a more generic contextual antecedent. See the app design diagram in Figure 7. Our example rule stated that "Do not share camera resource with social media applications at work". The 'at work' part of the rule for a graduate student profile is used to infer that a location related antecedent applies and University Campus is a 'work' location. Now, user may choose to make location antecedent into a more generic antecedent by going up the relationship chain defined in our ontology. An

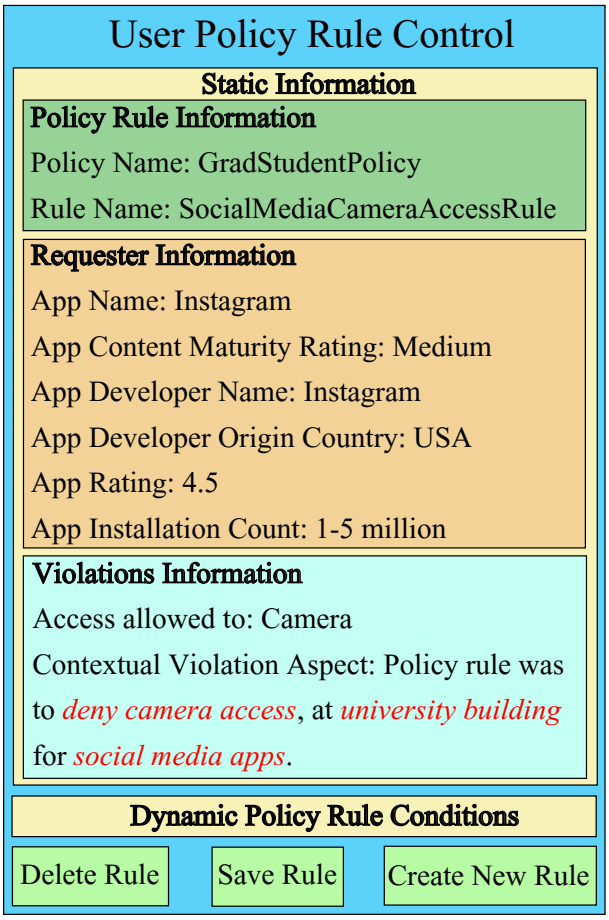


Fig. 5. Rule violation meta-data displayed to user

example of such generalization would be that user wants to apply the camera restriction at city level. New generic rule now applies as "Do not share camera resource with social media applications in Baltimore county".

USE CASE FV-3: "Rule requires antecedent specialization, modify it". User could also choose to make the rule more specific. For example they could state that the rule applies only at the 'University Lab'. The reasoner will be able to infer that a modified rule needs to be enforced at a more specific location than previously captured. In pretty much the same way as the above use case, user is allowed to choose a more specific location antecedent by parsing down the relationship chain in our ontology. The modified policy thus becomes "Do not share camera resource with social media applications at University Lab".

USE CASE FV-4: "Rule has too many or is missing conditions, delete or add them". The most interesting use case is that of adding or deleting antecedents to the rule. As we saw in Figure 3. We could have a situation where the rule only applies if it is official work hours or in presence of certain other people. In such a scenario, our system allows the user to add or even remove contextual and other antecedents to the rule. Thus allowing us to capture more fine-grained policies than previously possible. Our example policy for social media camera resource access thus becomes "Do not share camera resource with social media applications at University Lab between 9AM and 5PM on a weekday in presence of Advisor". Such a rule can be captured by user feedback process only, thus justifying the need for our system.

USE CASE FV-5: "New rule is required". An extension of the above use case would be that user needs a new rule altogether. This option is also available to user through our system. The user may simply choose to start an empty rule and add new antecedents and state a consequent that captures some aspect of the user's policy that was not covered by the initial policy. This flexibility allows our system to be capable of defining policies with all possible combinations of our system's known antecedents. As a result, with proper feedback we will always be able to reach the user's preferred policy.

## V. SYSTEM IMPLEMENTATION

Since MITHRIL uses a feedback mechanism to iteratively modify policy rules, we need to take a look at the rule capture interface and process. We have implemented a prototype system that has four modules. The first module detects app launch and API call behavior. This is to determine when an app, for example, requests location update. The second module gathers contextual information. The third module intercepts the calls made by an app and either returns dummy responses or no response at all. Since Android Marshmallow no data return is an acceptable behavior and we take advantage of this feature. Thus achieving data privacy and security with low system instability.

The user policy control is the final part of our system implementation. Figure 5 shows the violation meta-data as seen by the user and Figure 7 shows the policy modification options that a user is provided during the process of capturing their preferred policy. We have also added a set of screenshots from our prototype app showing the steps of rule capture (see Figure 4). A feedback iteration starts with a list of violations, obtained from policy decision module, being presented to a user. When the user chooses to look at a specific rule violation from a list they are presented with a specific rule's violation meta-data, which includes actual rule statement and a list of facts about an app that is violating a rule. User then has the option of further exploring the violation by clicking on "Display Policy Rule Conditions" button for exploring context antecedents for said rule.

The frequency of feedback is a admin or user setting. During each feedback iteration, the user is shown a list of all potential violations on their mobile device. As explained in previous section, user has two options at this point. They can choose to state a violation as a true violation or as a false violation. Our ontology and user context facts allows us to generalize or specialize over user's context. This provides a convenient way for user to modify policy conditions, in order to define changes in the current rules. We use two types of generalization: by location and by activity.

Location generalization in our ontology is achieved by using the transitive properties "is$^{'}$ and "part_of". We define in our

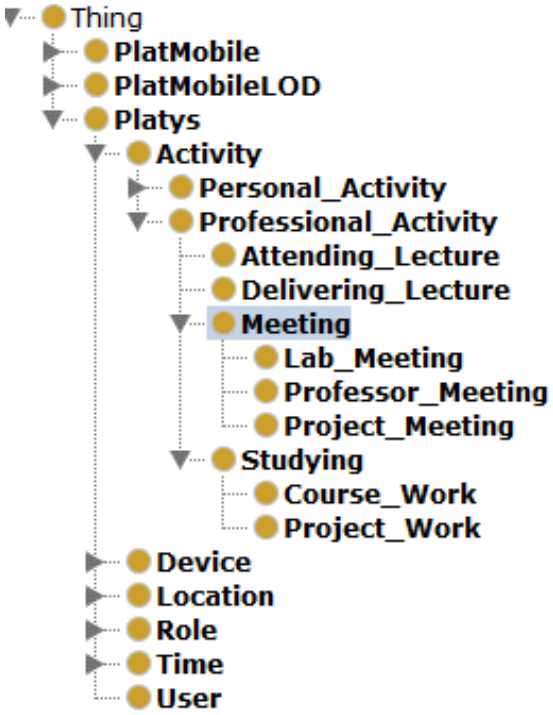


Fig. 6. Snapshot of Platys Ontology defining context hierarchy

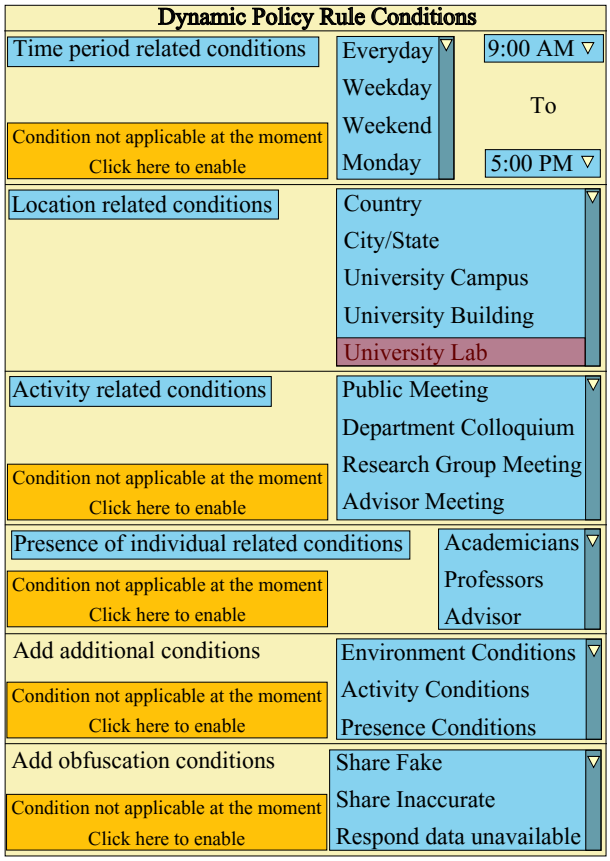


Fig. 7. Ontology-driven hierarchical options for rule modification

ontology every place in the world as a sub class of the Location class. Thus we have sub classes Country, State, City, Organization, Building, Room, Point. Using these sub classes and the part_of property we define a hierarchical location model and represent them using some simple axioms like "Room is a part of Building". We use an OWL-DL reasoner to infer different relations existing between instances of these sub-classes.

Every activity in our ontology is the sub class of the Activity class. It is possible to obtain user activities, using Google APIs, which are related to a device's motion. For example if a user is walking, running or in a car or not. On the other hand we can obtain user activity information from User's calendar too. In our ontology we have classes defined like Professional_Activity, Meeting, Lab_Meeting, Professor_Meeting, Project_Meeting etc. We can see this class hierarchy from the ontology in Figure 6. This allows us to define a hierarchical activity generalization model via sub class relationships between generic and specific activities. Imagine a scenario where an app is collecting microphone data and we want to protect private lab information. We can then define policies for Lab_Meeting or we can define activity context based policies for any "Meeting", if we want to prevent recordings at all of our meetings.

A sample view of hierarchical choices can be seen in Figure 7. Although we have discussed six use case scenarios that might occur during a policy capture process based on violation information presented to users, it is possible to have more use cases which might be beyond even the violation capture process. One such scenarios would be when a policy rule's consequent is modified. This could either negate our initial rule or may add conditions on what data could be shared. Such a condition might include data obfuscation techniques. In this case user will have to add antecedents that define those limitations. For example, the user might want to share a fake Location, an inaccurate location or state that location is unavailable.

Clearly, our policy rules are significantly more complicated as opposed to a simple permission based model that Android currently follows. The dynamic nature allowed by the variable actions and the granularity provided by the contextual antecedents are contributing factors to this complexity. However, it also gives more control to the user over her data.

*Towards automatic rule generation:* The use of a hierarchical context model via an ontology allows us to infer subsumption relationship between a generic and a specific rule. As a result, we can use an ontology to infer decisions for contextual situations, for which no "specific" rules exist. For example, if there is a rule that states "Do not allow access to camera at work". That means any location that can be determined to be a work location can be assumed to be a place where camera access is not allowed. However, once the user modifies this rule to a more specific rule stating "Do not allow access to camera in university Building" what

can we assume about the locations that are still work and were part of the rule but are not anymore? Can we generate more policies that state "Do not allow camera access in university parking lot"? We can discard the rule that states access denied at cafeteria as we observed the user's response to that specific violation but what about the other conditions? Given that our ontology defines the state of the world, we can possibly generate all such conditional rules using our hierarchical context model. In a similar way, if we observe, as per our contextual model, rules being added for every sub class or context piece at a particular hierarchy then we can infer a more generic antecedent for the specific context piece and reduce the rule set to a smaller set. To the extent of what context we can capture in this hierarchical manner, we may generate rules for a user automatically. Given our system design and an initial policy for a user category, we can carry out the reduction and expansion of policies into a bigger and smaller set of rules. However, such reduction or expansion is beyond the scope of the current paper.

## VI. Experimental results

We used a LG Nexus 5 device with CM13 for our experiments. For this study we wanted to show the usability of the Violation Metric (VM) as a way to determine if user's preferred policy had been captured or not.

### A. Experimental setup:

We setup the experiment with about a 100 different policies curated by hand using various combinations of contextual situations that we envisioned. Some of our sample location context included: Home, Work, Lab, Department_Office, Classroom, Meeting_Room, Supervisor_Office. Sample activity context included: Sleeping, Dining, Traveling, Personal_Activity, Professional_Activity, Meting, Lab_Meeting, Studying, Project_Work. We also used presence info and temporal context of working and non-working hours, in our rules. As stated in our assumptions in Section III-B, the above semantic context is available to the prototype system. For simplicity of experimental setup, we used NFC tags that were programmed with contextual situations to simulate changes in context. After a context change was observed, we use an automated script to start various apps on the mobile device that would violate our default policy *P*. The examples of such violations has been discussed Section IV. All automated user behavior on the mobile device was created using monkeyrunner API from Android [30].

### B. Observations and discussions

After each feedback iteration, we recorded the value of the VM metric. Take a look at the graph in Figure 8. Our simulation includes experiments that VM metrics over eight feedback iterations from our simulated user. It shows that the variation in VM metric when a user provides consistent feedback over a number of iteration cycles. You can see when the feedback is consistent, the VM metric steadily increases towards a high value. The predefined threshold we use, may be

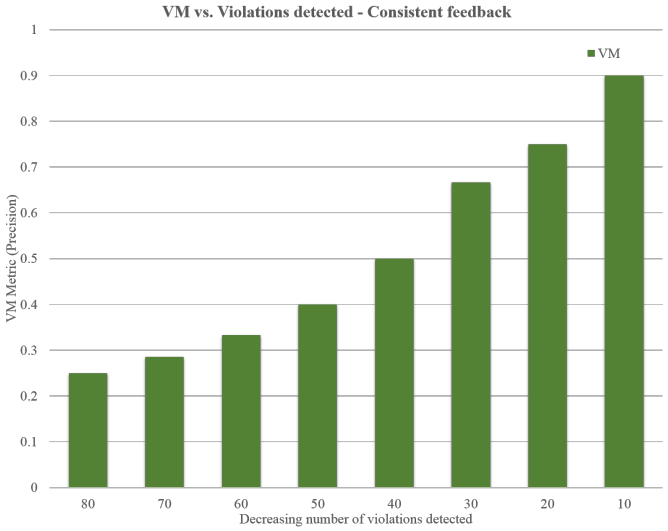


Fig. 8. Consistent feedback in policy rule changes by user

varied by a system admin. However, when MITHRIL reaches a state with high precision as determined by the VM metric, we are able to conclude that policy capture process is complete, and we may transition from OBSERVER mode to ENFORCER mode.

The VM metric initially shows a low value as a lot of policies are being modified. As iterations go by, we see a decline in the number of FALSE violations. Once a violation has been determined to be TRUE we will not require a feedback from user on that rule, anymore. As a result of that, in the graph, we see a constant decline in the number of violations and the VM metric increases in value, getting upto 0.9 by our last feedback iteration. As explained before, this means that whatever violations the system records are denoted as TRUE violations and we have captured the user's preferred policy.

Understandably, the VM metric does have certain limitations when it comes to user feedback being erratic. However, in a hostile user situation we believe that no system can perform well. However, in case of a consistent feedback, the system is capable of capturing the preferred user policy and also in determining the state of the system effectively.

## VII. Conclusions

Capturing users' privacy and security policies is not a new goal. Significant research work has been done in this field. However, all the research has brought us to a point where we are capable of generalizing access control to user policy profiles. In our current research, we are representing access control policies on mobile devices using Semantic Web technologies like SWRL and OWL. We are capturing user preferred policy rule modifications using a hierarchical context model and an iterative feedback process. We have designed and developed a prototype system to capture policy modifications required for user policy. The system has two working modes OBSERVER and ENFORCER. We only observe

system behavior in OBSERVER mode and we execute access control policies captured by our system in ENFORCER mode. We use a violation capture methodology and a violation metric to determine system state. Finally, we infer policy decisions using an OWL-DL reasoner and hierarchical context model defined in the Platys ontology. We use the inferred policy decisions to handle access control on a mobile device using a custom ROM and an app with root privileges.

Any system which defines methods of actual user study have to handle usability issue. Such issues could be mitigated by conducting real world studies. However, we only used a simulated user study due to the fact that our goal was to show the effectiveness of the violation metric, in determining transitional state of a policy capture system. We argue that such effectiveness may be shown using simulated studies like we have carried out.

As part of future work, we would like to extend the experimental scenarios and study more variations in user feedback patterns. A complete end-to-end system implementation with real context generation from sensors and consequent release in Google Play Store [24] is another goal. Currently, we do not use any predictive or generative model for policy rule modification. An obvious future work would be in pattern recognition in user's feedback process and predicting feedback for rules in known antecedent feature space.

## VIII. ACKNOWLEDGMENT

Support for this work was provided by NSF grants 0910838 and 1228198.

## REFERENCES

[1] J. Callaham, "Google says there are now 1.4 billion active android devices worldwide," September 2015. [Online]. Available: http://www.androidcentral.com/google-says-there-are-now-14-billion-active-android-devices-worldwide

[2] T. Muller and A. Kirschner, "Apple celebrates one billion iphones," July 2016. [Online]. Available: http://www.apple.com/newsroom/2016/07/apple-celebrates-one-billion-iphones.html

[3] Statista, "Number of available applications in the google play store from december 2009 to july 2015," 2015. [Online]. Available: http://www.statista.com/statistics/266210/number-of-available-applications-in-the-google-play-store/

[4] R. Chang, "10 alternative android app stores," 2014. [Online]. Available: http://code.tutsplus.com/articles/10-alternative-android-app-stores--cms-20999

[5] I. Security, "Mcafee lbs: Threats report," November 2014. [Online]. Available: http://www.mcafee.com/us/resources/reports/rp-quarterly-threat-q3-2014.pdf

[6] G. A. Security, "The google android security teams classifications for potentially harmful applications," April 2016. [Online]. Available: https://static.googleusercontent.com/media/source.android.com/en//security/reports/Google_Android_Security_PHA_classifications.pdf

[7] M. Conti, V. T. N. Nguyen, and B. Crispo, "Crepe: Context-related policy enforcement for android," in *Information Security*, ser. Lecture Notes in Computer Science, M. Burmester, G. Tsudik, S. Magliveras, and I. Ilic, Eds. Springer Berlin Heidelberg, 2011, vol. 6531, pp. 331–345.

[8] W. Enck, P. Gilbert, B.-G. Chun, L. P., Cox, J. Jung, P. McDaniel, and A. N. Sheth, "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*, 2010, pp. 1–6.

[9] L. Kagal and H. Abelson, "Access control is an inadequate framework for privacy protection," in *In W3C Privacy Workshop*, 2010.

[10] 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*, vol. 21, p. 79, 2004.

[11] P. Jagtap, A. Joshi, T. Finin, and L. Zavala, "Preserving privacy in context-aware systems," in *Semantic Computing (ICSC), 2011 Fifth IEEE International Conference on*, Sept 2011, pp. 149–153.

[12] S. Bechhofer, F. Van Harmelen, J. Hendler, I. Horrocks, D. L. McGuinness, P. F. Patel-Schneider, L. A. Stein *et al.*, "Owl web ontology language reference," *W3C recommendation*, vol. 10, no. February, 2004.

[13] 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.

[14] L. Kagal and T. Berners-Lee, "Rein: Where policies meet rules in the semantic web," *Computer Science and Artificial Intelligence Laboratory, Massachusetts Institute of Technology, Cambridge, MA*, vol. 2139, 2005.

[15] D. Ghosh, A. Joshi, T. Finin, and P. Jagtap, "Privacy control in smart phones using semantically rich reasoning and context modeling," in *Security and Privacy Workshops (SPW), 2012 IEEE Symposium on*, 2012, pp. 82–85.

[16] M. Benisch, P. G. Kelley, N. Sadeh, and L. F. Cranor, "Capturing location-privacy preferences: Quantifying accuracy and user-burden tradeoffs," *Personal Ubiquitous Comput.*, vol. 15, no. 7, pp. 679–694, Oct. 2011. [Online]. Available: http://dx.doi.org/10.1007/s00779-010-0346-0

[17] N. Sadeh, J. Hong, L. Cranor, I. Fette, P. Kelley, M. Prabaker, and J. Rao, "Understanding and capturing people's privacy policies in a mobile social networking application," *Personal Ubiquitous Comput.*, vol. 13, no. 6, pp. 401–412, Aug. 2009. [Online]. Available: http://dx.doi.org/10.1007/s00779-008-0214-3

[18] J. Lin, B. Liu, N. Sadeh, and J. I. Hong, "Modeling users' mobile app privacy preferences: Restoring usability in a sea of permission settings," in *Symposium On Usable Privacy and Security (SOUPS 2014)*. Menlo Park, CA: USENIX Association, Jul. 2014, pp. 199–212. [Online]. Available: https://www.usenix.org/conference/soups2014/proceedings/presentation/lin

[19] B. Liu, J. Lin, and N. Sadeh, "Reconciling mobile app privacy and usability on smartphones: Could user privacy profiles help?" in *Proceedings of the 23rd International Conference on World Wide Web*, ser. WWW '14. New York, NY, USA: ACM, 2014, pp. 201–212. [Online]. Available: http://doi.acm.org/10.1145/2566486.2568035

[20] V. C. Hu, D. Ferraiolo, R. Kuhn, A. R. Friedman, A. J. Lang, M. M. Cogdell, A. Schnitzer, K. Sandlin, R. Miller, K. Scarfone *et al.*, "Guide to attribute based access control (abac) definition and considerations (draft)," *NIST Special Publication*, vol. 800, no. 162, 2013.

[21] A. K. Dey and G. D. Abowd, "Towards a better understanding of context and context-awareness," in *First Int. symposium on Handheld and Ubiquitous Computing (HUC)*, 1999.

[22] N. K. Sharma and A. Joshi, "Representing attribute based access control policies in owl," in *2016 IEEE Tenth International Conference on Semantic Computing (ICSC)*. IEEE, 2016, pp. 333–336.

[23] T. Gu, X. H. Wang, H. K. Pung, and D. Q. Zhang, "An ontology-based context model in intelligent environments," in *Communication Networks and Distributed Systems Modeling and Simulation Conf. (CNDS)*, 2004.

[24] Google, "Android apps on google play," January 2015. [Online]. Available: https://play.google.com/store/apps?hl=en

[25] P. N. Mendes, M. Jakob, and C. Bizer, "Dbpedia: A multilingual cross-domain knowledge base." in *LREC*, 2012, pp. 1813–1817.

[26] M. Bokhorst(M66B), "Xprivacy," June 2013. [Online]. Available: https://github.com/M66B/XPrivacy

[27] rovo89, "Xposed," August 2014. [Online]. Available: https://github.com/rovo89/XposedBridge/wiki/Development-tutorial

[28] Cyanogenmod, "Cyanogenmod: About," July 2015. [Online]. Available: https://wiki.cyanogenmod.org/w/About

[29] L. Zavala, R. Dharurkar, P. Jagtap, T. Finin, and A. Joshi, "Mobile, collaborative, context-aware systems," in *Proc. AAAI Workshop on Activity Context Representation: Techniques and Languages, AAAI. AAAI Press*, 2011.

[30] Google, "The monkeyrunner api," October 2016. [Online]. Available: https://developer.android.com/studio/test/monkeyrunner/index.html