ACCESSORS
A Data-Centric Permission Model for the Internet of Things

Christoph Stach and Bernhard Mitschang
Institute for Parallel and Distributed Systems, University of Stuttgart,
Universitätsstraße 38, D-70569 Stuttgart, Germany

Keywords: Permission Model, Data-Centric, Derivation Transparent, Fine-Grained, Context-Sensitive, IoT.

Abstract: The *Internet of Things* (*IoT*) is gaining more and more relevance. Due to innovative IoT devices equipped with novel sensors, new application domains come up continuously. These domains include *Smart Homes*, *Smart Health*, and *Smart Cars* among others. As the devices not only collect a lot of data about the user, but also share this information with each other, privacy is a key issue for IoT applications. However, traditional privacy systems cannot be applied to the IoT directly due to different requirements towards the underlying permission models. Therefore, we analyze existing permission models regarding their applicability in the IoT domain. Based on this analysis, we come up with a novel permission model, implement it in a privacy system, and assess its utility.

1 INTRODUCTION

The *Internet of Things* (*IoT*) is leaving the *Early Adopters* stage as it is becoming more and more popular (Davies et al., 2016). New *Things*, i. e., small gadgets with sensors which are connected to the Internet, come onto the market. Not only technology enthusiasts are interested in these devices, but they also get into the focus of the general public. IoT has opened the way for novel use cases in various domains including *Smart Homes*, *Smart Health*, or *Smart Cars* (Aman et al., 2017).

As the Things[1] are able to capture a lot of diverse data about the user and they can share this information with each other, IoT applications (or *apps*) gain a comprehensive knowledge about their users. That way, they are able to improve the quality of our everyday life, as they are able to predict the most likely user demands at an early stage and adapt their provided services accordingly. However, the possible threats deriving from such apps cannot be underestimated. Individuals can be monitored permanently without their knowledge. Therefore, IoT apps have an high impact on privacy (Aggarwal et al., 2013).

Technical approaches are required to conceal sensitive information and give users the ability to control what happens with their data. However, most privacy management systems overwhelm users. Generally, such systems enable to restrict the usage of a certain data processing unit. Yet, users cannot estimate which threats originate from these entities (Felt et al., 2012). From the data gathered for a data producer, a lot of additional information can be derived (Perera et al., 2014). For instance, a proximity sensor can disclose the absolute location of a user also, when it gathers the distance to a Thing with a stationary location. So, a data-centric and thus comprehensible approach is required to secure private data.

For this very reason, we provide the following five contributions in our work:

(1) We deduce requirements towards a permission model for IoT apps from a use-case scenario.

(2) We analyze permission models which are applied in existing privacy systems and provide a comprehensive overview of their features and their applicability in the IoT domain.

(3) We construct a d**a**ta-**c**entri**c** p**e**rmi**ss**ion m**o**del for the Inte**r**net of Thing**s**, called ACCESSORS.

(4) We apply ACCESSORS to an existing privacy system, the **P**rivacy **M**anagement **P**latform (*PMP*) (Stach and Mitschang, 2013, Stach and Mitschang, 2014). However, we could use any of the many similar privacy systems as a foundation for our model without a loss of argument.

(5) We evaluate our model and assess its utility.

[1] We use the generic term *Thing* for any kind of device equipped with sensors and Internet access.

Stach, C. and Mitschang, B.
ACCESSORS - A Data-Centric Permission Model for the Internet of Things.
DOI: 10.5220/0006572100300040
In *Proceedings of the 4th International Conference on Information Systems Security and Privacy (ICISSP 2018)*, pages 30-40
ISBN: 978-989-758-282-0

The remainder of this paper is as follows: Section 2 introduces a real-world use-case scenario to illustrate the challenges for a permission model for IoT apps. Then, Section 3 postulates five key requirements for such a permission model. Section 4 discusses various existing permission models. Our model—ACCESSORS—is introduced in Section 5. Section 6 describes how to apply ACCESSORS to a privacy system. Finally, Section 7 assesses our approach before Section 8 concludes this work and gives a short outlook on future work.

2 USE-CASE SCENARIO

(Istepanian et al., 2011) describe how IoT technologies can be applied for non-invasive glucose level sensing and diabetes management. To that end, the patients' diabetes devices are linked via IPv6 connectivity to their healthcare provider to forward any measured data. Yet, patients do not know, which data exactly is forwarded, especially since such a device is able to gather various kinds of data. For instance, some devices also add location data to each glucose metering, since this information can be relevant for the diagnostic analysis (Knöll, 2010, Stach and Schlindwein, 2012, Stach, 2016). By combining the gathered data, further knowledge can be derived (e. g., a combination of blood sugar values and location data enables to draw inferences about the user's eating behavior as a rising blood sugar level shortly after walking past a candy shop indicates that the user has bought some sweets (Knöll, 2009)).

The number of available IoT health devices is considerable growing. Each new generation comes up with more sensors, offering a wider service spectrum than the preceding generation (Vashist et al., 2014). The accuracy of the sensors is also getting better and better (Kovatchev et al., 2004). However, not every IoT app requires such a high accuracy. Meaning that from a privacy perspective, the data quality should be downgraded in order to conceal some private information. While some of the data provided by such IoT devices are uncritical from a privacy point of view or so vital that the data is required all the time, other sensitive data is required only in case of an emergency. For instance, if a patient suffers an insulin shock, any available data should be sent to his or her physician in order to get the best medical attendance as possible.

In such a scenario a privacy system—or more specifically its permission model—has to meet several novel requirements in order to be effective (Sicari et al., 2015). For instance, a focus on data-centric protection goals is becoming essential (Agrawal et al., 2012), especially since the processed information is assembled from various sources which are not necessarily known to the user (Takabi et al., 2010). Furthermore, as permanently new devices with new kinds of sensors are rolled out, the permission model has to persist in such a fluctuating setting (Aggarwal et al., 2013).

3 REQUIREMENTS SPECIFICATION

As the foregoing scenario illustrates, there are special requirements towards a permission model for IoT apps which are detailed in the following.

[R1] Data-Centric Policy Rules. In order to be comprehensible and manageable for users, permissions have to refer to types of data (e. g., blood sugar level) instead of data providers (e. g., glucometer). While it is obviously that a glucometer meters the blood sugar level, some devices provide additionally location data. If policy rules are based on data providers, a user might allow a health app to use his or her glucometer without knowing that s/he also gave access to such non-medical data in the process. Moreover, the same type of data can be provided by several providers (e. g., the blood sugar level is provided by glucometers as well as *Apple Watches*). If a user wants to prohibit access to his or her health data, then this rule has to be applied to any possible provider.

[R2] Derivation Transparency. An IoT app has access to several sensors and thus various types of data. By combining these, additional data can be derived. A permission model has to be able to represent such coherences. For instance, if A can be derived from B and C and a user prohibits access to A, then an app must not be allowed to access B and C at the same time. This can be archived by describing what data can be derived from which sources. Then the user can assign permissions at data level and the privacy system has to apply corresponding rules to the associated sources.

[R3] Extendable Permission Model. The IoT is constantly evolving as new sensor technologies or communication standards arise. A static permission model, i. e., a model with a fixed set of protected entities, gets outdated very fast. Therefore, the model has to be dynamically extendable. In particular, all extensions have to be backward compatible, that is, by extending the model previous policy rules must not become invalid.

[R4] Fine-Grained Policy Rules. In order to enable a user to manage his or her data in a privacy-aware manner, s/he requires total control over information

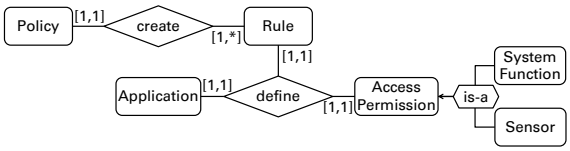


Figure 1: The Permission Model Applied in Android.

distribution and disclosure. That implies that the permission model has to support fine-grained policy rules in terms of two dimensions: On the one hand, the protected entities have to be fine-grained. For instance, Android provides a `Bluetooth` permission which controls any data that is provided by a device via a Bluetooth connection—such a permission specifies neither a type of data nor a sensor. As a consequence, users have to permit apps to use a Bluetooth headphone and a Bluetooth medical device at the same time via a single permission! On the other hand, a user has to have several choices how to constrain a certain permission. Most permission models allow Boolean decisions, only (grant or deny). However, a permission for location data also could restrict the accuracy of the data.

[R5] Context-Sensitive Policy Rules. Since IoT apps are often context-aware, that is, an app react on the situation it is currently used, also the policy rules should be context-sensitive. For instance, a medical app should have access to any kind of data in case of an emergency. Otherwise, more restrictive policy rules should be applied. We follow the context definition of (Dey, 2001), i. e., "any information that can be used to characterize the situation of an entity".

4 RELATED WORK

Based on these requirements, we analyze permission models which are currently used in the IoT context. As the IoT is a highly heterogeneous environment with various operating systems running on the Things, a lot of different privacy systems and thus permission models are being used which aggravates the privacy issues. Yet, there are several approaches to establish Android in the IoT, e. g., *Android Things* (Google Inc., 2017) or *RTAndroid* (Kalkov et al., 2012). So, we focus on Android-based privacy systems in our work. However, as any mobile platform applies a comparable permission-based privacy system, we loose no argument by doing so (Barrera et al., 2010).

In Android a quite simple permission model is applied (see Figure 1). Each Permission regulates either the usage of a system functions (e. g., adding entries to the calendar) or access to a sensor (e. g., the camera). An app which uses such a function or sensor has to request the appropriate permission. For

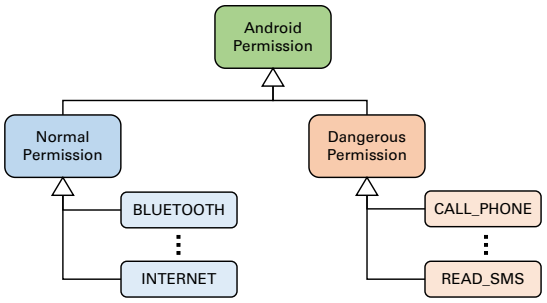


Figure 2: Classification of Android Permissions in Normal and Dangerous Ones, cf. (Wei et al., 2012).

each granted request a policy rule is created. Most permissions are automatically granted when an app is installed, while *dangerous* permissions have to be granted at runtime (Enck et al., 2009).

Whenever Google adds new permissions or relabels existing permissions, app developers have to add these permissions to their already released apps in order to keep them operative (Sellwood and Crampton, 2013). Yet, a single permission can cover several system functions or sensors, which makes it hard for a user to comprehend the permissions. This is aggravated by the fact that there are too many different permissions, even for noncritical operations (e. g., the usage of the vibration function) (Felt et al., 2012). That is why Google no longer informs the user about noncritical permissions, the so-called *normal* permission. However, Google classifies access to the Internet or the usage of Bluetooth device as safe operations (see Figure 2), although both can have a severe impact on the user's privacy. Thus, such a basic permission model is not applicable for the IoT.

(Sekar et al., 2012) introduce *Selective Permissions*. That is, any Android permission requested by an app is stored in a *Shadow Manifest* which can be modified at runtime. In this way, a user is able to withdraw certain permissions similar to the Android runtime permissions. Yet, the Selective Permissions have two advantages. On the one hand, a user can withdraw any permission and on the other hand a missing permission does not lead to a security exception. Instead a null value is returned to the app. However, this approach does not alter the Android permission model and therefore it does not fulfill any of the requirements postulated in Section 3 likewise.

CRêPE introduces a context-sensitive permission model (Conti et al., 2011). Each access to a data source, i. e., each permission request, can be linked to a spatio-temporal context. This context defines a condition under which the permission is granted. Yet, the rules are mapped to Android permissions and therefore, CRêPE has the same shortcomings as Android's permission model.

Apex introduces an XML-based policy language in order to restrict the usage of Android permissions (Nauman et al., 2010). The user is able to define, e. g., how often a certain permission can be used or in which chronological order permissions can be granted. Aside from such constraints, the permission model allows no wide-ranging contextual constraints or fine-grained permission settings. Also, the model is neither data-centric nor derivation transparent and it cannot be extended as it is based on Android permissions.

In *YAASE* a user defines what operations a certain app is allowed to perform on a resource, i. e., either a content or service provider (Russello et al., 2011). Data from these resources can be tagged, e. g., to differentiate between public and private data provided by the same resource. The user specifies whether only certain tags are accessible for an app. Moreover, s/he can define operations which have to be performed before the data is forwarded to an app, e. g., a filer operation to remove sensitive data. That way, YAASE is able to specify very fine-grained policy rules. However, these rules are not data-centric, derivation transparent, or context-sensitive. Furthermore, the expandability of the model is restricted to specified operations.

Sorbet mainly focuses on undesired information flows between apps (Fragkaki et al., 2012). To this end, the underlying permission model allows to specify information-flow constraints in order to prevent privilege escalation. That way, Sorbet supports some kind of context-sensitivity. In addition, the resources protected by the permissions can be any kind of component (e. g., services or content providers). So, the permission model is extendable. Moreover, the model allows to specify constraints that limit the usage of certain permissions, e. g., by adding a lifespan to it. This results in fine-grained policy rules even though Sorbet still deals with Boolean decisions. Also, Sorbet neither supports data-centric nor derivation transparent policy rules.

RetroSkeleton is an app rewriting system capable of replacing method calls with arbitrary code fragments (Davis et al., 2012, Davis and Chen, 2013). Its policy consists of Clojure replacement commands. The user is able to postulate derivation transparent, fine-grained, and context-sensitive policy rules—provided that s/he has the required programming skills. Since the model itself is generic and does not map to predefined permissions, it is extendable. Yet, as it only replaces method calls, the model is not tailored to data-centric permissions.

Constroid, which is based on the $UCON_{ABC}$ model (Park and Sandhu, 2004), provides subjects (e. g., processes) with rights to operate on data items (e. g., all business contacts) (Schreckling et al., 2012). The data items can be associated with attributes (e. g., contacts without a private phone number) to restrict the access rights. Since the access rights are related to data only, the considered operations are restricted to create, read, update, and delete. Optional conditions specify whether a rule is applicable under a certain context. Yet, the model is not extendable and does not support derivation transparent policy rules.

SPoX is a security policy specification language (Hamlen and Jones, 2008). SPoX rules define a state machine that accepts any sequences of commands that comply with the security policy. (Backes et al., 2013) apply this language in their privacy system called *AppGuard*. So, the user is able to formulate fine-grained policy rules, e. g., by limiting network access to a certain address. As any command can be restricted by policy rules, AppGuard's permission model is extendable. That way, even novel data sources are supported by AppGuard out of the box. By chaining several rules, the user is able to model some kind of derivation transparency. Also the context in which a certain command is executed can be restricted (Backes et al., 2014). However, these restrictions only refer to the command sequence and not to the user's context. Moreover, the privacy model is not data-centric.

(Scoccia et al., 2017) introduce flexible permissions for Android which are called *AFP*. In AFP the permissions are bound to features of an app. That is, an app is only allowed to request a permission in order to perform a certain task. Moreover, AFP enables users to assign fine-grained permissions, e. g., by granting only access to selected contacts instead of the whole contact list. Since AFP defines its own permissions, the model is extendable. Yet, the policy rules are neither data-centric nor derivation transparent.

DroidForce introduces data-centric policy rules (Rasthofer et al., 2014). *OSL* (Hilty et al., 2007) is used to specify the rules. In this way, temporal conditions as well as cardinality and time constraints can be added to each permission. Therefore both, fine-grained and context-sensitive policy rules are supported. The key feature of DroidForce is its focus on data-centric permissions. That is, the permissions are mapped to data domains (e. g., location data or contacts data) rather than system sensor functions. However, coherences between protected data sources cannot be modeled and the applied model is not extendable.

The *Privacy Management Platform* (*PMP*) (Stach and Mitschang, 2014) applies the *Privacy Policy Model* (*PPM*) (Stach and Mitschang, 2013) (see Figure 3). The PPM is extendable and enables fine-grained and context-sensitive policy rules. It splits apps into self-contained *Service Features*. In this way, permissions are not granted to apps, but directly to the respective

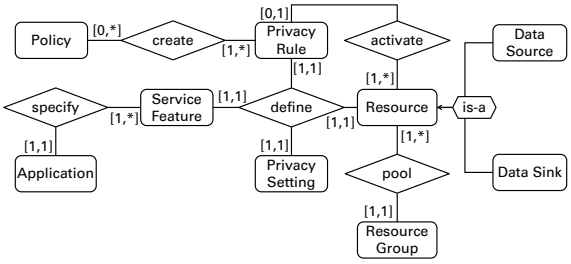


Figure 3: The Privacy Policy Model (PPM).

Service Features. The permissions get an earmarking for a certain purpose and the user can (de-)activate these features according to his or her needs. Access to data sources or sinks is provided via so-called *Resources*. Related Resources can be pooled in a *Resource Group* (e. g., GPS and WiFi positioning are part of a location Resource Group). For each Resource, a user can define individual *Privacy Settings* (e. g., to reduce the accuracy of location data for a certain Service Feature). The data from the Resources is also used to add contextual constraints to each privacy rule to define its scope of application. So, the PPM already meets most of the requirements towards a permission model for the IoT. Yet, it does not support data-centric policy rules and therefore overstrains the user in such a heterogeneous environment. This flaw gets obvious in the following *Smart Health* example:

If a user keeps an electronic health diary via a *Smart Health* app for his or her Smartphone. However, s/he only wants to track his or her fitness progress including heart rate (pulse meter), activities (accelerometer and orientation sensor), and training locations (GPS). Additionally, s/he wants to use the camera for a visual documentation of his or her training progress. For instance, a insurance company provide a special tariff rate for users of this app which rewards a healthy lifestyle.

Hence, a user might not want to share any data that could indicate an illness such a the body temperature with the app as this could lead to a higher insurance rate. In the PPM, the user is therefore able to prohibit this app to access a Bluetooth medical thermometer for the purpose of measuring the body temperature. The thermometer then is represented as a Resource and the measuring is represented as a Service Feature.

However, if we assume that the Smartphone is equipped with a thermographic camera, the user has to be aware of this feature. If s/he does not consider that, such a camera is also able to reveal his or her body temperature. To prevent this s/he has to define an additional Privacy Rule in the PPM for the measuring Service Feature which prohibits the access of the camera Resource. For any data source s/he has to reflect which knowledge can be derived from its data.

In theory, the PPM is indeed able to secure sensitive data also for IoT apps. However, with an increasing number of different sensors, it is almost impossible for a user to keep track of all the possible data leaks due to the Resource-centric Policy Rules of the PPM. Nevertheless, the PPM is a sound foundation for ACCESSORS which introduces a data-centric perspective.

5 THE ACCESSORS MODEL

As shown in the previous section, the PPM is well-suited for Smartphone apps with a manageable amount of sensors involved. In an IoT scenario with lots of different sensors, another approach is required since humans tend to think in a data-centric way. That is, a user knows which data s/he wants to keep secret and s/he does not want to bother about which sensor or data source could reveal this type of data. For instance, if a user wants to prohibit that an app has access to his or her body temperature, this should be represented via a single rule instead of one rule per data source which holds this information. To this end, a permission model has to be able to map data producers to the type of data which is provided by them. In this way, a user can select which type of data should be available to an app (e. g., body temperature) and the model unfolds which data sources have to be considered (e. g., medical thermometer and thermographic camera). Our approach of a d**a**ta-**c**entri**c** p**e**rmis**s**ion m**o**del for the Inte**r**net of Thing**s**—or short ACCESSORS—is shown in Figure 4. In the following, we detail its key components and elaborate on how they contribute to meet the five requirements.

Rule Core. Similar to the PPM, an ACCESSORS policy rule basically consists of three parts: an *access purpose*, a permission to *access* a data processing unit[2], and a *constraint*. These triples compose the *rule core*. Optionally, each policy rule can be linked to a *context* under which it is activated (see Paragraph Context Abstraction).

User Abstraction. Any *inquiring entity*—i. e., an *app*, a *Smart Thing*, or even a *user*—is able to specify one or multiple access purposes that require access to a protected type of data. An access purpose is a code fragment within an app that carries out a single task. For instance, such an access purpose in the use-case scenario described in Section 2 could be the graphical representation of metering locations on a map. That

[2]A data processing unit is either a *data producer* or a *data consumer* (see Paragraph Data Abstraction).

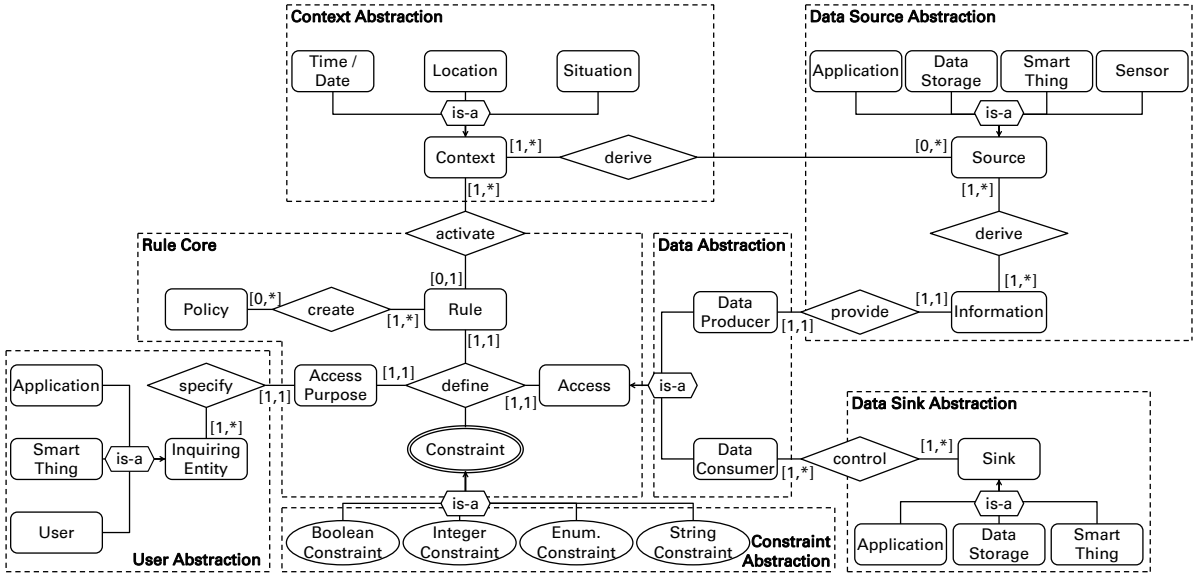


Figure 4: ACCESSORS — The **D**ata-**C**entri**c** **P**ermis**s**ion M**o**del for the Inte**r**net of Thing**s**.

way, permissions are not linked to an app but to a certain access purpose. As a consequence, a user can decide, which access purposes are justified for a certain type of app and whether s/he wants to execute the associated code fragment when it requires the specified access permissions. Analogous to the PPM, non-essential app features can be skipped to reduce the amount of required private data. The *user abstraction* assures that further types of *Smart Things* can be added on demand.

Data Abstraction. The *data abstraction* facilitates to link permissions to both, *data producers* as well as *data consumers*. However, the focus for both units is on the type of data that is produced or consumed. That is, an inquiring entity has to specify which data it needs access to such as location data or health data instead of a concrete data processing unit such as GPS or a glucometer.

Data Sink Abstraction. Each data consumer is associated with several *data sinks* such as *apps* or *services*, *data storages*, or even other *Smart Things*. In other words, the user can specify policy rules how data can be preprocessed for an app. For instance, s/he could allow an app to use a service which stores health data for long-term monitoring of a particular health condition.

Data Source Abstraction. Each data producer is associated with a certain type of *information*. An information is any aspect that can be derived from raw data. That is, it can be the raw data itself (e. g., a single blood sugar value) or any kind of higher order data gained by combining several sources (e. g., a health record containing data from various metering devices). For each type of data diverse *data sources* can be specified in the ACCESSORS model. A data source does not have to be a *sensor* necessarily, but also *apps*, *data storages*, and *Smart Things* are qualified as data sources. In this way, complex coherences can be modeled (e. g., the information "activity" can be derived either by a combination of data from an accelerometer and a position sensor or directly by a readout from a fitness tracker). Due to the *data sink abstraction* and *data source abstraction*, the policy rules remain completely detached from a concrete technology. The rules are automatically adapted to the available data sources and sinks.

Context Abstraction. Optionally, an activation context can be attached to each policy rule. This context describes under which conditions a rule has to be enforced by a privacy system. The context is described in accordance with (Dey, 2001) as a *spatio-temporal* condition (e. g., a rule should only be applied during working hours) or as a higher order *situation* (e. g., a rule is only valid in case of a medical emergency). Higher order situations can be modeled as a combination of values from data producers.

Constraint Abstraction. For each policy rule various constraints can be defined. The most basic one is a Boolean constraint to grant or deny to process a certain type of data[3]. Depending on the type of data, ACCESSORS supports three additional constraint types: integer constraints, enumeration constraint, and string constraints. Integer constraints can be used to define

[3]If the access permission is denied, the particular code fragment is skipped in the app.

an upper or lower boundary. For instance, a maximum accuracy for a certain type of data such as location data can be specified in this manner. An enumeration constraint defines several valid setting options. For instance, for health records there could be settings granting access only to domain-specific record sections such as pulmonary data or cardiac data. Finally, string constraints enable users to enter textual conditions. For instance, a user can declare a MAC address of a Thing with which s/he wants to share his or her health data. Thereby it is ensured that health data is sent to the specified destination address, only.

All in all, ACCESSORS supports data-centric policy rules as the focus of the permission is on types of data instead of concrete data processing units. Since data producers provide higher order information which can be composed of data from several sources, ACCESSORS is able to model relations between various types of data. As the policy rules interrelate access purposes, access permissions, constraints, and contexts, only, they are independent from concrete inquiring entities or data producers and data consumers, respectively. This means in effect that ACCESSORS has two types of extensibility due to its abstraction layers. On the one hand it can be broadened (e. g., by adding new Things) and on the other hand it can be deepened (e. g., by adding new coherences between data sources if new methods to derive a certain kind of information from raw data are discovered). Policy rules modeled with ACCESSORS are very fine-grained. On the one hand, the multi-valued constraints allow a highly precise vernier adjustment of the permission rights. On the other hand, since the permissions are bound to a certain access purpose and need not be granted to an app in general, the user is able to tailor the privacy policy entirely towards his or her needs. Lastly, each policy rule can be enriched by an activation context. This context is highly generic, since it can be composed of any data currently available.

When comparing the PPM (see Figure 3) with ACCESSORS (see Figure 4), it is obvious that the two models have several common components. The rule core of ACCESSORS is almost equivalent to the PPM. However, ACCESSORS has two significant advantages: a) ACCESSORS introduces abstraction layers for users, data, data sinks, data sources, contexts, and constraints. b) ACCESSORS considers a different protection goal. While the PPM is designed for Smartphones and therefore considers only apps as potential attackers and sensors or system functions as feasible targets (labeled as Resources), ACCESSORS is entirely designed for the IoT. On that account, not only the potential attackers are interpreted in a wider sense—any kind of inquiring entity is considered as an attacker, including apps, *Smart Things*, and users—but also the targets which are protected by ACCESSORS have a different focus. The targets are geared to types of data instead of data sinks or sources.

Nevertheless, it seems natural to map policy rules defined in ACCESSORS to PPM rules, due to the similarity of the two models, as the PPM is already applied to existing privacy systems such as the PMP. In this way, we can exploit this infrastructure in order to enforce ACCESSORS policy rules. We describe a mapping of ACCESSORS policy rules to PPM rules in the following section.

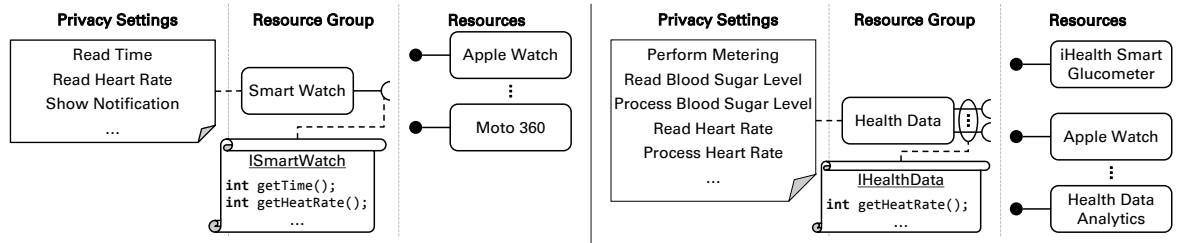
6 APPLICATION OF ACCESSORS IN THE PMP

From a modeling point of view, the fine-grained structure of ACCESSORS with its highly branched abstraction layers is necessary in order to gain a high expressiveness of the policy rules. However, from a implementation point of view, the number of utilized components should be kept low in order to reduce complexity. On that account, a mapping of the detailed ACCESSORS policy rules to similar PPM rules, is also recommended.

To that end, it is necessary to convert the access purposes specified by inquiring entities in ACCESSORS to Service Features. However, a Service Feature also defines certain permissions which are required in order to execute a particular code fragment. The focus on a broader range of possible attackers in ACCESSORS is not contradictory to the Service Features and a one-to-one mapping is possible without any further ado.

That is why the transition of data-centric targets modeled in ACCESSORS into PPM Resources poses the biggest problem for the mapping. In particular this implies that all Resources have to be replaced by new data-centric components. Nevertheless, ACCESSORS can be applied to the PMP, due to its modular architecture. In the PMP, each Resource Group is implemented as an independent functional unit which can be installed individually. Moreover, additional Resources can be added to a Resource Group at any point of time (Stach, 2013). Therefore, existing Resources can be replaced by new data-centric ones in order to apply ACCESSORS to the PMP.

Figure 5a depicts the model of a Resource Group for Smart Watches. The Resource Group defines a common interface for all of its Resources. An arbitrary Resource, which provides the required functionality, can be plugged into the the Resource Group at runtime. That is, the Resources are concrete implementation artifacts of the interface for a given hardware (e. g., an



(a) Model of a PMP Resource Group

(b) Application of ACCESSORS in the PMP

Figure 5: Mapping of ACCESSORS Rules to PMP Resource Groups.

Apple Watch or a Moto 360). The Resource Group also defines feasible Privacy Settings, i. e., how the user is able to restrict access to a particular Resource.

Figure 5b illustrates how this model has to be adapted in order to make the PMP compatible to ACCESSORS privacy rules. In the first instance, the hardware- or service-based focus of the Resource Groups has to be shifted to a data-centric one. The Resource Group given in the example deals with any kind of health-related data. This data can be provided by legit health devices such as a glucometer or by novel Things such as a Smart Watch. Moreover, this Resource Group also deals with data consumers of health data such as analytics libraries. In order to be able to plug in all of these data processing units, the Resource Group's interface has to be broadened accordingly.

The PPM Resource Groups provide only a single plug for one Resource at a time. To support derivation transparency, i. e., to be able to model data which is assembled from various sources, the ACCESSORS Resource Groups need multiple plugs. For instance, it is possible to deduce the blood sugar level considerably accurate by monitoring the activities of a user and his or her eating behavior (Zeevi et al., 2015). So, the Resource Group for health data has to be able to plug in a Resource capturing physical activities and a Resource gathering nutrition data, simultaneously. Furthermore, each Resource can be associated with multiple Resource Groups. For instance, a Smart Watch providing both, location and health data belongs to a location Resource Group as well as a health data Resource Group. The Privacy Settings for the new data-centric Resource Groups are carried over from the PPM's Resources that are pooled in the respective Resource Group.

That way, ACCESSORS can be mapped to the PPM. PPM rules are executable on the PMP and similar privacy systems. As the PMP runs on Android and Android is becoming increasingly pertinent to the IoT, such an implementation constitutes a serviceable privacy system for the IoT.

Table 1: Comparison of Current Permission Models.

Approach	R1	R2	R3	R4	R5
Android	✗	✗	(✓)	✗	✗
Selective Permissions	✗	✗	✗	✗	✗
CRêPE	✗	✗	✗	✗	✓
Apex	✗	✗	✗	(✓)	(✓)
YAASE	✗	✗	(✓)	✓	✗
Sorbet	✗	✗	✓	(✓)	(✓)
Retro-Skeleton	✗	(✓)	✓	(✓)	(✓)
Constroid	✓	✗	✗	(✓)	✓
AppGuard	✗	(✓)	✓	✓	(✓)
AFP	✗	✗	✓	✓	✓
DroidForce	✓	✗	✗	✓	✓
PMP	✗	✓	✓	✓	✓
ACCESSORS	✓	✓	✓	✓	✓

7 DISCUSSION

ACCESSORS is outright **data-centric** since all of its protected entities (data producers as well as data consumers) are related to a certain type of data (e. g., health data). Apps request access to such data without having to specify which sensor or system function provides this data. That way, ACCESSORS enables also **derivation transparency**. Each protected data object can originate from various sources. Additionally, multiple sources can be combined in order to derive a certain

type of data (e. g., the activity of a user can be derived from an accelerometer in combination with a position sensor). ACCESSORS enables to model a single data source as the producer of highly diverse data, e. g., an Apple Watch can produce location data as well as health data. The ACCESSORS model is **extendable**. On the one hand, additional data sources and sinks can be added at runtime in order to react on forthcoming hardware. For instance, the Apple Watch is able to meter the blood sugar level after a glucometer upgrade. On the other hand, our model supports various types of entities. An inquiring entity can be mapped to an app, a *Smart Thing*, or a user. That way, ACCESSORS is not committed to a fixed entity type. In the IoT context where new Things arise frequently, such an extensibility is essential. Moreover, ACCESSORS is **fine-grained** and **context-sensitive**. That is, multi-valued constraints as well as spatio-temporal or situation-based conditions can be added to each policy rule.

Table 1 summarizes the key characteristics of the permission models analyzed in the work at hand (see Section 4). Moreover, it provides a comparison of ACCESSORS with these approaches. The examined characteristics correspond to the postulated requirements in Section 3, namely **[R1]** data-centric policy rules, **[R2]** derivation transparency, **[R3]** extendable permission model, **[R4]** fine-grained policy rules, and **[R5]** context-sensitive policy rules. Due to the comprehensive abstraction approach covering users, data, data sinks, data sources, contexts, and constraints ACCESSORS is able to meet all requirements towards a permission model for the IoT.

8 CONCLUSION

The IoT is becoming more and more popular. Interconnected devices with novel sensors come onto the market. As a consequence, innovative apps from highly diverse domains including *Smart Homes* and *Smart Health* are developed. Such apps gather a lot of data about their users and derive further information from this data. While these apps are able to improve the quality of our everyday life, they also pose a threat to user privacy. Therefore, technical approaches are required to give users full control over their data.

For this reason, we postulate a requirements specification towards permission models for the IoT. Based on these requirements, we provide a comprehensive overview and assessment of currently existing permission models. Since none of these permission models is appropriate for IoT apps, we introduce a d**a**ta-**c**entri**c** p**e**rmi**ss**ion m**o**del for the Inte**r**net of Thing**s** called ACCESSORS. We describe how privacy demands towards IoT apps can be modeled in ACCESSORS. For the implementation of ACCESSORS, we provide a mapping of the ACCESSORS policy rules to an existing permission model, the **P**rivacy **P**olicy **M**odel (*PPM*). We chose the PPM since it already fulfills a lot of requirements towards a permission model for the IoT. In this way, ACCESSORS policy rules can be applied in an actual privacy system, such as the **P**rivacy **M**anagement **P**latform (*PMP*). An assessment demonstrates the utility of our approach.

Since many IoT apps fall back on online services for data processing (e. g., (Steimle et al., 2017)), future work has to investigate, how ACCESSORS-based policy rules can be applied to a privacy mechanism for stream processing systems. Therefore, this research is carried on in the *PATRON* research project[4]. The focus of *PATRON* is on complex event processing in distributed systems and how ACCESSORS-like policy rules can be applied in such a system (Stach et al., 2017).

ACKNOWLEDGEMENTS

This paper is part of the PATRON research project which is commissioned by the Baden-Wrttemberg Stiftung gGmbH. The authors would like to thank the BW-Stiftung for the funding of this research.

REFERENCES

Aggarwal, C. C., Ashish, N., and Sheth, A. (2013). *The Internet of Things: A Survey from the Data-Centric Perspective*, chapter 12, pages 383–428. Springer.

Agrawal, D., El Abbadi, A., and Wang, S. (2012). Secure and Privacy-preserving Data Services in the Cloud: A Data Centric View. *Proceedings of the VLDB Endowment*, 5(12):2028–2029.

Aman, M. N., Chua, K. C., and Sikdar, B. (2017). Secure Data Provenance for the Internet of Things. In *Proceedings of the 3rd ACM International Workshop on IoT Privacy, Trust, and Security*, IoTPTS '17, pages 11–14.

Backes, M., Gerling, S., Hammer, C., Maffei, M., and Styp-Rekowsky, P. (2014). *AppGuard – Fine-Grained Policy Enforcement for Untrusted Android Applications*, pages 213–231. Springer.

Backes, M., Gerling, S., Hammer, C., Maffei, M., and von Styp-Rekowsky, P. (2013). AppGuard: Enforcing User Requirements on Android Apps. In *Proceedings of the 19th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, TACAS '13, pages 543–548.

[4]see http://patronresearch.de/

Barrera, D., Kayacik, H. G., van Oorschot, P. C., and Somayaji, A. (2010). A Methodology for Empirical Analysis of Permission-based Security Models and Its Application to Android. In *Proceedings of the 17th ACM Conference on Computer and Communications Security*, CCS '10, pages 73–84.

Conti, M., Nguyen, V. T. N., and Crispo, B. (2011). *CRêPE: Context-related Policy Enforcement for Android*, pages 331–345. Springer.

Davies, N., Taft, N., Satyanarayanan, M., Clinch, S., and Amos, B. (2016). Privacy Mediators: Helping IoT Cross the Chasm. In *Proceedings of the 17th International Workshop on Mobile Computing Systems and Applications*, HotMobile '16, pages 39–44.

Davis, B. and Chen, H. (2013). RetroSkeleton: Retrofitting Android Apps. In *Proceeding of the 11th Annual International Conference on Mobile Systems, Applications, and Services*, MobiSys '13, pages 181–192.

Davis, B., Sanders, B., Khodaverdian, A., and Chen, H. (2012). I-ARM-Droid: A Rewriting Framework for In-App Reference Monitors for Android Applications. In *Proceedings of the 2012 IEEE Conference on Mobile Security Technologies*, MoST '12, pages 28:1–28:9.

Dey, A. K. (2001). Understanding and Using Context. *Personal and Ubiquitous Computing*, 5(1):4–7.

Enck, W., Ongtang, M., and McDaniel, P. (2009). Understanding Android Security. *IEEE Security and Privacy*, 7(1):50–57.

Felt, A. P., Ha, E., Egelman, S., Haney, A., Chin, E., and Wagner, D. (2012). Android Permissions: User Attention, Comprehension, and Behavior. In *Proceedings of the Eighth Symposium on Usable Privacy and Security*, SOUPS '12, pages 3:1–3:14.

Fragkaki, E., Bauer, L., Jia, L., and Swasey, D. (2012). *Modeling and Enhancing Android's Permission System*, pages 1–18. Springer.

Google Inc. (2017). Android Things. https://developer.android.com/things.

Hamlen, K. W. and Jones, M. (2008). Aspect-oriented Inlined Reference Monitors. In *Proceedings of the Third ACM SIGPLAN Workshop on Programming Languages and Analysis for Security*, PLAS '08, pages 11–20.

Hilty, M., Pretschner, A., Basin, D., Schaefer, C., and Walter, T. (2007). *A Policy Language for Distributed Usage Control*, pages 531–546. Springer.

Istepanian, R. S. H., Hu, S., Philip, N., and Sungoor, A. (2011). The Potential of Internet of m-health Things "m-IoT" for Non-Invasive Glucose Level Sensing. In *Proceedings of the 2011 Annual International Conference of the IEEE Engineering in Medicine and Biology Society*, EMBS '11, pages 5264–5266.

Kalkov, I., Franke, D., Schommer, J. F., and Kowalewski, S. (2012). A Real-time Extension to the Android Platform. In *Proceedings of the 10th International Workshop on Java Technologies for Real-time and Embedded Systems*, JTRES '12, pages 105–114.

Knöll, M. (2009). *Diabetes City: How Urban Game Design Strategies Can Help Diabetics*, pages 200–204. Springer.

Knöll, M. (2010). "On the Top of High Towers . . . " Discussing Locations in a Mobile Health Game for Diabetics. In *Proceedings of the 2010 IADIS International Conference Game and Entertainment Technologies*, MCCSIS '10, pages 61–68.

Kovatchev, B. P., Gonder-Frederick, L. A., Cox, D. J., and Clarke, W. L. (2004). Evaluating the Accuracy of Continuous Glucose-Monitoring Sensors. *Diabetes Care*, 27(8):1922–1928.

Nauman, M., Khan, S., and Zhang, X. (2010). Apex: Extending Android Permission Model and Enforcement with User-defined Runtime Constraints. In *Proceedings of the 5th ACM Symposium on Information, Computer and Communications Security*, ASIACCS '10, pages 328–332.

Park, J. and Sandhu, R. (2004). The UCON_{ABC} Usage Control Model. *ACM Transactions on Information and System Security*, 7(1):128–174.

Perera, C., Zaslavsky, A., and Christen, P. (2014). Context Aware Computing for The Internet of Things: A Survey. *IEEE Communications Surveys & Tutorials*, 16(1):414–454.

Rasthofer, S., Arzt, S., Lovat, E., and Bodden, E. (2014). DroidForce: Enforcing Complex, Data-centric, System-wide Policies in Android. In *Proceedings of the 2014 Ninth International Conference on Availability, Reliability and Security*, ARES '14, pages 40–49.

Russello, G., Crispo, B., Fernandes, E., and Zhauniarovich, Y. (2011). YAASE: Yet Another Android Security Extension. In *Proceeding of the 2011 IEEE Third International Conference on Privacy, Security, Risk and Trust and 2011 IEEE Third International Conference on Social Computing*, PASSAT '11, pages 1033–1040.

Schreckling, D., Posegga, J., and Hausknecht, D. (2012). Constroid: Data-centric Access Control for Android. In *Proceedings of the 27th Annual ACM Symposium on Applied Computing*, SAC '12, pages 1478–1485.

Scoccia, G. L., Malavolta, I., Autili, M., Di Salle, A., and Inverardi, P. (2017). User-centric Android Flexible Permissions. In *Proceedings of the 2017 IEEE/ACM 39th International Conference on Software Engineering Companion*, ICSE-C '17, pages 365–367.

Sekar, L. P., Gankidi, V. R., and Subramanian, S. (2012). Avoidance of Security Breach Through Selective Permissions in Android Operating System. *ACM SIGSOFT Software Engineering Notes*, 5(37):1–9.

Sellwood, J. and Crampton, J. (2013). Sleeping Android: The Danger of Dormant Permissions. In *Proceedings of the Third ACM Workshop on Security and Privacy in Smartphones & Mobile Devices*, SPSM '13, pages 55–66.

Sicari, S., Rizzardi, A., Grieco, L. A., and Coen-Porisini, A. (2015). Security, privacy and trust in Internet of Things: The road ahead. *Computer Networks*, 76:146–164.

Stach, C. (2013). How to Assure Privacy on Android Phones and Devices? In *Proceedings of the 2013 IEEE 14th International Conference on Mobile Data Management*, MDM '13, pages 350–352.

Stach, C. (2016). Secure Candy Castle — A Prototype for Privacy-Aware mHealth Apps. In *Proceedings of the*

2016 IEEE 17th International Conference on Mobile Data Management, MDM '16, pages 361–364.

Stach, C., Dürr, F., Mindermann, K., Palanisamy, S. M., Tariq, M. A., Mitschang, B., and Wagner, S. (2017). PATRON — Datenschutz in Datenstromverarbeitungssystemen. In *Informatik 2017: Digitale Kulturen, Tagungsband der 47. Jahrestagung der Gesellschaft für Informatik e.V. (GI), 25.9-29.9.2017, Chemnitz*, volume 275 of *LNI*, pages 1085–1096. *(in German)*.

Stach, C. and Mitschang, B. (2013). Privacy Management for Mobile Platforms – A Review of Concepts and Approaches. In *Proceedings of the 2013 IEEE 14th International Conference on Mobile Data Management*, MDM '13, pages 305–313.

Stach, C. and Mitschang, B. (2014). Design and Implementation of the Privacy Management Platform. In *Proceedings of the 2014 IEEE 15th International Conference on Mobile Data Management*, MDM '14, pages 69–72.

Stach, C. and Schlindwein, L. F. M. (2012). Candy Castle — A Prototype for Pervasive Health Games. In *Proceedings of the 2012 IEEE International Conference on Pervasive Computing and Communications Workshops*, PerCom '12, pages 501–503.

Steimle, F., Wieland, M., Mitschang, B., Wagner, S., and Leymann, F. (2017). Extended Provisioning, Security and Analysis Techniques for the ECHO Health Data Management System. *Computing*, 99(2):183–201.

Takabi, H., Joshi, J. B. D., and Ahn, G. J. (2010). Security and Privacy Challenges in Cloud Computing Environments. *IEEE Security and Privacy*, 8(6):24–31.

Vashist, S. K., Schneider, E. M., and Luong, J. H. (2014). Commercial Smartphone-Based Devices and Smart Applications for Personalized Healthcare Monitoring and Management. *Diagnostics*, 4(3):104–128.

Wei, X., Gomez, L., Neamtiu, I., and Faloutsos, M. (2012). Permission Evolution in the Android Ecosystem. In *Proceedings of the 28th Annual Computer Security Applications Conference*, ACSAC '12, pages 31–40.

Zeevi, D., Korem, T., Zmora, N., Israeli, D., Rothschild, D., Weinberger, A., Ben-Yacov, O., Lador, D., Avnit-Sagi, T., Lotan-Pompan, M., Suez, J., Mahdi, J. A., Matot, E., Malka, G., Kosower, N., Rein, M., Zilberman-Schapira, G., Dohnalová, L., Pevsner-Fischer, M., Bikovsky, R., Halpern, Z., Elinav, E., and Segal, E. (2015). Personalized Nutrition by Prediction of Glycemic Responses. *Cell*, 163(5):1079–1094.