YaPPL - A Lightweight Privacy Preference Language for Legally Sufficient and Automated Consent Provision in IoT Scenarios*

Max-R. Ulbricht[0000−0001−7134−4351] and Frank Pallas[0000−0002−5543−0265]

TU Berlin,
Information Systems Engineering,
Einsteinufer 17,
10587 Berlin,
Germany
{mu, fp}@ise.tu-berlin.de
www.ise.tu-berlin.de

Abstract. In this paper, we present YaPPL — a Privacy Preference Language explicitly designed to fulfill consent-related requirements of the GDPR as well as to address technical givens of IoT scenarios. We analyze what criteria consent must meet in order to be legally sufficient and translate these into a formal representation of consent as well as into functional requirements that YaPPL must fulfill. Taking into account further nonfunctional requirements particularly relevant in the IoT context, we then derive a specification of YaPPL, which we prototypically implemented in a reusable software library and successfully instantiated in a proof of concept scenario, paving the way for viable technical implementations of legally sufficient consent mechanisms in the IoT.

Keywords: Privacy Preference Language · Internet of Things · Consent.

1 Introduction

In a world pervaded by connected things, where mobile phones, wearables, environmental sensors and smart home components constantly communicate with backend infrastructures and, through these, are dynamically interconnected with further services, it becomes increasingly challenging for device owners to keep track and control of respective data transfers. Due to the foreseeably growing complexity of such IoT environments, technical mechanisms and tools will become virtually indispensable for effectively exerting individual control over ones data.

At the same time, the collection and provision of legally sufficient consent — which is foundational for many realistic IoT applications — becomes increasingly

* This work is part of a project supported by funds of the Federal Ministry of Justice and Consumer Protection (BMJV) based on a decision of the Parliament of the Federal Republic of Germany via the Federal Office for Agriculture and Food (BLE) under the innovation support programme.

difficult, given the strict requirements of the GDPR (General Data Protection Regulation) on the one and the technical characteristics and constraints of IoT environments on the other hand. Without proper ways for obtaining and actually implementing legally sufficient consent, however, many practical IoT applications and services would lack the necessary basis of lawfulness and could thus not be implemented in a legally compliant way.

In this context, two core problems can be identified: First, individual consent must be adequately specific with regard to utilizers and purposes in order to provide a legally sufficient basis for the collection and processing of personal data. Current practices and technical approaches for consent provision — especially in the IoT context with dynamically changing interconnections of devices and multiple services — do typically not meet this requirement but rather follow an approach of overly "broad consent" instead. Second, consent must also fulfill form-related obligations such as informedness or explicitness. Meeting these obligations requires appropriate interfaces, which IoT devices typically lack.

To solve these challenges and pave the way for viable technical implementations of legally sufficient consent mechanisms in the IoT context, we propose a lightweight, tripartite approach consisting of (1) a policy specification service running on edge devices with sufficient user interaction capabilities, (2) a policy provider persistently storing user-specified preferences and previous versions thereof, and (3) a policy proxy applying the preferences to concrete, purpose-specific data requests. All these components function and interact on the basis of a preference language that is capable of codifying consent in line with GDPR requirements.

Contributions: The design and specification of this preference language — named YaPPL — is the primary subject of this paper. Through conscious integration of legal requirements into the language design from ground up, YaPPL allows to codify legally sufficient consent and thus provides a valuable basis for GDPR-compliant consent management. In a nutshell, YaPPL is a message and file format that, in combination with the proposed service architecture,

a) fulfills legal requirements for technically mediated consent provision
b) can act as an archive for expired preferences for auditing purposes
c) provides an enhanced user-centric access control model for future or unforeseen data processing requests.

Furthermore, YaPPL is explicitly designed to suit constrained execution environments like those typically present in the IoT context. Besides the specification, we also implemented a prototypical YaPPL software library and, on this basis, the three above-mentioned components. We then instantiated these components in a concrete, realistic setting consisting of constrained IoT devices and multiple cloud services. Preliminary experiments conducted in this setting strongly point towards YaPPLs practical viability, vividly demonstrating the potential of consequently designing technology to address currently unsolved legal challenges.

Structure: The remainder of this paper is organized as follows: In section 2, we analyze the requirements for legally sufficient consent, explore constraints given by IoT environments and briefly sketch our assumed overall architecture. In section 3, we develop a formal representation of consent and identify functional as well as non-functional requirements resulting from the previous considerations. On this basis we then provide the language specification for YaPPL and explain its core functionalities. Finally, we discuss YaPPL in the light of related work (section 4) and conclude.

2 Consent in the IoT context

Consent is one of the cornerstones of modern privacy regulations, following the idea that individuals should be able to determine who knows what about them or, respectively, what facts about their private life are communicated to others [27]. This fundamental understanding has not only influenced legislative procedures [6], it is also reflected in numerous privacy principles [3], guidelines [2] and frameworks/standards [4].

Under the legal regime of the GDPR, consent must fulfill certain criteria in order to provide lawfulness for the collection or processing of personal data. To establish a sound foundation for technically representing consent in a form that satisfies legal requirements, we therefore briefly analyze these criteria below. In addition, we shortly explore constraints given by IoT environments and sketch a technical architecture facilitating the technically mediated provision of legally sufficient consent in IoT scenarios for which the preference language to be developed herein shall serve as the underlying basis.

2.1 Legal Requirements for Consent

According to the *Principles relating to processing of personal data* provided in Article 5 of the GDPR [6], "personal data shall be [...] processed lawfully, fairly and in a transparent manner [...]". *Lawfulness* can be assumed if "the data subject has given consent to the processing of his or her personal data for one or more specific purposes". Article 7 (*Conditions for consent*) stipulates additional conditions that consent has to fulfill in order to be legally sufficient. In particular, given consent has to be easily retractable at any time and the party that initially collects the personal data has to ensure that it is able to demonstrate that a specific data subject has actually consented to the processing of his or her data [6, Article 7 (1)].

Furthermore, the legal sufficiency of consent is subject to form-related requirements. Besides the fact that it must be freely given, consent particularly must be a "specific, informed and unambiguous indication of the data subject's wishes" and needs to be provided "by a statement or by a clear affirmative action" according to Article 4 of the GDPR [6]. Specificity, the quality of being informed, and the need for a clear affirmative action therefore deserve closer examination.

Specificity. In the GDPR itself it is not explicitly specified how *"specific consent"* is to be interpreted. However, it is at least declared that consent has to be given for "one or more specific purposes" [6, Article 6 (1 a)]. Furthermore, the Article 29 Data Protection Working Party (Art. 29 WP) comments that consent, to be specific, "should refer clearly and precisely to the scope [...] of the data processing. It cannot apply to an open-ended set of processing activities. [...] In other words, blanket consent without specifying the exact purpose of the processing is not acceptable" [8]. If data should be processed for more than one purpose, the controller "should provide a separate opt-in for each purpose, to allow users to give specific consent" [10]. Any technical representation of consent must therefore allow to codify specific purposes at a sufficient level of detail.

Informedness. There are clear explanations within the GDPR itself about the meaning of *"informed consent"*. In recital 42 of the regulation [6], it is stated that "[f]or consent to be informed, the data subject should be aware at least of the identity of the controller and the purposes of the processing for which the personal data are intended". Thus, legally sufficient consent does at least require that the data subject knows who intends to processes their personal data for what reason. Again, any viable technical representation of (requested and given) consent must allow to codify this information.

Clear affirmative action. Finally, a controller has to ensure that consent is provided in an unquestionable manner. Acceptable forms, according to recital 32 of the GDPR [6], are oral or written statements, including by electronic means, but also "ticking a box when visiting an internet website" or "choosing technical settings for information society services". As opposed to such clear affirmative actions, "[s]ilence, pre-ticked boxes or inactivity should not therefore constitute consent" [6, Recital 32]. Even though this requirement primarily regards user interfaces, the explicitly mentioned choice of technical settings is of particular relevance for us, as it allows for more technical approaches of consent provision, which is particularly necessary in the IoT context.

2.2 IoT — Systems Perspective

As noted by the Article 29 Data Protection Working Party (Art. 29 WP) [9] the Internet of Things raises several new challenges regarding the legally compliant provision and withdrawal of consent. In particular, sensor devices like fitness trackers or personal weather stations are often not designed to provide extensive information by themselves and also do not contain any interfaces sufficient to obtain individual consent.

Beside this lack of useful interfaces, components of an IoT environment are often constrained in their computation, storage and communication capabilities [22]. For our consent management architecture proposed in the next section, we classify components into three different categories: 1) sensors devices, 2) edge-devices and 3) cloud components.

Of these, sensor devices usually have none or only few computational and storage capabilities. Interfaces for interacting with the device itself are also missing. Edge-devices, in contrast, offer resources to communicate with users as well as with sensors and higher level services, to cache or store and (pre-) process sensor data before sending them to the respective cloud service components [24], but are semi-constrained with regard to their computational power and/or storage space. Cloud components, in turn, have virtually unlimited computational and storage capabilities.

Thus, from an IoT systems perspective, tasks involving substantial processing and/or storing sensor or other data, like privacy preferences [23], can only be performed either in the cloud or on the networked edge devices — bridging the transfer of measured data between sensor devices and the cloud — considering the semi-constrained characteristics of the latter ones.

2.3 Scenario & Architecture

Because of the lack of interaction possibilities on sensor devices, the provision or revocation of consent needs to be performed on another device under the control of the sensor owner. In most cases, sensor devices use the owners' desktop computers, home routers or mobile computing devices to synchronize their data with the cloud backends of associated service providers. These edge-devices are therefore promising points of operation for the management of consent statements [23].

The Art. 29 WP suggests the use of *Privacy Proxies* in the context of IoT applications. By employing such a *Privacy Proxy* for "executing" technically codified consent statements, "data requests are confronted with predefined policies governing access to data [...]. By defining sensor and policy pairs, third parties requests for collection or access to sensor data would be authorized, limited or simply rejected." [9, p. 21, note 30]

Picking up the idea of edge devices as configuration points for a *Privacy Proxy* governing access to personal data, figure 1 shows our generalized IoT architecture with sensors, edge-devices and cloud storage systems, supplemented by a *Policy Provider* and a *Policy Proxy*. Users of sensor devices can specify their preferences regarding the further use of their sensor data on their edge-device before the upload to the cloud storage system of the respective service provider is triggered. To avoid the necessity of a new consent provision for every new data processing activity by the service provider, the *Specification Service* on the user-controlled edge-device is used for "choosing technical settings" (see section 2.1). These are then stored in the form of policies by another service, the proposed *Policy Provider*. For incoming requests regarding the sensor data, the *Policy Proxy* ask the *Policy Provider* for an associated policy and answers the request according to the rules and preferences stated by the sensor owner in the received policy.

Thus, our architecture not only implements the recommendations of the Art. 29 WP regarding consent provision in IoT environments, it also can act as an archive for expired preferences for auditing purposes as required by the GDPR.

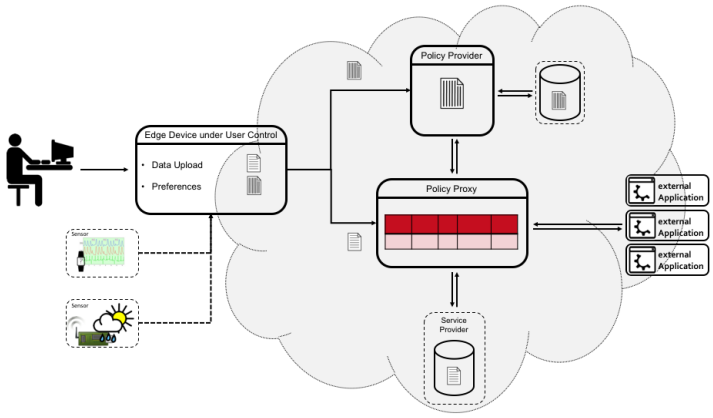

Fig. 1. IoT architecture with a user-controlled edge device running a Preference Specification Service. A Policy-Provider-Service and a Policy-Proxy-Service as a database wrapper guarding the access to data.

Beyond that, it provides an enhanced user-centric access control model for future or unforeseen data processing purposes. In this model, we give owners of sensor devices full control but also the responsibility to manage preferences regarding the future usage of their sensor data by shifting the Policy Administration Point (PAP) as known from usual access control systems completely to the user. While the proposed *Policy Provider Service* acts as a Policy Information Point (PIP), the *Policy Proxy Service* substitutes the Policy Decision Point (PDP) as well as the Policy Enforcement Point (PEP) from conventional access control architectures. If these services are implemented as micro-services, which communicate through standardized interfaces (like REST), the integration into different existing infrastructures is easily manageable. They could be integrated into platforms [26] or deployed as independent distributed (micro-) services.

In order to enable such an architecture, we need 1) a standardized *policy / preference specification language* as well as 2) a respective *message format* to make sure that the three services mentioned above can communicate on the basis of a shared understanding of how to parse, create, evaluate and enforce the policies containing the sensor owners' data processing preferences. Taking into account the legal requirements for consent outlined above, both will be developed below.

3 YaPPL

In this section we will introduce the specification of YaPPL. We start with a formalized representation of consent, formulate requirements and motivate our design decisions to derive the internal structure of a YaPPL policy on that basis. Afterwards we present a concrete example of a policy from our prototypical

implementation, illustrate the core functionalities of our software library and discuss our evaluation.

3.1 Formalization of Consent

As a first step towards our language, we need a formal representation of consent in the sense of the GDPR as outlined above. Consent *(C)* as broadly assumed in legal discussions can, on an abstract level, be interpreted as a relation that maps a tuple containing a utilizer[1] *(U)* and a data processing purpose *(P)* specified by the utilizer onto data to be released. Interpreted as permission to process (personal) data *(D)* related to a data subject *(S)*, we get the following function, whereat D_{col} represents all *collected* data and D_{rel} the data to be *released*:

$$c : D_{col} \times C \to D_{rel} \implies D_{col} \times U \times P \to D_{rel}$$

$$c(d_{col}, u, p) := \begin{cases} d_{rel} = d_{col} & \text{if } \textit{consent} \text{ is given by S} \\ \emptyset & \text{if } \textit{consent} \text{ is revoked or not present} \end{cases}$$

We strive for an understanding of consent that goes beyond this established conception and allows data subjects to specify transformations T that reflect their wishes for personal data relating to them to be preprocessed before disclosure. The introduced *transformation* can be used to specify computational tasks that have to be performed before the data is transferred. Examples are different types of aggregation, pseudo-/anonymization, or cutting out certain data-points from series of measurements. Thus, T is a transformation from data into other, new data depending on the combination of utilizer and purpose:

$$t \in T : D_{col} \to D_{rel}; \quad \implies t_{up}(d_{col}) = d_{rel}$$

We now can use the specification of t to reflect all kinds of possible consent. If, e.g., consent should be revoked or not be given for certain data, we can specify t in way that $t_{up}(d_{col}) = d_{rel} = \emptyset$. If we generalize this approach towards using transformations as a basic concept for a consent-based privacy preference language, for every single datapoint or whole datasets, a 3-tuple containing a utilizer u, a processing purpose p and a transformation t forms a resulting rule r that describes which data may to what extent be transferred to the requesting institution:

$$r : D_{col} \times U \times P \times T \to D_{rel}$$

$$r(d_{col}, u, p, t) = t_{up}(d_{col}) = d_{rel}$$

Based on this approach, data subjects can define rules that exactly codify their consent regarding which data can be released to a given set of utilizer and purpose. On this formalized basis, we can now proceed defining our aspired language, with rules that act as consent statements.

[1] In the understanding pursued herein, this is the institution that is aspires to collect and/or process personal data – in legal terms: the controller and/or processor

3.2 Requirements & Design Decisions

Besides the legal aspects outlined in section 2.1, the requirements for the development of a formal language designed to enable specifications of user preferences can be divided into functional and nonfunctional factors. The motivation to develop an own language is particularly rooted in the intention to give domain experts — namely, in the case of a description language for consent-based usage preferences, legal practitioners — the possibility to play a significant role in the design, development, and maintenance of new systems in their domain [20]. For this reason, we consciously chose language constructs and abstract concepts that also non-technologists can easily read, create, and — in the event of audits — evaluate without further utilities. The following requirements will later be addressed and explained in the design decisions section:

Functional Requirements

- **Consent:** distinct representation of consent statements as a combination of processing purposes, potential utilizers and transformations.
- **Purposes & Utilizers:** representation of possible processing purposes and potential utilizers as directed graph to emphasize relationships between entities as well as hierarchies.
- **Transformations:** distinct specification of domain specific transformation functions

Nonfunctional Requirements

- **Readability** should be increased by the usage of appropriate formats that are human- as well as machine readable
- **Extendability** regarding new purposes, utilizers and transformation functions
- **Efficiency** regarding the memory and resource consumption on limited IoT devices

Design Decisions In our language, we address these functional and nonfunctional requirements as follows:

Functional As deduced in section 3.1, any single preference rule has to contain at least the three mentioned and obligatory components, *utilizer*, *purpose*, and *transformation*. As they must be adjustable for different domain specific needs, all of these are represented by plain strings within a rule. In our prototypical implementation, we codify the directed graphs containing available utilizers, processing purposes, and generalized categories thereof in JSON configuration files that are integrated at runtime, which allows to easily adjust the respective available vocabularies to, e.g., domain-specific purpose hierarchies.

Since we want to be able to give and codify conditional consent for different circumstances (like, e.g., “Utilizer A is allowed to process my data for purpose

X without any restrictions, but for utilizer B only weekly aggregates should be released although it is for the same purpose X"), multiple independent *rules* can be combined in a single *preference*. It is thus possible to assign different transformations to be applied for different utilizers and/or purposes.

To facilitate legally sufficient consent provision also for future and unforeseen purposes and utilizers, we allow the nested attributes "permitted"and "excluded" to be provided for both, utilizer and purpose. With this extension, it is possible to explicitly exclude purposes and utilizer, allowing to make statements like "my data can be used for all future purposes except ..." or "my data can be used by anybody except ...".

*Remark 1 (**Conflicting Rules**).* Even if we can try to prevent the establishing of conflicting rules by carefully checking the creation or update processes inside the *Preference Specification Service*, one should be aware of conflicting rules. Thus, in the *Policy Proxy Service*, which acts as a Policy Enforcement Point (PEP), mechanisms have to be in place that ensure the proper execution. To comply with the basic idea of a prohibition with a reservation of authorization, which constitutes all data protection regulations, rules preventing data transfers should always have higher priorities than the ones allowing them.

To address the fact that a single sensor device often collects multiple measurands (e.g. smartwatches can record heart-rates, sleep phases, etc.), we designed the *transformation* field of a rule as a list of transformation objects, each containing a transformation function and the sensor attribute to which it refers. So far, we only implemented simple examples — like an *average* and a *minmax* function with the possibility to assign different time intervals — in our prototype. However, additional names of available transformation functions can be subsequently added through configuration files as soon as they are specified and implemented in the *Policy Proxy Service* to be enforced in operation.

*Remark 2 (**Revocation & Archiving**).* We decided to integrate two timestamps into each rule, represented by the *'valid_from'* and the *'exp_date'* attributes. The *'valid_from'* value is initialized with the current timestamp of the moment the rule is created, whereas the *'exp_date'* value is initialized with a zero value ("0000-00-00T00:00:00.00Z"). Thus, we can easily determine whether a rule is (or was at a given time) valid by evaluating this attribute. The value of *'exp_date'* is only changed under two conditions: 1) when the sensor owner decides to delete the rule — which means withdrawing this specific consent statement. In this case, the *'exp_date'* is set to the current timestamp and thus renders the rule invalid. Noteworthily, invalidated rules are not deleted but rather kept *archived*. The other case of *'exp_date'* being changed is 2) when a sensor owner updates the rule with different values for utilizers, purposes or transformations. In this case, the *'exp_date'* is also set to the current timestamp, the respective rule renders invalid and thus is *archived*. Simultaneously, a new rule with the old values is created which can then be updated to the user's changed intentions. This procedure makes sure that it is always possible to track back the evolution of the preference as a whole and thus to fulfill the legal requirement from Article

7 (1) of the GDPR [6], which obligates a utilizer to be able to demonstrate that a data subject has consented to the processing at a given moment.

Nonfunctional To address the aforementioned nonfunctional requirements regarding readability in combination with the efficiency criteria that has to be considered in IoT environments, we chose JSON[2] both as file- and as message-format, because policies in the JSON format are human- as well as machine-readable. As an alternative, we also considered the more generic YAML[3] format. Besides the pro-YAML-argument that YAML is slightly easier to 'parse' for humans [14], technical arguments led to the decision against YAML. In particular, benchmarks have shown that the processing time for serializing and deserializing YAML files can be up to 10-20 times longer than the processing of the same data codified in JSON [1, 14]. Furthermore, because of the structure of JSON messages, it is easy to decide if a data transmission has failed, as an interrupted transmission will automatically render to invalid code. For a message format to be used in communication between different web-services, this is a significant advantage.

As a further alternative to JSON, we also considered XML because of its also widespread use in the service domain. But if we take the requirements in IoT contexts seriously and compare XML and JSON with regard to resource consumption for parsing and evaluating data or the storage footprint of data files, JSON is faster to process [21] and more efficient in resource consumption both at rest and 'on the wire'. Thus, JSON is the better fit in an IoT context [16]. Also with respect to human readability JSON has advantages over XML.

*Remark 3 (**Extendability**).* Regarding the last nonfunctional point we decided that the fundamental structure of policies and rules has to be stable and consistent. Besides the timestamps, every rule therefore must have declarations for processing purposes, utilizers and transformation functions, as these are essential for sufficient consent statements. However, and as already mentioned above, it is possible to adapt our design to different domains just by adjusting or modifying the options available for these three obligatory components into a domain-specific variation simply by using different configuration files.

3.3 Policy Format Specification & Example

A YaPPL policy itself contains two first-level attributes: an *id*, which is used to link specific sensor data to a corresponding policy, and a *preference*-block which contains a list of one or more *rules*. These rules codify consent-based usage preferences to regulate the access to the values measured by the respective sensor. A *rule* consists of the three main building-blocks we have derived from our formalized consent model in section 3.1, namely *purpose*, *utilizer* and *transformation*. As delineated in the design decisions section 3.2, the *purpose* part as well as the *utilizer* part of a rule are both divided into a *permitted* and a *excluded* block.

[2] `https://json.org/`
[3] `http://www.yaml.org/`

In listing 1.1 a snippet representing such a *rule* part of a policy is extracted out of the complete BNF for YaPPL, which will be accessible online later on. The aforementioned mandatory building-blocks of every rule (*purpose*, *utilizer*, *transformation*) are complemented by a pair of additional attributes, which obviously are timestamps. The *valid_from* value is set at (and to) the time of the rule creation, whereas the value of the *exp_date* attribute is initialized with a "zero" value and an alteration can be triggered for different reasons like the revocation of the represented consent statement or the update of the respective rule with new settings regarding *purpose*, *utilizer* or *transformation*, as illustrated in the design decisions in section 3.2.

Listing 1.1. "rule" snippet from BNF Policy Specification

```
<rule> ::= '{'
           <purpose> ','
           <utilizer> ','
           <transformation> ','
           <valid_from> ','
           <exp_date>
           '}'

<purpose> ::= '"' 'purpose' '"' ':' '{'
              <permitted_purpose> ','
              <excluded_purpose>
              '}'

<utilizer> ::= '"' 'utilizer' '"' ':' '{'
               <permitted_utilizer> ','
               <excluded_utilizer>
               '}'
```

The policy shown in listing 1.2 is an example from our prototypical implementation in an IoT testbed. It is used to specify user preferences regarding an indoor environmental sensor which is capable to measure temperature, illumination, air pressure and light spectrum.

Listing 1.2. Example Policy with only one Rule

```
{
  "_id": 4493,
  "preference": [
    {
      "rule": {
        "purpose": {
          "permitted": ["statistics", "planology"],
          "excluded": ["commercial"]
        },
        "utilizer": {
          "permitted": ["wikimedia", "tu_berlin"],
          "excluded": ["netatmo", "gate5"]
        },
        "transformation": [
          {
            "attribute": "temperature",
            "tr_func": "minmax_hourly"
          }
        ],
        "valid_from": "2017-10-09T00:00:00.00Z",
        "exp_date": "0000-00-00T00:00:00.00Z"
      }
    }
  ]
}
```

The ensuing *transformation* block is organized as a list of transformation objects, which represents the conditions for the processing of specific values. Due to the *attribute* field, it is possible to assign different transformation functions to each measurand of a multi sensor device. This is especially useful for the adoption of YaPPL in the wearables sphere, where sophisticated swartwatches capture multiple body- and health-related values like heartrate, sleep phases or the gps route of the last training run. As examples for these transformation functions, we implemented an average as well as a minmax function. While the first one calculates the arithmetic average of a bulk of passed values, the latter returns a minimum and a maximum value of the passed data. Both functions can be triggered with different time intervals like daily, monthly and so on to aggregate values as desired by the sensor owner.

3.4 Core Functionalities based on YaPPL

On the basis of the aforementioned structure and characteristics of YaPPL policies, the implementation of services which enable consent-based data sharing or data processing activities is straightforward. To showcase the possibilities of YaPPL, we developed a prototypical software library that is integrated into the three services mentioned above to manage legally sufficient consent provision in an IoT testbed. Besides the necessary parser, a validator for the policies, and the usual and obvious CRUD methods (Create, Read, Update and Delete) for rules, this library provides two essential functionalities needed to fulfill the preferences codified in a YaPPL policy: First, we need to know all excluded entities to prohibit further data transfers, and second, we must be able to extract explicit transformation rules in order to customize the sensor values according to the wishes of the sensor owner.

getExcludedEntities The *excluded* blocks in both, the *purpose* and the *utilizer* part of the rules are intended to explicitly prevent data transfers to specific institutions, generalized categories of potential utilizers (like, e.g., the military or advertising companies), specific processing purposes or categories thereof (e.g., all commercial purposes). There are two methods in our YaPPL library which return the respective lists by traversing all valid rules inside a given policy. If the *policy proxy* receives a request from *'Institution_G'* to provide sensor values for *'Purpose_R'*, it fetches all respective *policies* from the *policy provider* and decides if a transfer is allowed. If neither *'Institution_G'* nor *'Purpose_R'* is in the returned lists, the proxy will extract the *TransformationRules* (as described in the following section) to preprocess the sensor values according to the rules.

getTransformationRules If a requesting institution with a specific data processing purpose is not excluded from data transfer by the respective rules, the policy proxy will extract a list of transformation objects, each containing a list of (concrete or generalized categories of) permitted utilizers, a list of permitted processing purposes and corresponding transformation functions. Thus, every transformation object can be interpreted as a users consent to the processing

of her sensor data for the covered purposes by the contained utilizers under the given conditions (aka. transformations). The proxy passes all original sensor data to the appropriate transformation function, catches the aggregated results and responds to the original request.

3.5 Evaluation

To test our concept we have developed a library, as mentioned above, implemented all components of the consent management architecture as introduced in section 2.3 and integrated the services into a real world IoT application. As this application is a participatory sensing platform, we implemented the specification service for adjusting preferences on a small sensor device that transfers the measured data — supplemented by an extra metadata field which contains the URL of the respective policy — to the platform and sends the appropriate policies to our *policy provider*, which runs inside a container on a cloud server. The *privacy proxy*, also running on a cloud server, mirrors the API of the sensing platform and monitors the requests. When values of a sensor with the aforementioned metadata field are requested, the request is intercepted, the appropriate policy is fetched from the *policy provider* and evaluated. If the data transfer is permitted, the sensor values are pre-processed according to the transformation rules and transferred to the requesting utilizer.

While performance considerations are left to future work, our prototype performs well as a proof of concept and fulfills the requirements deduced in the previous sections.

4 Related Work & Discussion

Some of the ideas presented herein are inspired from previous works on *purpose based access control* [11, 15, 18]. The original concept of the division of purposes into *'prohibited' and 'intended'* purposes, which we also use for utilizers, was proposed by Byun, Bertino & Li [11].

Furthermore a vast amount of languages for privacy-, security- and access control-policies exists (see [19] for a comprehensive overview). The most prominent one in the area of privacy policy languages regarding websites and -services is P3P [13] and its companion for user preferences named APPEL [12]. P3P has a limited vocabulary with predefined values for purposes and recipient. While the predetermined purposes hinder an adaption to other or unforeseen processing purposes, the preassigned values for potential recipients of the respective data are not explicit but characterize the relation to the data controller that originally collected the data. In addition, even if the P3P vocabulary is limited, the correct formulation of user preferences in APPEL to match them with P3P policies is difficult and error prone [7].

As a standard for the formulation of attribute based access control policies, XACML [5] has been established over the last decade. Since it was designed to be used by institutions to regulate the access to data and resources within

organizations or federations thereof, the usability as preference language for end users seems at least questionable. Thus, while XACML is capable of representing fine-grained access control policies, for the usage as preference language it needs extensions: the PrimeLife Policy Language pursues a similar approach to specify preferences for the future use of specific data like YaPPL. The idea of *Downstream Usage* describes the definition of conditions for the usage of data by third parties after the initial collection and storage. Also obligations are available as a counterpart to YaPPL *transformations* [25]. But since it is an extension to XACML, based on XML and with an extensive ruleset for the formulations of policies, it inherits all the shortcomings mentioned in section 3.2 and policies are by far not human readable or auditable by legal practitioners without the help of additional tools or technicians.

Besides these examples numerous other approaches for privacy preference languages exits. Some are lightweight enough for the usage in IoT environments but use strong compression or binary formats and are therefore not human read- and auditable (e.g. [17]). Most others try to achieve human readability by using XML dialects with the aforementioned drawbacks. Most importantly, however, none of them is explicitly designed to fulfill the legal requirements regarding the provision of consent, acts as an archive for expired preferences for auditing purposes and provides an enhanced access control model for future or unforeseen data processing requests.

While we worked on YaPPL, several open questions arose, which are out of scope of a technical solution, but nonetheless important to discuss for real world use cases. Some examples are:

- Who is responsible for the initial state of the utilizer and purpose graph?
- How should the extension of these graphs be organized?
- Who is responsible for adding new graph entries into appropriate categories?

Beside these more or less organizational problems, also operational questions have to be answered. Dependent from the deployment of the proposed services varied communication patterns have to be established. If, e.g., the *Policy Proxy Service* is operated by another institution than the original service provider, a transfer of all data to the proxy with a subsequent execution of the *transformation* functions is not feasible. In this case, the policy evaluation should lead to a query modification. Thus, the proxy would not see any data not intended for release. Such questions are strongly related to operator models and matters of trust.

These and several further questions will need to be addressed during future developments and, in particular, through implementing our concept in real use cases with practitioners.

5 Conclusion & Outlook

As we have outlined in this paper, technical representations of consent must meet certain requirements in order to be legally sufficient under the GDPR. In

particular, this regards the specificity in matters of utilizer and purpose as well as form-related requirements for informedness and clear affirmative actions. In the IoT context, especially the form-related requirements can, however, hardly be met without novel mechanisms and approaches of technical support.

Based on a legal analysis and a formalization of consent, we therefore designed and specified YaPPL, a Privacy Preference Language explicitly designed to fulfill consent-related requirements of the GDPR as well as to suit constrained execution environments like those typically present in the IoT context. We implemented YaPPL in a reusable software library and successfully employed this library to achieve technical consent awareness in a realistic IoT scenario, thus demonstrating the practical viability of our approach. Besides legally sufficient consent provision in IoT environments, the presented file- and message-format in combination with the proposed architecture can also act as an archive for expired preferences and provide a user-centric access control model for future or unforeseen data processing purposes. YaPPL is therefore a valuable contribution to paving the way for sustainable technical implementations of legally sufficient consent mechanisms in the IoT context.

Beyond this, we explicitly foresee YaPPL and the underlying approach to be also applied to IoT scenarios that do *not* fall under the GDPR but still call for the possibility to technically represent and implement differentiated usage preferences. For instance, this could refer to scenarios of participatory environmental sensing, where participants might also be empowered to explicitly govern the use of data provided by them with regard to utilizers and purposes based on the technologies presented herein. The mere fact that certain data are not "personal data" in the sense of the GDPR does of course not invalidate the general validity of consent principles materialized therein.

The YaPPL language specification as well as the mentioned library will shortly be released under an open source license and can thus be used and extended in other projects as well.

References

1. Serializing data speed comparison: Marshal vs. JSON vs. Eval vs. YAML. http://www.pauldix.net/2008/08/serializing-dat.html
2. OECD Guidelines on the Protection of Privacy and Transborder Flows of Personal Data (1980)
3. Privacy Online: A Report to Congress. Tech. rep., FTC (Jun 1998)
4. ISO/IEC 29100:2011 - Information technology – Security techniques – Privacy framework (2011)
5. eXtensible Access Control Markup Language (XACML) Version 3.0. OASIS Standard (2013)
6. Regulation (EU) 2016/679 of the European Parliament and of the Council of 27 April 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing Directive 95/46/EC (General Data Protection Regulation). Official Journal of the European Union **L 119/1**, 1–88 (Apr 2016)

7. Agrawal, R., Kiernan, J., Srikant, R., Xu, Y.: XPref: A preference language for P3P. Computer Networks **48**(5), 809–827 (Aug 2005)
8. Article 29 Data Protection Working Party: Opinion 15/2011 on the definition of consent. Tech. rep. (Jul 2011)
9. Article 29 Data Protection Working Party: Opinion 8/2014 on the on Recent Developments on the Internet of Things. Tech. Rep. 14/EN WP 223 (Sep 2014)
10. Article 29 Data Protection Working Party: Guidelines on Consent under Regulation 2016/679. Tech. Rep. WP259 rev.01 (2018)
11. Byun, J.W., Bertino, E., Li, N.: Purpose Based Access Control of Complex Data for Privacy Protection. In: Proceedings of the Tenth ACM Symposium on Access Control Models and Technologies. pp. 102–110. SACMAT '05, ACM, New York, NY, USA (2005). https://doi.org/10.1145/1063979.1063998
12. Cranor, L., Langheinrich, M., Marchiori, M.: A P3P Preference Exchange Language 1.0 (APPEL1.0). Tech. rep., World Wide Web Consortium (2002)
13. Cranor, L.F.: Web Privacy with P3P. O'Reilly Media (2002)
14. Eriksson, M., Hallberg, V.: Comparison between JSON and YAML for data serialization. Tech. rep., The School of Computer Science and Engineering Royal Institute of Technology (2011)
15. Ghani, N.A., Selamat, H., Sidek, Z.M.: Credential Purpose-based Access Control for Personal Data Protection. J. Web Eng. **14**(3&4), 346–360 (2015)
16. Guinard, D., Trifa, V., Mattern, F., Wilde, E.: From the Internet of Things to the Web of Things: Resource-oriented Architecture and Best Practices. In: Uckelmann, D., Harrison, M., Michahelles, F. (eds.) Architecting the Internet of Things, pp. 97–129. Springer Berlin Heidelberg (2011)
17. Henze, M., Hiller, J., Schmerling, S., Ziegeldorf, J.H., Wehrle, K.: CPPL: Compact Privacy Policy Language. In: Proceedings of the 2016 ACM on Workshop on Privacy in the Electronic Society. pp. 99–110. WPES '16, ACM, New York, NY, USA (2016). https://doi.org/10.1145/2994620.2994627
18. Kabir, M.E., Wang, H., Bertino, E.: A role-involved purpose-based access control model. Information Systems Frontiers **14**(3), 809–822 (2012)
19. Kasem-Madani, S., Meier, M.: Security and Privacy Policy Languages: A Survey, Categorization and Gap Identification (2015)
20. Mernik, M., Heering, J., Sloane, A.M.: When and How to Develop Domain-Specific Languages. ACM computing surveys (CSUR) **37**(4), 316–344 (2005)
21. Nurseitov, N., Paulson, M., Reynolds, R., Izurieta, C.: Comparison of JSON and XML data interchange formats: A case study. In: Proceedings of the ISCA 22nd International Conference on Computer Applications in Industry and Engineering, CAINE 2009. pp. 157–162. San Francisco, California, USA (2009)
22. Reinfurt, L., Breitenbücher, U., Falkenthal, M., Leymann, F., Riegg, A.: Internet of things patterns. pp. 1–21. ACM Press (2016)
23. Satyanarayanan, M.: The Emergence of Edge Computing. Computer **50**(1), 30–39 (Jan 2017). https://doi.org/10.1109/MC.2017.9
24. Shi, W., Dustdar, S.: The Promise of Edge Computing. Computer **49**(5), 78–81 (May 2016). https://doi.org/10.1109/MC.2016.145
25. Trabelsi, S., Sendor, J., Reinicke, S.: PPL: PrimeLife Privacy Policy Engine. In: 2011 IEEE International Symposium on Policies for Distributed Systems and Networks. pp. 184–185 (Jun 2011). https://doi.org/10.1109/POLICY.2011.24
26. Ulbricht, M.R., Pallas, F.: CoMaFeDS - Consent Management for Federated Data Sources. In: Proceedings of the 2016 IEEE International Conference on Cloud Engineering Workshops. pp. 106–111. IEEE, Berlin (2016)
27. Westin, A.F.: Privacy and Freedom. Atheneum, New York (1967)