# How a Pattern-based Privacy System Contributes to Improve Context Recognition

Christoph Stach*, Frank Dürr*, Kai Mindermann†, Saravana Murthy Palanisamy*, and Stefan Wagner†
*Institute for Parallel and Distributed Systems | †Institute of Software Technology
University of Stuttgart
Universitätsstraße 38, 70569 Stuttgart, Germany
forename.surname@informatik.uni-stuttgart.de

***Abstract*—As *Smart Devices* have access to a lot of user-preferential data, they come in handy in any situation. Although such data—as well as the knowledge which can be derived from it—is highly beneficial as apps are able to adapt their services appropriate to the respective context, it also poses a privacy threat. Thus, a lot of research work is done regarding privacy. Yet, all approaches obfuscate certain attributes which has a negative impact on context recognition and thus service quality. Therefore, we introduce a novel access control mechanism called *PATRON*. The basic idea is to control access to information patterns. For instance, a person suffering from diabetes might not want to reveal his or her unhealthy eating habit, which can be derived from the pattern "rising blood sugar level" → "adding bread units". Such a pattern which must not be discoverable by some parties (e. g., insurance companies) is called *private pattern* whereas a pattern which improves an app's service quality is labeled as *public pattern*. PATRON employs different techniques to conceal private patterns and, in case of available alternatives, selects the one with the least negative impact on service quality, such that the recognition of public patterns is supported as good as possible.**

***Index Terms*—privacy, access control, pattern concealing, stream processing, complex event processing, databases.**

## I. Introduction

Mark Weiser's vision of *the computer for the 21st century* [1] became true due to the ubiquitous availability of so-called *Smart Devices*[1]. Various sensors such as cameras, microphones, location sensors etc. can be integrated into almost any kind of modern devices. Each single sensor captures only a certain aspect in most cases. However, by combining the data of different sensors, comprehensive and detailed insights about the users of these Smart Devices can be derived.

So-called *Smart Apps* use these context data in order to tailor their services to the user's current situation and thus increase their utility. According to Dey, the context comprises any kind of information that can be used to draw conclusions about the situation of a user [2]. Therefore, it is obvious that there are many different application areas for Smart Apps, including *eHealth* [3] or *Industry 4.0* [4].

Due to the huge amount of sensitive data and the profound knowledge which can be derived from this data, Smart Apps pose threats and providing benefits at the same time. The captured data is very informative as it contains both, knowledge about the user's *digital life* (e. g., browser history) as well as his or her *real life* (e. g., visited places) [5]. Thereby, a detailed profile about the user can be created. Such a profile contains highly valuable information and this knowledge is often misused. Consequently, privacy is a key issue in the context of Smart Apps [6].

The State of the art Privacy systems applied on Smart Devices in general either obfuscate or completely block certain sensor data. Whereas the granularity on which these systems operate differ—they could affect a sensor in total or only some of its functions—such a procedure always has a negative impact on service quality. Without access to a certain type of data the accuracy of context recognition is reduced. Especially in the eHealth domain such privacy systems cut both ways: On one hand, medical data might require some reactive measures particularly in emergency situations. On the other hand, eHealth apps still should be able to draw conclusions about the user's health condition [7] to provide the promised services.

Moreover, it is difficult for common users to understand which context information can be derived from which sensors. This is exacrbated by the fact that they might not know which are the additional data sources available to a certain Smart App. On that account, users are outrightly overchallenged by these privacy systems. The all-or-nothing approach of these systems—privacy or service quality—has the effect that users either grant Smart Apps unrestricted access to any requested data or they do not use a certain app at all [8].

To improve this state of affairs, we make the following five contributions: (1) We derive requirements of a privacy system for Smart Apps from a real-world use case. (2) We introduce a novel pattern-based access control system for Smart Apps called *PATRON*[2], where a pattern is defined as a sequence of events, for instance, captured by sensors. PATRON conceals private patterns from unauthorized applications, while exposing public patterns to ensure a certain quality of service. (3) We show the applicability of PATRON to both, real-time data as well as history data. (4) We implement various pattern concealing techniques in PATRON and the system selects the most appropriate one, i.e., the one which preserves the best data quality in terms of *false positives* and *false negatives* that in turn affects the service quality. (5) We use a quality metric

[1]We use the term Smart Device for any kind of device equipped with sensors and connectivity options.

[2]The acronym PATRON is short for **P**riv**a**cy in S**tr**eam Pr**o**cessi**n**g.

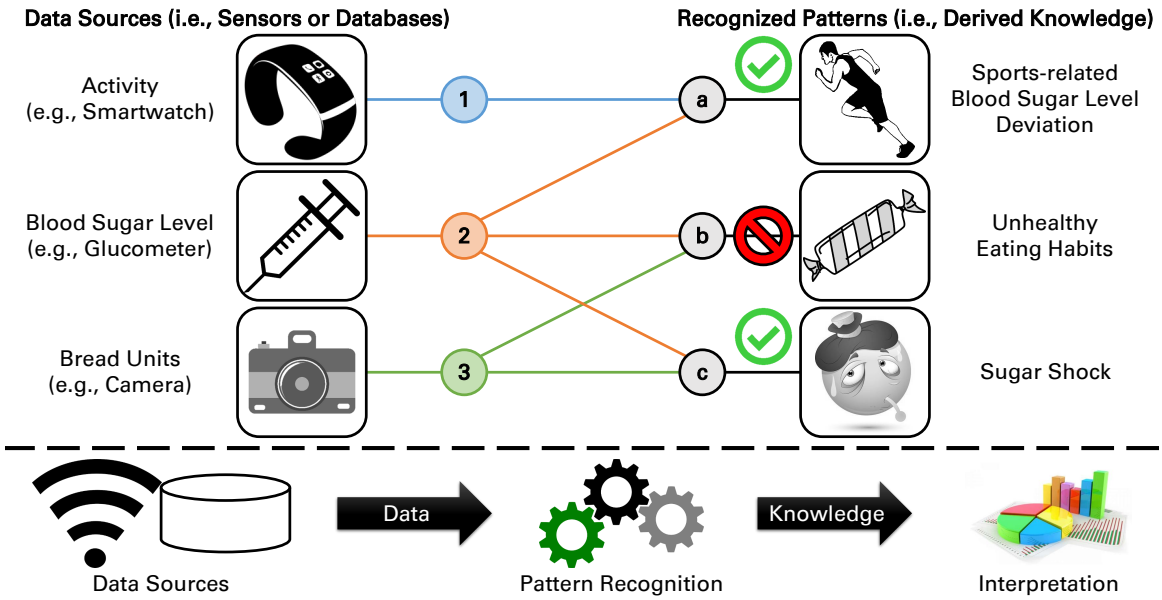


Figure 1. eHealth Examples for Data Sources and Derivable Knowledge Patterns.

to assess the performance of each concealing techniques in a given situation.

In our previous work, we already discussed legal aspects of PATRON, i. e., whether there are any legal limitations concerning the private patterns [9]. In the paper at hand, we examine to what extent an approach like PATRON restricts context recognition respectively how it improves it.

The rest of this paper is organized as follows: In Section II a real-world Smart App is introduced to point out special requirements of such apps towards a privacy system. Section III provides an overview on related work in the context of privacy systems for Smart Apps. Then, our PATRON approach is introduced in Section IV. Subsequently, Section V assesses its practicality before Section VI concluding this work.

## II. Use Case Analysis

In the health domain, Smart Apps can help to reduce medical costs, unburden physicians, and ease dealing with a disease. There is literally an app for any need [3]. Especially patients with chronic conditions such as diabetes benefit from Smart Apps, which is the reason why many health experts call for innovative and transdisciplinary research approaches [10]. Required duties (e. g., keeping a diabetes diary including activities, blood sugar levels, and consumed bread units) can be facilitated by apps, e. g., by integrating them into the patient's daily routine in a gamified manner [11]. The required data can be captured automatically or semi-automatically by sensors integrated into common Smart Devices: The current activity can be recognized by Smartwatches—even finger movement can be captured [12]. For continuous blood glucose monitoring, there are small sensors implanted in the upper arm [13]. The consumed bread units can be determined by analyzing pictures of a meal [14]. Yet, the capabilities of Smart Apps do not end here. They are able to combine these individual readings from multiple sensors and draw conclusions about the patient's condition.

Figure 1 shows three possible insights that a Smart App is able to derive from the given data sources: If the activity sensor detects that the patient is doing sport and the blood glucose monitoring notices a deviation of the blood sugar level shortly after, then Pattern $a$ is detected (i. e., the blood sugar level deviation is sports-related and therefore not critical). Pattern $b$ can be revealed by a rising blood sugar level after eating sweets (i. e., the diabetic patient has eating habits which are not good for his or her health). However, if the blood sugar level is low and the patient eats sweets as a follow-up then, Pattern $c$ is recognized (i. e., the patient ate sweets to prevent a sugar shock). Naturally, the patient wants to reveal Pattern $a$ and $c$ to his or her physician, whereas s/he wants to conceal Pattern $b$ for example to an insurance company at all costs (e. g., because of rising insurance rates in case of an unhealthy lifestyle).

As a Smart App must not be manipulated (e. g., by forcing it to "forget" that it has detected a certain pattern), one has to ensure that Pattern $b$ is not detectable in the first place. The most obvious approach is to block access to one of the data sources which are required to detect the pattern (i. e., Source 2 or 3). This approach is applied in most related work (see Section III). However, by suppressing data from Source 2 none of the three patterns is detectable. By blocking Source 3, Pattern $a$ is detectable but Pattern $c$ is still concealed. As Pattern $b$ and $c$ dependent on the same common sources, such an attribute-based approach cannot conceal Pattern $b$ and still reveal Pattern $c$, since it is far too restrictive. Moreover, common users do not know the correlations between data sources and patterns [8]. This is aggravated by the fact that Smart Apps do not only have access to real-time data, but also to history data. Some patterns become evident, when looking at the user's prior behavior: If

the user has a regular training session which is indicated by previous data, one could draw conclusions about his or her future activities, even without access to Source 1. This example shows that a privacy system is necessary for Smart Apps that has to meet specific requirements, which are postulated as follows.

**Requirements Specification.**

$R_1$ : **Pattern-based.** Privacy concerns are not raised at sensor level (i. e., raw data) but at knowledge level (i. e., sequences of events). Therefore, privacy rules should be mapped to patterns instead of single attributes. In conformity with Yeye He et al. [15], we differentiate between *public patterns*, i. e., patterns which should be detectable by Smart Apps, and *private patterns*, i. e., patterns which should be concealed at all costs.

$R_2$ : **Source Abstraction.** Since some patterns can be revealed by several sources (e. g., Smartphones and Smartwatches can identify the user's activity), the pattern description has to be source-independent.

$R_3$ : **Domain Abstraction.** Since each domain (e. g., health domain or industrial domain) has its specific data processing practices, domain experts have to tailor patterns to the particular domain.

$R_4$ : **Processing Abstraction.** Smart Apps have access to both, real-time data and history data. Accordingly, private patterns have to be concealed from both, data stream and database accesses. Any combination of real-time and history data has to be considered as well.

$R_5$ : **Quality-preserving.** The service quality of a Smart App should not be impaired by concealing private patterns. Public patterns still have to be recognized as much as possible, as measured by the amount of false positives (pattern is detected, but did not actually occur) and false negatives (pattern occurred, but is not detected).

## III. Related Work

In the following, we discuss different privacy control mechanisms for Smart Apps. We consider approaches for real-time data (i. e., approaches for data stream systems), approaches for history data (i. e., approaches for databases), as well as privacy mechanisms for Smart Devices in general. In particular, we assess the impact of these approaches on context recognition. The discussed approaches constitute representative samples for the many similar approaches.

**Privacy Approaches for Data Stream Systems.** *AC-Stream* [16] provides fine-grained access control over data streams. Its policies specify which attributes can be queried. Constraints restrict data access further, e. g., by granting access to aggregated data, only. However, access to information is controlled only at the level of attributes, which limits context recognition heavily. Kim et al. [17] introduce a delay-free anonymization mechanism based on *l-diversity*. That is, this approach also operates on attributes, only. *SKY* [18] enforces *k-anonymity* on streaming data. By pooling multiple similar users, private data cannot be associated to an individual. Wayne [19] introduces a similar approach in which *differential privacy* is applied. However, such approaches are not applicable in many use cases. For instance in the health domain, a physician needs to know exactly which health record belongs to which patient. Assam et al. [20] introduce a refined differential privacy approach by considering more contextual data. Thereby, their approach provides strong privacy even for a single object. However, this approach works only on spatio-temporal data. *StreamShield* [21] adds processing rules as metadata to each data set. These rules are executed in each streaming operator—i. e., each operator monitors itself. However, this approach assumes a trustworthy execution environment. Moreover, it is also entirely attribute-based which has a negative impact on context recognition. *DEFCon* [22] prevents illegal message flows between the processing units of an event processing system. This does not restrict the context recognition within a Smart App since it only takes data leaks within an app into consideration. However, this also means that it contributes little in ensuring privacy.

**Privacy Approaches for Databases.** Wang et al. [23] propose a solution for fine-grained access control in relational databases. In their approach, certain attributes can be filtered out for each resulting tuple. With respect to context recognition, this has the same shortcomings as attribute-based privacy approaches for data stream systems. Silva et al. [24] introduce a differential privacy approach for RDF databases. This approach thus enables accurate analyzes on the stored data are possible, while no information about individuals are revealed. However, this is also not applicable to Smart Apps, which provide user-related services (e. g., eHealth apps). Olumofin and Goldberg use *Dynamic SQL* in order to remove private information from the results [25]. Query rewriting technique in this approach is not capable of concealing complex patterns. The *SDC* [26] and *CURATOR* [27] are secure data containers for Smart Devices. A relational database respectively an object store is enhanced by security mechanisms to obfuscate or remove private data from query results. Its access control takes effect at the tuple level. Yet, both approaches operate on attributes rather than patterns.

**Privacy Approaches for Smart Devices.** Chitkara et al. [28] address the problem that a lot of private data is gathered by third-party libraries. These libraries—and thus their data—are shared among several apps. Therefore their approach does not grant particular apps access to private data. Instead, their system tracks any data access attempt involving private information at runtime and grants or denies it considering all currently running apps or services. The *PMP* (Privacy Management for Mobile Platforms) [29] is a fine-grained privacy system for Smart Devices. Its policy rules take the user's current context into account, e. g., to apply different rules at workplace and at home. *PolEnA* [30] uses bytecode instrumentation to add monitoring components to an app. These components decide at runtime whether or not a requested Android Permission is granted. *TrUbi* [31] introduces privacy domains. These domains represent specific use cases (e. g., usage of a private laptop for

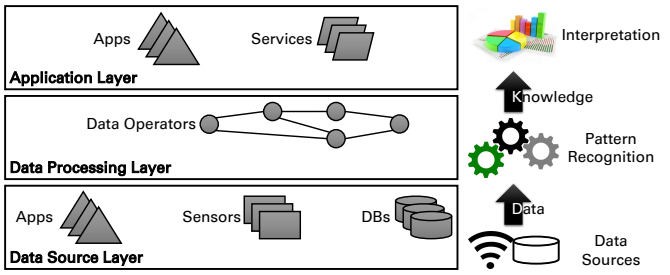


Figure 2. Model Architecture for Smart Apps.

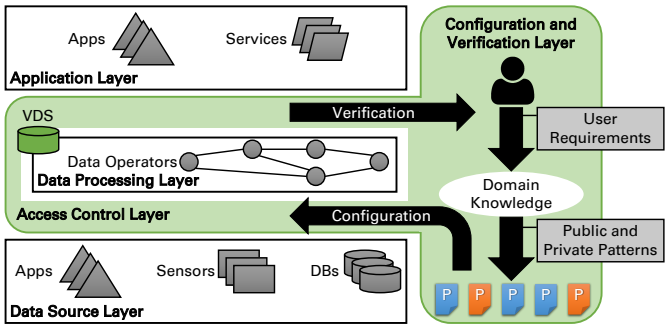


Figure 3. The PATRON Architecture for Smart Apps.

an exam). For each use case, other data access restrictions are applied (e. g., access to the Internet is prohibited during exams). All of these approaches operate on the level of attributes and therefore impair context recognition significantly.

## IV. The PATRON Approach

Since the currently existing privacy systems for Smart Apps impair the service quality substantially due to their focus on attribute-based access control, we introduce PATRON, a novel pattern-based privacy approach. To this end, we need to keep in mind the Smart App architecture (see Figure 2): The *Data Source Layer* provides access to various data sources such as sensors, databases, or other Smart Apps. This input data is then processed in the *Data Processing Layer*. A network of various data operators preprocesses the data progressively, as required by the Smart App. Intermediate data or even the stream of real-time data as a whole can be fed back to the Data Source Layer, e. g., to preserve as data history. Smart App Components which interact with users are part of the *Application Layer*.

In other words, data origins, knowledge acquisition, and knowledge processing are strictly separated from each other. From a privacy perspective this entails that a pattern-based system such as PATRON has to be wrapped around the Data Processing Layer in order to know any incoming raw data and where the preprocessed data is forwarded to (see Figure 3). Basically, PATRON consists of two additional layers: A horizontal *Access Control Layer* and a vertical *Configuration and Verification Layer*. In PATRON, a user describes his or her privacy requirements in natural language. Domain experts translate these requirements into public and private patterns. In order to model these patterns and the derivable knowledge from given data sources, e. g., ACCESSORS [32] can be used. This translation is made semi-automatically, i. e., the domain experts are supported by tools, indicating potential privacy threats. According to the resulting patterns a configuration for the Access Control Layer is compiled. To this end, a quality metric is applied in order to find the best configuration, i. e., the one which reveals as many public patterns as possible while concealing all private patterns.

The Access Control Layer monitors the input streams of the data operators while also considering the previous knowledge of an operator. If a private pattern can be derived then, the Access Control Layer intervenes. To ensure privacy, different techniques are available e. g., changing the sequence of events or dropping events. It is noteworthy that it is not adequate to look at the output of the Data Processing Layer and remove private patterns from it. The layer could secretly integrate indicators that a certain private pattern has occurred into seemingly harmless patterns.

To persist intermediate data or history data, *Virtual Data Stores* (VDS) are used. A VDS can be used like a normal database but it provides additional security features. These features ensure that no private patterns are revealed. The VDSs are tightly integrated into the Access Control Layer in order to ensure that the layer always knows all data accessible by an operator. Otherwise, an operator could derive private patterns by combining real-time and history data.

The results of the Data Processing Layer are sent back to the Configuration and Verification Layer to verify that no private patterns were revealed. In the following, we focus on the Access Control Layer. For details on the Configuration and Verification Layer, please refer to [33].

**Privacy Mechanism for Streaming Data.** Different techniques can be used to conceal privacy-critical patterns in data streams. These techniques include suppression, obfuscation, and reordering of sensor events. These techniques are illustrated in the following example: Assuming there is a data stream, transmitting the followimg sequence of events: $A \rightarrow B \rightarrow A \rightarrow C$. To conceal the private pattern $B \rightarrow A$, *suppression* drops the second $A$ event (or the $B$ event, respectively). The resulting data steam looks like this: $A \rightarrow B \rightarrow C$. *Obfuscation* modifies an event to disguise it as another event. One possible solution looks like this: $A \rightarrow B \rightarrow A' \rightarrow C$. Finally, *reordering* changes the sequence of the events in the data stream, e. g., $A \rightarrow A \rightarrow B \rightarrow C$.

The choice on one of these techniques in a given use case has to be decided based on two criteria: a) Concealing private patterns must impair the recognition of public patterns as little as possible (see the Quality Metric Section for more information). b) An attacker might have background knowledge about the data stream and this should also be considered while concealing. For instance, if a probability distribution suggests that every second event in a data stream is $A$, none of the solutions proposed above is advisable. An attacker notices a manipulation of the stream and therefore s/he is able to draw conclusions. This background knowledge is different for every

use case and has to be taken into account by the domain experts while configuring for PATRON.

**Privacy Mechanism for History Data.** VDSs persist both, a history of raw sensor data as well as any intermediate data accruing in the Data Processing Layer. To protect these potentially sensitive data, various techniques are applied in the VDSs. First of all, all stored data is completely encrypted and only the VDS has the key. Therefore, third-parties cannot read the data if the VDS does not authorize the access. This approach also enables a reliable deletion of the data. If the Data Processing Layer is compromised, the VDS can destroy the key, making all stored data unreadable.

A VDS apply several techniques to conceal private patterns during query processing. These techniques are divided into two categories: On the one hand, the concealing can be enforced *a priori* by rewriting incoming queries. By adding selection operators, the scope of the returned tuples can be restricted (e. g., only diabetes diary entries of a certain patient are returned) whereas projection operators restrain the number of attributes per returned tuples (e. g., only the blood sugar level is returned for each diabetes diary entry). On the other hand, the result set can also be concealed *aposteriori*. This enables to apply filter and obfuscation operators as well as timestamp modifiers in order to suppress private patterns. Similar to the privacy mechanism for streaming data, the experts have to select the best approach case-by-case based on the respective application domain.

**Quality Metric.** PATRON's quality metric rates each configuration in order to determine the best one, i. e., the configuration which provides the best service quality and conceals privacy-critical pattern. A PATRON configuration specifies which privacy mechanism has to be employed to both, streaming data and history data.

The quality metric takes three aspects into account: The total number of recognized public patterns has a positive influence on the service quality, while the total number of false positives—i. e., falsely recognized public patterns—has a negative influence. False negatives are already considered by the public pattern aspect—a false negative is synonymous with a not recognized public pattern. If a configuration reveals inadvertently a private pattern, the service quality is not deteriorated. Yet, the quality metric also considers such privacy breaches by introducing a penalty weight for this purpose.

$$\begin{aligned} QM = &\sum_i public\_pattern_i \quad * w_{pub\_i} \\ &- \sum_j false\_positive_j \quad * w_{pub\_j} * p \\ &- \sum_k private\_pattern_k * w_{priv\_k} \end{aligned} \tag{1}$$

Equation (1) shows a simplified representation of PATRON's quality metric. As one can see, each of the three components can be individually weighted. As a result, certain public patterns can be declared as comparitively important or private patterns as less privacy-critical. In the case of false positives, an additional penalty weight can be specified. For instance, in the eHealth domain, wrong diagnoses might have dire consequences and therefore have to be ruled out. Theses weight parameters are set by the domain experts when they translate a user's privacy requirements into PATRON patterns. In this way, PATRON's configuration can be adapted even better to the user's needs and the experienced service quality can be maximized.

Table I. PATRON's Compliance with the Requirements.

| Requirement | Fulfillment in PATRON | Experienced Impact |
|---|---|---|
| $R_1$ | Various techniques to conceal private patterns and reveal public patterns | Improvement of context recognition as the accessible raw data is maximized |
| $R_2$ | Focus on event sequences instead of data sources | Any kind of data source is protected out of the box |
| $R_3$ | Involvement of domain experts in pattern generation and configuration process | Privacy mechanisms tailored to requirements and application context |
| $R_4$ | Privacy mechanisms for streaming and history data | Privacy protection for both, real-time and history data |
| $R_5$ | Quality metric rates PATRON configurations | Least possible impairment of service quality |

## V. Assessment

To assess whether PATRON is suitable as a private system for Smart Apps, the requirements specification postulated in Section II is applied. This assessment also analyzes the experienced impact on behalf of the user.

PATRON is designed from scratch towards a *pattern-based* processing approach. Accordingly, private patterns can be defined, which must be concelaed at all costs. Thus, all events from which no such pattern can be derived are forwarded to the Data Processing Layer without compromising its accuracy. In this way, public patterns can still be recognized. Furthermore, PATRON introduces a *source abstraction*, i. e., it abstracts completely from concrete data sources. Private patterns are defined without having to specify from which sources the events that form the pattern occur. Thereby the defined private patterns are automatically applied to any currently available data source. Also *domain abstraction* is considered in PATRON. Domain experts translate user-defined privacy requirements into domain-specific patterns. These experts have insights into Smart Apps in their respective domain, including which data sources are involved and which knowledge can be derived from them. As common users do not have such a background knowledge, this is the only way to ensure that all privacy requirements are met. By introducing privacy mechanisms for both, streaming data and history data, PATRON implements a *processing abstraction*. That way it is ensured that Smart Apps cannot accumulate real-time and history data in order to draw conclusions about occurrences of private patterns. Finally, PATRON is also *quality-preserving*. Based on a quality metric the best PATRON configuration is selected. The quality of a configuration is defined by how well it conceals private patterns and reveals public ones, thereby having the least

possible impairment of public pattern recognition. Table I summarizes the key findings of this assessment.

**Lessons Learned.**

§1 Privacy and service quality are not mutually exclusive.

§2 Attribute-based privacy approaches are too restrictive for smart apps.

§3 Attribute-based privacy rules overwhelm common users.

§4 Pattern-based privacy approaches contribute to improve context recognition.

§5 A privacy system for smart apps has to consider both, streaming data and history data.

§6 PATRON conceals *private patterns* completely while *public patterns* are revealed as good as possible.

§7 A quality metric selects the concealing technique which maximizes the experienced service quality.

## VI. Conclusion

Smart Devices are becoming increasingly popular due to the unique situation-tailored services offered by Smart Apps. To this end, such apps require access to a lot of different types of data about the user. From these data Smart Apps derive comprehensive knowledge. Here users wish to benefit from this knowledge in terms of a better service quality, while still wanting to protect their privacy i. e., an app must not be able to derive certain private information. Current privacy approaches are not only incomprehensible to users but also far too restrictive. By blocking certain attributes completely, these approaches impair service quality significantly as apps cannot gather the required knowledge reliably. On that account, we introduce with PATRON a novel pattern-based privacy approach for Smart Apps. In PATRON a user describes his or her privacy requirements in natural language and experts translate them into two types of knowledge patterns—a knowledge pattern is a sequence of events. Public patterns should be detectable by Smart Apps while private patterns must be concealed. To suppress private patterns, PATRON uses various techniques. It selects the one with the smallest impact on service quality. PATRON operates on both, real-time data and history data. Due to these features, PATRON helps to improve context recognition and thus service quality.

## Acknowledgment

The PATRON research project is commissioned by the Baden-Württemberg Stiftung GmbH. The authors would like to thank the BW-Stiftung for the funding of PATRON.

## References

[1] M. Weiser, "The Computer for the 21st Century," *Scientific American*, vol. 265, no. 3, pp. 94–104, 1991.

[2] A. K. Dey, "Understanding and Using Context," *Personal and Ubiquitous Computing*, vol. 5, no. 1, pp. 4–7, 2001.

[3] D. Siewiorek, "Generation Smartphone," *IEEE Spectrum*, vol. 49, no. 9, pp. 54–58, 2012.

[4] C. Gröger *et al.*, "A mobile dashboard for analytics-based information provisioning on the shop floor," *International Journal of Computer Integrated Manufacturing*, vol. 29, no. 12, pp. 1335–1354, 2016.

[5] C. Buck *et al.*, "Mobile Consumer Apps: Big Data Brother is Watching You," *Marketing Review St. Gallen*, vol. 31, no. 1, pp. 26–35, 2014.

[6] G. Suarez-Tangil *et al.*, "Evolution, Detection and Analysis of Malware for Smart Devices," *IEEE Communications Surveys & Tutorials*, vol. 16, no. 2, pp. 961–987, 2014.

[7] L. Ohno-Machado *et al.*, "Protecting patient privacy by quantifiable control of disclosures in disseminated databases," *International Journal of Medical Informatics*, vol. 73, no. 7, pp. 599–606, 2004.

[8] A. P. Felt *et al.*, "Android Permissions: User Attention, Comprehension, and Behavior," in *SOUPS '12*, 2012.

[9] C. Stach *et al.*, "PATRON – Datenschutz in Datenstromverarbeitungssystemen," in *INFORMATIK '17*, (in German), 2017.

[10] M. Knöll, "Urban health games. Collaborative, expressive & reflective," Ph.D. dissertation, University of Stuttgart, 2012.

[11] C. Stach, "Secure Candy Castle — A Prototype for Privacy-Aware mHealth Apps," in *MDM '16*, 2016.

[12] C. Xu *et al.*, "Finger-writing with Smartwatch: A Case for Finger and Hand Gesture Recognition Using Smartwatch," in *HotMobile '15*, 2015.

[13] J. A. Tamada *et al.*, "Keeping Watch on Glucose," *IEEE Spectrum*, vol. 39, no. 4, pp. 52–57, 2002.

[14] R. Almaghrabi *et al.*, "A Novel Method for Measuring Nutrition Intake Based on Food Image," in *I2MTC '12*, 2012.

[15] Y. He *et al.*, "On the Complexity of Privacy-preserving Complex Event Processing," in *PODS '11*, 2011.

[16] J. Cao *et al.*, "ACStream: Enforcing Access Control over Data Streams," in *ICDE '09*, 2009.

[17] S. Kim *et al.*, "A Framework to Preserve the Privacy of Electronic Health Data Streams," *Journal of Biomedical Informatics*, vol. 50, pp. 95–106, 2014.

[18] J. Li *et al.*, "Anonymizing Streaming Data for Privacy Protection," in *ICDE '08*, 2008.

[19] L. Waye, "Privacy Integrated Data Stream Queries," in *PSP '14*, 2014.

[20] R. Assam *et al.*, "Differential Private Trajectory Protection of Moving Objects," in *IWGS '12*, 2012.

[21] R. V. Nehme *et al.*, "StreamShield: A Stream-centric Approach Towards Security and Privacy in Data Stream Environments," in *SIGMOD '09*, 2009.

[22] M. Migliavacca *et al.*, "DEFCON: High-performance Event Processing with Information Security," in *USENIXATC '10*, 2010.

[23] Q. Wang *et al.*, "On the Correctness Criteria of Fine-grained Access Control in Relational Databases," in *VLDB '07*, 2007.

[24] R. R. C. Silva *et al.*, "A Differentially Private Approach for Querying RDF Data of Social Networks," in *IDEAS '17*, 2017.

[25] F. Olumofin and I. Goldberg, "Privacy-Preserving Queries over Relational Databases," in *PETS '10*, 2010.

[26] C. Stach and B. Mitschang, "The Secure Data Container: An Approach to Harmonize Data Sharing with Information Security," in *MDM '16*, 2016.

[27] ——, "CURATOR — A Secure Shared Object Store: Design, Implementation, and Evaluation of a Manageable, Secure, and Performant Data Exchange Mechanism for Smart Devices," in *SAC '18*, 2018.

[28] S. Chitkara *et al.*, "Does This App Really Need My Location?: Context-Aware Privacy Management for Smartphones," *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, vol. 1, no. 3, 42:1–42:22, 2017.

[29] C. Stach and B. Mitschang, "Privacy Management for Mobile Platforms – A Review of Concepts and Approaches," in *MDM '13*, 2013.

[30] G. Costa *et al.*, "PolEnA: Enforcing Fine-grained Permission Policies in Android," in *SAFECOMP '17*, 2017.

[31] M. B. Costa *et al.*, "TrUbi: A System for Dynamically Constraining Mobile Devices Within Restrictive Usage Scenarios," in *Mobihoc '17*, 2017.

[32] C. Stach and B. Mitschang, "ACCESSORS: A Data-Centric Permission Model for the Internet of Things," in *ICISSP '18*, 2018.

[33] K. Mindermann *et al.*, "Exploratory Study of the Privacy Extension for System Theoretic Process Analysis (STPA-Priv) to elicit Privacy Risks in eHealth," in *ESPRE '17*, 2017.