Palpable Privacy through Declarative Information Flows Tracking for Smart Buildings

François Lesueur*, Sabina Surdu*, Romuald Thion†, Yann Gripay* and Meriam Ben Ghorbel-Talbi*
*INSA-Lyon, CNRS, LIRIS, UMR5205, F-69621 France
†Université Lyon 1, CNRS, LIRIS, UMR5205, F-69622 France
Email: {firstname.lastname}@liris.cnrs.fr

***Abstract*—Smart buildings are more and more common due to recent technological advances. They promise to improve users' lives, but they are packed with sensors that gather user related data, fueling ever increasing privacy infringement suspicions. Captured data usually takes the form of dynamic streams, hence such buildings can naturally be programmed using Data Stream Management Systems (DSMSs) that execute long-running queries on data flowing from sensors. In this paper we address the problem of the dissemination control of private data, encountered in smart buildings. We introduce Tuple-Based Access Control (TBAC), a novel access control model that tracks sensor information flows in a DSMS. We provide users with the ability to enforce easy-to-understand, intuitive security policies on sensor-produced data. When such data are combined by queries in the system, so are their security policies, hence data access control is disseminated throughout the system. We argue that such a model is mandatory to ease the acceptance of smart buildings. Nevertheless, TBAC can also be relevant to other scenarios involving dissemination of aggregable private data.**

I. Introduction

Smart buildings are more and more common due to recent advances in sensor networks, data gathering and data analysis technology. They promise to improve users' lives, offering comfort, greenness, and a string of innovative features like regulating heating and cooling in an efficient way, while automatically adapting to users' needs. But all this smartness comes with a price. Such buildings are packed with arrays of sensors and actuators, which pose as many threats to user privacy. Therefore, people are often reluctant to smart buildings, fearing being spied by their boss or colleagues, since gathered and observed data are personal (e.g., door opening, time of arrival). To allay security and privacy fears, these data need to be firmly protected. To this end, we aim at giving users the control over how data that concerns them can be accessed by other actors in the system. We want to provide users with the ability to enforce easy-to-understand, intuitive security properties on data, which propagate throughout the system. We believe that security and privacy are mandatory for the development of smart buildings.

Smart buildings contain sensors that provide data streams and actuators that offer invokable functionalities. A smart building may for instance provide the following sensors for every office: door opening, room presence, light level, temperature; and the following actuators: heating, cooling, light and shutter switches. All sensor data can be collected by a server which, in turn, commands actuators according to its program. The following actions can be triggered by flowing data or at specified time instants: on room presence, if the light level is below a certain threshold, switch the light bulb on; on room presence, activate the heating system to 21°C, on absence, set it to 19°C; at the end of each week, send the *sum* of the presence hours to supervisors. In this scenario, the smart building can improve user comfort (automatic actions), reduce the carbon footprint of the company (switching unneeded cooling systems) or ease management (presence hours of employees). However, such setup naturally worries employees about their privacy. Without privacy protection, people with access to the system can obtain precise working habits, break intervals or arrival times which can lead to unacceptable intrusion into users' lives. Actually, supervisors only need data aggregated at a weekly granularity to check worked hours and not the precise arrival or departure times.

To this aim, the contribution of this paper is Tuple-Based Access Control (TBAC), a novel access control model that tracks information flows among data streams. TBAC tackles some of the privacy issues encountered in smart buildings, i.e., the dissemination control of private data. With respect to alternative approaches, TBAC presents two innovative features. First, users only need to express access rights over sensors targeting them, choose at which time/space granularity these data can be declassified and authorize different granularities to different readers, hence TBAC provides a sense of tangible security to users, which is critical in smart buildings. Second, access rights are associated to data following the *sticky policy paradigm*, propagated and combined throughout the system to guarantee that declassification of combined data respects every original rule set on sensor-produced data. Both the dissemination control offered by the TBAC model and its tangible aspect can lead to a better acceptance of these buildings, while not strictly limited to this scenario: more generally, dissemination of aggregatable private data or e-health are envisioned.

In this contribution, TBAC is integrated into the SoCQ system [1] which is a Service-Oriented Data Stream Management Systems (DSMSs), improved DBMSs that can execute continuous queries on dynamic data streams. SoCQ transparently interacts with streams and functionalities, by hiding imperative code behind declarative interfaces in an SQL-like language. The SoCQ system has been used in [2] to program a scenario very similar to smart buildings, encompassing sensors and actuators and thus it is chosen as an enabling technology in this paper (Section III).

In Section II, we discuss the related work dealing with privacy in smart building and dissemination control. Then, in Section III, we describe how to program a smart building using a declarative approach with the SoCQ DSMS. In Section IV, we present the TBAC model and we show how to integrate it into the SoCQ system. In Section V, we describe a policy administration GUI to show how users can specify their privacy policy. Finally, in Section VI, we propose some future work.

In this paper, we make the following assumptions. First, the DataBase Admin is trusted (the DBA problem is orthogonal). Then, data are currently stored in a trusted server without cryptographic protection (future improvement).

II. Related Work

Since we propose to use flow control to protect privacy in smart buildings, we first describe related work on privacy in smart buildings and we then discuss previous propositions on generic-purpose flow control.

A. Privacy in Smart Buildings

The privacy issues have been widely investigated in the literature and many obfuscation strategies have been proposed in order to keep personal informations confidential and to prevent from malicious invasion of privacy. As we stated previously, smart buildings are designed to exploit personal informations to give personalized services to users. Moreover, users do want to disclose some of their personal information to better collaborate with their colleagues. Hence, a balance between privacy and utility must be preserved and it is mandatory to provide means to users to specify their own privacy policy. We discuss in the following related work on the privacy issue in smart environments.

In [3] authors propose an access control mechanism for the *Solar* system, an event-based context distribution infrastructure. The proposed approach is based on a conservative information-flow model where end-users can express discretionary relaxation of the resulting Access Control List (ACL). Each event is tagged with an ACL derived automatically from the ACL on events that contributed to its production. However, to release constraints on a data inside the system, users need to express exactly on which operator in the system this declassification happens and thus need a global view and understanding of the global workflow, which seems hard to require from end-users. Moreover, although controlled by the end-user, this declassification is prone to information leakage if the user does not fully understand all the private data which can arrive at this point. This is contrary to our current work where users only need to specify properties to be validated on original streams and do not need any insight on the global workflow.

In [4] authors present a policy language based on the metaphor of physical walls, called *virtual walls*, using the *Solar* system as sensing infrastructure. They introduce the concept of personal footprints, a various classes of events containing details about users' activities, the timestamp and the set of sensors where the footprint originates. End-users are allowed to regulate access to their personal footprints by specifying virtual walls around a given place, e.g. a room or a building's floor. Each virtual wall applies to a set of users and has transparency (*transparent, translucent and opaque*). Here authors focus on how users can specify their privacy requirements and how to help them to understand it, by giving an intuitive policy abstraction, like for instance the circle metaphor in the Google+ social network. The TBAC model is at a lower abstraction level, thus the virtual walls approach may be used to simplify the specification of large policies by providing an intuitive metaphor on top of TBAC.

Confab [5] provides a framework for ubiquitous computing applications, where personal informations are captured, stored and processed on the end-user's computer as *infospaces*. Tuples are used to represent users contexts, that can be static, as name and email, or dynamic, as location and activity. Each tuple can have a privacy tag which describes the end-user's privacy preferences. These tag can specify *TimeToLive*, i.e. how long data should be retained; *MaxNumSightings*, i.e. the maximum number of previous values that should be retained; *Notify*, i.e. an address to send notifications of second use to; *GarbageCollect*, i.e. additional hints on when the data should be deleted. Privacy preferences are specified by end-users when a client application makes a request to an infospace. These preferences are then stored and used later for the associated service. This work is similar to ours in the sense that they allow users to specify their own privacy policy and stick this policy with users data to enforce it. However, they do not focus on the combination and aggregation problems, which are at the center of our work.

In this paper, we explore a new access control model using a declarative DSMS framework to program smart buildings. In fact, tracking information flows in a declarative program offers different possibilities, compared to the previously cited papers which are all using imperative programs. First, the higher level semantics of declarative operators allows a better understanding of programs which tend to be more concise and clearer. Then, declarative programming is intensively used by developers today, e.g.,

the prominence of SQL in DBMS, so we merely tailor our approach to fit current development usages. The database layer has also shown over the years to be a good abstraction for security features.

B. Flow Control in DSMS

Lattice-Based or Mandatory Access Control, generalizing the confidentiality model from Bell and LaPadula [6], can be used to control information flows in a DSMS. Each data has a label, combining data creates new data with a more restricted label and users can only access data with a label equal or inferior to their clearance level. Such models guarantee that information cannot be accessed by users who had no access to original data. However, they are too restrictive in our context as, for instance, a mean of a set of temperatures would be labeled as confidential as soon as one, possibly in one thousand, is confidential. Rendering these systems more usable requires declassification: a highly privileged process can downgrade the label of some composite data to make them more accessible, e.g., to set the mean temperature as public. However, declassification is executed on composite data with no history on contributing data, yielding potential errors and unexpected information leakage. Moreover, data creators cannot control this process, what data it declassifies and how.

Dissemination Control (also called Originator Control) models [7] tackle uncontrolled declassification. Users attach security policies to the data they produce. The dissemination of these data must obey expressed constraints, while still providing some guarantees on information flows. These constraints may express a path of dissemination, some intermediary controls, etc. For instance, [8] proposes a *Policy, Enforcement, Implementation* model for secure information sharing. This paper encompasses a wide area from policy to user authentication and thus only provides a bird eye's view on the topic of declassification. The combination and aggregation problems, which are at the center of our work, are however not addressed.

In [9] authors propose a stream-centric approach where security restrictions are streamed together with the data, as Security Punctuations (SPs). SPs are meta-data that specify who has access rights to which streaming data. These meta-data are specified by end-users and then combined by the DSMS server with the server-side policies in order to allow organizations to enforce their own policies. To address the combination and aggregation issues, authors propose to extend stream algebra to become security-aware. Authors advocate their proposition as an access control enforcement mechanism rather than a new control model, hence our TBAC model is complementary and may in fact be implemented using SPs.

III. Enabling Technology: Declarative Smart Building

The SoCQ system is a DSMS that provides a relational abstraction and a SQL-like query language on top of distributed services and can be used in a smart building environment. We briefly describe this enabling technology which is the underlying block on which we add dissemination control mechanism to provide a privacy-aware smart buildings infrastructure. Note that an other declarative DSMS might be used as well.

For each type of sensor, a relational table is created. This table is tightened to sensors and actuators through *binding patterns*, which link declarative statements to imperative code and protocol. The content of the table is a structured stream of data tuples from sensors and SoCQ can then execute SQL-like queries on sensor data. For each type of actuator, a table is also created. In a symmetric way, binding patterns link tuples inserted by the DSMS into this relation to imperative code and protocol. Tuple insertions trigger imperative code which can, in turn, activate actuators. Imperative code is minimalistic in our scenario, doing very simple tasks. All the computation and combination of information flows is done inside SoCQ, at the declarative level.

The SoCQ system is used in the SoCQ4Home project[1], a smart building management project. An experimentation platform is installed in the authors' building at LyonTech Campus in France : 50 wireless sensors (temperature, CO_2, humidity, luminosity, presence, doors) are deployed in about 20 rooms (offices, common room, teaching rooms). The smart building program is implemented through SoCQ continuous queries that subscribe to streams (e.g., temperature streams from sensors), manipulate data, or invoke actuator functionalities (e.g., activate heater).

The global workflow is illustrated in Figure 1, where the so named *CS smart building* contains presence and temperature sensors, and temperature heater actuators. A background continuous query `CQ1` monitors tuples from presence sensors streams and, based on presence (ON) or absence (OFF), sets the temperature of the corresponding area accordingly, through a temperature heater actuator.

Moreover, data from the system can also be queried by a Building Manager (BM) who wants to see aggregated temperatures based on temporal and spatial criteria (e.g., query `CQ2`). In this query, the BM should not have access to data in red rows, i.e., data that are privacy invasive, but only to aggregated data. For instance, the BM cannot access a row that gives the average temperature for one office on a per-minute basis as being too specific. By contrast, if either the area or the time interval increases, the data are no sensitive anymore, so the BM can access them. In our proposition, this access decision is entirely controlled by the users to whom sensors relate.

IV. Tuple-Based Access Control

In this section, we present the TBAC model on top of the SoCQ system. Our approach is based on policies expressed on sensors and on subsequent combinations of

[1] http://liris.cnrs.fr/socq4home/

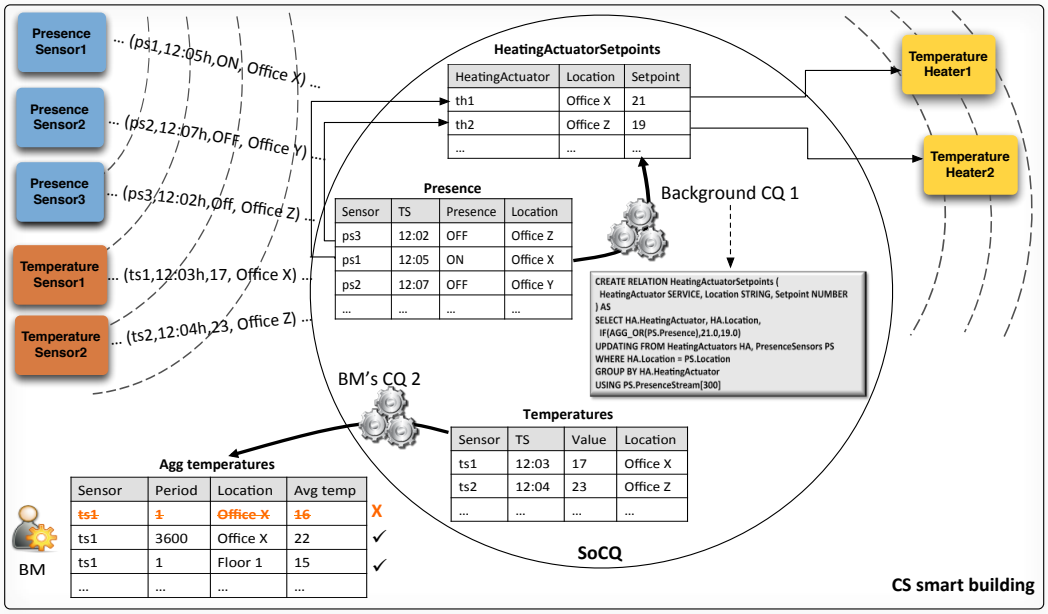

Fig. 1. Global workflow

policies throughout the SOCQ system. First, users tag the sensors they are authoritative for with a security policy, named *s-tags*, defining who has access to data produced by these sensors and at which aggregation level. Second, the output data stream of a sensor is duplicated at each aggregation level and the resulting tuples are tagged with the sensor's set of authorized user. Third, when tuples are combined by SoCQ queries in order to provide smart features, their authorized users are combined to tag the output. Finally, when a user like the BM tries to access some data, she obtains the result only if she satisfies all the policies stated by original users.

A. Security Policies

In smart-metering in particular, a privacy-aware security policy should ensure users cannot execute queries that are too intrusive, based on spatial or temporal criteria. E.g., in the motivating scenario, one does not want the BM to access their precise arrival and departure time, but only office spent time on a weekly basis. Likewise, BMs should be prevented from deducing when an employee is present based on temperatures variations (door opening, etc). Instead, BMs may only obtain values aggregated on space or time.

The key ingredient of these requirements is the granularity of the aggregation operators on spatial and temporal dimensions. The set K of all s-tags is defined as a set of atomic policies, each one defines a minimum granularity at which a data must be aggregated before disclosure. We note by T the lattice of time aggregation levels, S the lattice of space aggregation levels, OP the required aggregation operations (e.g., min, max, average, median, count) and U the set of users. The definition of T and S as lattices is standard in multidimensional datawarehouses to build the lattice of all possible aggregation levels, named *lattice of cuboids*. For instance the triple $(\text{week}, \text{office}, \text{avg}) \in T \times S \times Op$ allows the use of the sensor's data by queries that are coarser than a weekly report on offices' temperatures. We note by $\mathcal{P}$ the powerset, by $+$ the disjoint union and by $\perp$ a special element that captures the lack of any constraint and we formally define K:

$$K = \mathcal{P}((\{\perp\} + T \times S \times OP) \times \mathcal{P}(U))$$

For example, consider the s-tag $k = \{k_0, k_1\}$ constructed from two atomic policies $k_0 = ((\text{week}, \text{office}, \text{avg}), \{BM, BMa\})$ and $k_1 = (\perp, \{Alice\})$. This policy is attached to the Temperature Sensor 1 (TS1). This s-tag states that the building manager BM and his/her adjoint BMa are authorized to query the stream only for a weekly report. However, the user named *Alice* can compute everything using data emitted by TS1 without constraints.

When a SoCQ query is executed, for instance computing the weekly temperature report (`CQ2`), it consumes input streams, combines them and produces an output stream whose data must be tagged according to the inputs' s-tags. Security policies are thus propagated throughout the system.

B. Aggregations of sensor streams

With the proposed security policies, access to composite data should be granted only if those data have been

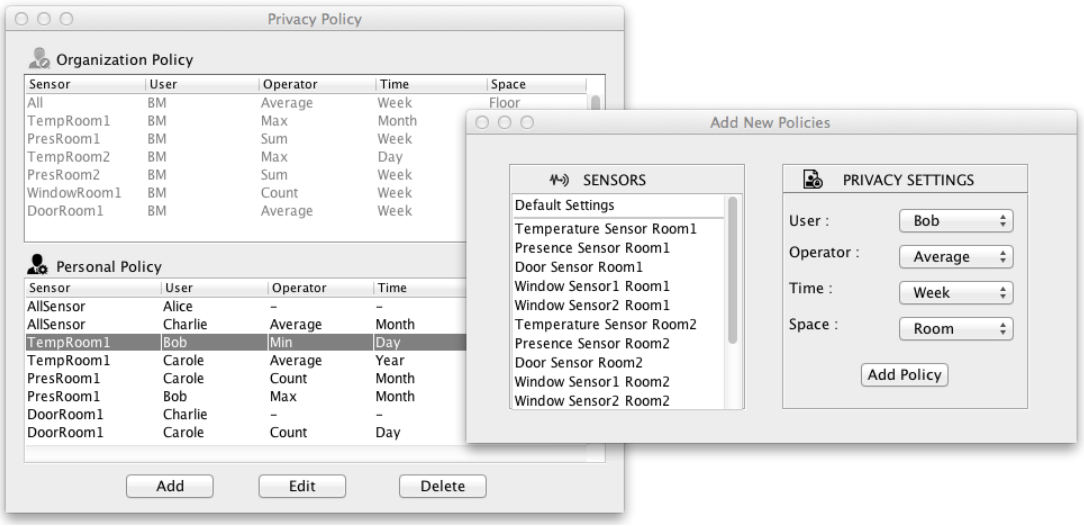

Fig. 2. TBAC-SoCQ Graphical User Interface

aggregated according to all the policies set on original data. However, being able to ask for aggregation queries on raw tuples could lead to information leakage; Consider the following scenario. Alice, in the CS building of Figure 1, tags her temperature sensor TS1 with the k policy. Inside the SoCQ system, a table `Temperatures` off all rooms in the building is generated. The BM could then directly query `Temperatures`, containing only current temperatures, for an average over 7 days of Alice's data. Since there is only one value in the table to average, which is the current temperature in Alice's office, the average would be this precise value, which should not be obtained by the BM. However, the system would validate the fact that data has been averaged over 7 days, since the query required this average, and send the data to the BM. While we could imagine some metrics based on k-anonymity, l-diversity, t-closeness or differential privacy, we propose an alternative and complementary solution.

For each sensor stream entering the system, we duplicate it according to the aggregation levels specified in its s-tags. E.g., for sensor TS1, tagged with $k = \{k_0, k_1\}$, two streams are fed into the SoCQ system: `TS1_Alice`, the raw stream authorized to *Alice* by k_1, and `TS1_week_room_avg`, the stream averaged over a week on a room basis, authorized to BM and BMa by k_0. This duplication can be done automatically by enumerating the constraints in the expressed security policies and then naming the stream in a canonical way. Such views on streams are a logical definition that will be executed only when needed and not materialized systematically. The idea of defining a set of views, one for each aggregation level, and choosing the right one at execution time is well studied in the field of data warehouse and multidimensional analysis. The duplicated views are those whose aggregation levels are coarser than required, e.g., with k_0, the views with aggregation at week or building levels are authorized.

Finally, data tuples stored in the system are only tagged with the set of users allowed to read them, instead of an element of K. For instance, all tuples in `TS1_Alice` are tagged with the singleton $\{Alice\}$ and those in `TS1_week_room_avg` are tagged with $\{BM, BMa\}$. This construction ensures that security policies expressed by users are enforced. The price to pay is denying access to data which could be allowed by the user, but we argue this is a reasonable trade-off between accuracy and privacy.

C. Combination Operators

We need operators to define how to combine set of users. As it is done in the relational model with provenance [10], each SoCQ relational operation has to define how allowed users are combined. One of the remarkable result of the provenance framework of [10] is that only two operators $\otimes$ and $\oplus$ are needed to combine meta-data associated to tuples when base tuples are combined by means of the relational algebra.

Basic SoCQ operations from relational algebra are Select, Project, Join, Rename and Union. Select and Rename are transparent as they do not alter the set of authorized users: renaming rows or picking a subset of data in a stream do not alter tuples. Project and Union can merge several original tuples into the same one. Its set is the *union* of allowed users on original data: access to the composite tuple requires access to *either* of the original ones, formally, $x \oplus y = x \cup y$. Join combines two original tuples into a composite one, whose set of authorized users is the intersection of allowed users on original data: access to the composite tuple requires access to *both* original ones, formally, $x \otimes y = x \cap y$.

D. Access Control Filter

The access control *filter* has to trap every access attempt and to filter whether the request is authorized by the

security policy. In our setting, requests are SoCQ queries that are either ran in the background to trigger actuators or submitted by users. Access must be filtered in both cases.

For instance, a tuple tagged with $\{Alice, Bob\}$ is allowed to be read by both Alice and Bob. This situation corresponds to the case where the tuple can be obtained in different ways, for instance if TS1 and TS2 are two sensors in different offices X and Y allowed to be accessed by Alice and Bob respectively. If both offices are quite warm, both Alice and Bob can read the positive answer to the query "is there any office with temperature higher than 21 °C?". There is only one value, namely *true*, tagged with $\{Alice, Bob\}$. However, if the query is "what are the offices with temperature higher than 21 °C?", Alice will only read Office X tagged with $\{Alice\}$.

For queries triggering an actuator, each actuator is empowered with the credentials of its owner. For instance, the heater actuator HA1 in Alice's office has Alice's credentials. For each table that sets an actuator's setting, the latter is effectively transmitted to the actuator if and only if this actuator belongs to the authorized userset. In the example, HA1 can use TS1's temperature values because of policy k_1 that authorizes Alice.

User queries are naturally empowered with user's credentials. When a user sends a query, the engine picks the view that best matches the query among the possible aggregation levels and then filters the stream on-the-fly, such that only tuples whose s-tags contain the user as a member are kept.

V. Policy specification

Figure 2 describes the user interface for specifying the privacy policy. Users can edit their privacy policy in order to add a new rule, to modify or to delete an existing one. The policy specified by the organization, which is read-only, is also shown on this interface. Although the user is not allowed to edit it, it is part of its contract to show, for instance, the number of presence hours in office averaged by week to its supervisor. Showing this policy makes it *transparent* to the user who is allowed to access which part of its data, which is an interesting privacy property.

To add a policy, a user can specify a default policy that applies to all of its sensors or a more specific one that applies for each sensor that he is authoritative for. To set her privacy preferences, she can specify for each user, which operation is allowed on the generated tuples and at which spatial and temporal aggregation level. The different s-tags specified for the same sensor (organization's policy, user's default policy, user's specific policy) are then combined using the $\oplus$ operator. Moreover, as we mentioned in the previous section, different users can tag the same sensor (shared offices for instance), in which case these s-tags will be combined using the $\otimes$ operator.

VI. Future Work

TBAC is a preliminary sketch of a security model presenting innovative features, like palpable, user-defined privacy, and dissemination control. It grounds on declaratively programming smart buildings, using DSMSs such as SoCQ. We envision a full integration of TBAC in SoCQ. We want to propose more expressive security policies and find alternatives to stream duplication (raw and aggregated), yet provide the same security guarantees. The integration of privacy measure such as k-anonymity, l-diversity or t-closeness into a DSMS is a promising direction to provide privacy settings complementary to s-tags. Storing data in secure personal tokens [11] can also alleviate the DBA trust hypothesis. Furthermore, we plan to enhance the user interface by using an abstraction level in order to ease the users' privacy management. We can ground our work on existing studies such as [4], where different transparency levels are proposed. Thus, instead of specifying details about the allowed operation that can be done on the generated tuples we can use privacy levels. Moreover, the concept of virtual walls can be used to allow users to specify a privacy policy that applies to a given place including a set of sensors.

Acknowledgements

This work has been partially funded by the French ANR KISS project under grant No. ANR-11-INSE-0005.

References

[1] Y. Gripay, F. Laforest, and J. Petit, "A Simple (yet powerful) Algebra for Pervasive Environments," in *EDBT*, 2010.

[2] Y. Gripay, F. Laforest, F. Lesueur, N. Lumineau, J. Petit, V. Scuturici, S. Sebahi, and S. Surdu, "ColisTrack: Testbed for a Pervasive Environment Management System (Demo)," in *EDBT*, 2012.

[3] K. Minami and D. Kotz, "Controlling Access to Pervasive Information in the Solar System," Dartmouth Computer Science, Tech. Rep., 2002.

[4] A. Kapadia, T. Henderson, J. J. Fielding, and D. Kotz, "Virtual Walls: Protecting Digital Privacy in Pervasive Environments," in *Pervasive Computing*, 2007.

[5] J. I. Hong and J. A. Landay, "An Architecture for Privacy-Sensitive Ubiquitous Computing," in *MobiSYS*, 2004.

[6] D. E. Bell and L. J. LaPadula, "Secure Computer Systems: Mathematical Foundations and Model," *MITRE CORP BEDFORD MA*, vol. 1, 1973.

[7] C. McCollum, J. Messing, and L. Notargiacomo, "Beyond the pale of MAC and DAC-defining new forms of access control," *IEEE Computer Society Symposium on Research in Security and Privacy*, 1990.

[8] R. Sandhu, K. Ranganathan, and X. Zhang, "Secure Information Sharing Enabled by Trusted Computing and PEI Models," in *ASIACCS*, 2006.

[9] R. V. Nehme, E. A. Rundensteiner, and E. Bertino, "A Security Punctuation Framework for Enforcing Access Control on Streaming Data," in *ICDE*, 2008.

[10] T. Green, G. Karvounarakis, and V. Tannen, "Provenance Semirings," *PODS*, 2007. [Online]. Available: http://portal.acm.org/ft_gateway.cfm?id=1265535&type=pdf&coll=DL&dl=ACM&CFID=23417009&CFTOKEN=49968894

[11] T. Allard, N. Anciaux, L. Bouganim, Y. Guo, L. L. Folgoc, G. Nguyen, P. Pucheral, I. R. I., Ray, and S. Yin, "Secure Personal Data Servers: A Vision Paper," *VLDB*, vol. 3, no. 1-2, 2010.