Data degradation to enhance privacy for the Ambient Intelligence

CTIT Technical Report

H.J.W. van Heerde
University of Twente
Enschede, The Netherlands
heerdehjw@ewi.utwente.nl

N. Anciaux
INRIA
Rocquencourt, France
Nicolas.Anciaux@inria.fr

ABSTRACT

Increasing research in ubiquitous computing techniques towards the development of an Ambient Intelligence raises issues regarding privacy. To gain the required data needed to enable application in this Ambient Intelligence to offer smart services to users, sensors will monitor users' behavior to fill personal context histories. Those context histories will be stored on database/information systems which we consider as honest: they can be trusted now, but might be subject to attacks in the future. Making this assumption implies that protecting context histories by means of access control might be not enough. To reduce the impact of possible attacks, we propose to use limited retention techniques. In our approach, we present applications a degraded set of data with a retention delay attached to it which matches both application requirements and users privacy wishes. Data degradation can be twofold: the accuracy of context data can be lowered such that the less privacy sensitive parts are retained, and context data can be transformed such that only particular abilities for application remain available. Retention periods can be specified to trigger irreversible removal of the context data from the system.

General Terms

Security, Algorithms

Keywords

Privacy, Ambient Intelligence, Regulating Context Histories

1. INTRODUCTION

Since many years, much research has been done in the field of 'Ambient Intelligence'. Although some definitions are available [17], it is still hard to capture the term 'Ambient Intelligence' in a clear definition. It is easier to define Ambient Intelligence by giving characteristics of it, like *'it consists of a pervasive network of intelligent devices that will cooperatively gather, process and transport information'* [25, 11], or in other words: it consists of a collection of ubiquitous computing techniques. Combine this property with the following and you get an impression of what a *smart* Ambient Intelligence stands for: AmI uses new forms of *natural* interaction with users in order to *'let people live easily in digital environments in which the electronics are sensitive to people's needs, personalized to their requirements, anticipatory of their behavior and responsive to their presence'* [24]. This means that Ambient Intelligence will be everywhere, invisible, will use powerful sensing capabilities and most of all needs a memory [19] in the form of *context histories* to be able to easy peoples' life in making automatic decisions [22].

An important side effect of Ambient Intelligence is, that a smart, anticipating and learning environment will have a great impact on privacy. One of the main difficulties with privacy in the ubiquitous computing, is the way how data is collected. Ubiquitous computing techniques, such as small sensors, active (RFID) badges or cameras equipped with powerful image recognizing algorithms, often collect data when people are not aware of it [18]. In that case it is possible that people could have a comfortable feeling of safety (e.g., when they are in a closed private area such as coffee rooms), but in *reality* they could be monitored by sensors without being aware of it. This leads to asymmetric information: the *owner* of the data (the *donor*) has less information than the *collector* of the data [18]. Xiaodong et al state that the presence of asymmetric information is the heart of the information privacy problem in ubiquitous computing. In environments with significant asymmetry between the information knowledge of *donor* and *collector*, negative side effects as privacy violations are much harder to overcome.

Several techniques have been proposed which let donors of the data specify privacy policies, in order to give control of their data to the owners of that data [28, 6]. Although such policies are rich enough and very useful to let users control who, when, how long and what kind of information can be disclosed to specific applications, enforcing those policies is managed by access control. As stated by Agrawal et al [1], limited retention and limited collection techniques (such as to some extend applied in *P3P* [28]) are highly desirable to prevent large collections of context histories to be disclosed at times when access control fails to resist against attacks on the database system, human mistakes or even malicious database administrators [13].

Companies providing smart services will have an interest in protecting the privacy of their customers in order to keep their market segment. They can not take the risk of violating users' privacy on purpose, and besides that they are pushed by the law to protect the privacy of their customers [12]. This means that servers storing context histories used by those companies can be considered as *honest*: they do their best effort to protect their content and are not voluntarily malicious. In practice however, most of such servers are eventually subject to attacks from the outside

and so can not be trusted forever [8]. Using database/network IDS (Intrusion Detection Systems), the state of an information system in practice can be viewed over time as followed: periods of normal behavior without attacks and problems are interrupted by attacks. With such a behavior, protecting information by means of access control might be not enough. Indeed, when the system is under attack, all data can be stolen and be disclosed. With a retention model, we benefit from the periods of 'normal behavior' to remove useless data and limit the impact of next attacks. The impact is even more limited with limited collection, since only data for which consent has been given will be stored. Only relying on access control mechanisms to protect against not allowed disclosure of data, is in terms of privacy protection not sufficient enough [1, 16].

1.1 Contribution

The main problem as presented in this paper is how to provide limited retention and limited collection. We recall the definitions of those two principles as stated by Agrawal as principles for an Hippocratic Database [1]:

- Limited Collection: The personal information collected shall be limited to the minimum necessary accomplishing the specified purposes.
- Limited Retention: Personal information shall be retained only as long as necessary for the fulfillment of the purposes for which it has been collected.

For particular, mostly static databases containing large datasets with privacy sensitive data (like medical data), anonymization can be used as a limited retention mechanism to prevent disclosure of individual privacy sensitive data [27, 26, 21, 7]. Regarding the Ambient Intelligence, this is a limited solution since anonymization gives not always adequate privacy protection for everyone, and the usability of the data becomes sometimes lower than needed because individual privacy concerns and personal interests are not taken into account. Xiao et al [30] recognize this problem and propose to *personalize* the anonymization of privacy sensitive data. However, to present real, irreversible privacy protection through anonymization or noise insertion leads often to an unacceptable loss of usability regarding smartness for the Ambient Intelligence.

Server side encryption methods can be used to protect data disclosure, but, as with access control, we can not trust the server forever. Hence, in this case encryption is not a solution.

The nature of data used in the Ambient Intelligence and smart environments is different and more dynamic than traditional static data. The amount of smartness of applications is bound to the quantity and quality of the data they can use. How more accurate the data, and how more data has been gathered from a certain individual, the better a smart application can learn from that data without user interaction [10]. How privacy sensitive certain data (for example: location data) is, depends partially on time. For some people, it is acceptable that their current location is visible for everyone, but not acceptable that people can track their whole path in history. Perhaps they find it less privacy sensitive if the exact time stamps belonging to their past locations are degraded to a less accuracy level, or that their exact location is degraded to the area in which they were. Our target is to find the best balance between the quality and quantity of data at the one side, and the privacy sensitivity of the data otherwise by retaining only that data necessary to fulfill applications purposes.

Our contribution is therefor twofold:

1. We provide means to *degrade* and *transform* data in order to collect and retain only that information needed for each individual application to supply their services to the users. Given specified retention period(s), data is only retained in a specific form as long the user gives consent for it.
2. By clearly stating which information is needed and will be stored for how long, users are in control over their data, decreasing the impact of the asymmetric information problems.

The organization of this paper will be as followed. First we present an introduction into techniques for degrading the accuracy of data values specified by *life-cycle policies*. After this we introduce the notice of degrading *abilities* by means of data transformation. We conclude with combining the introduced techniques into *application oriented data degradation*, such that each application will have access to a materialized view which contains only that data authorized by the users to this application.

2. VALUE DEGRADATION

One way to make data less privacy sensitive is to generalize the data to make the data less accurate. By removing parts of the data (e.g., the minutes part from a time value), less data is retained increasing privacy in trade of decreasing usability. However, for some services full accuracy is not required, making degradation acceptable for those applications. Besides that, using value degradation as an fine grained limited retention strategy means that not all data needs to be destroyed after a retention period, making more data available for application while still complying to users' privacy wishes. This can be done in multiple degradation steps.

In this section we provide a formalization of *value degradation* specified by a *life-cycle policies* [2, 29]. For simplicity, we model data as n-tuples (e.g: time, person, context, ...) which can take values in context states, exhibiting a certain level of accuracy specified in a domain generalization graph [15, 27] of that attribute. The generalization graphs form together a cube (see figure 1 in which we show three context dimensions *time*, *id* and *location*), in which each dimension represents the accuracy of an attribute of the original data triplet. A Life-Cycle Policy is a set of transitions between elements (states) of this cube and the events which trigger the transitions.

To be precise, we define a LCP as:

$$LCP = (S, \Sigma, \delta, s_0, s_f) \quad (1)$$

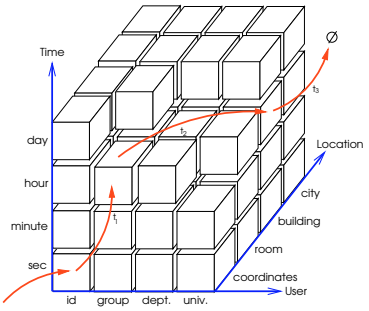

Figure 1: ***The different domain generalization graphs together form a cube. A Life-Cycle Policy is a set of transitions between elements (states) of this cube and the events which trigger the transitions.***

where S is a set of context states written as triplets (t, id, l) ($\emptyset$ is used to denote the 'removal state'), Σ a set of events, δ a set of *transition functions* $S \times \Sigma \rightarrow S$, s_0 the *start state* (also called the *construction state*) and s_f the *final state* of the degradation process, where $s_f = \{\emptyset\}$ indicates removal of the context data from the system. For example:

$S = \{s_0, s_1, s_2, s_f\} = \{(sec, id, url), (hour, group, url)$
$, (hour, uni, category), \emptyset\}$
$\Sigma = \{t_0, t_1, t_2\} = \{$*after 1 hour, after 1 month, after 1 year*$\}$
$\delta(s_0, t_0) = s_1$
$\delta(s_1, t_1) = s_2$
$\delta(s_2, t_2) = s_f$

A LCP can be represented by a *labeled directed acyclic graph*. The nodes of such graph are elements of the set of states Q, the labels are elements of Σ, in such way that an arc from s_i to s_j is labeled a if $s_j = \delta(s_i, a)$. A *DAG* of the LCP of above example is given in figure 2. Also the arrows in figure 1 illustrate transitions between context states corresponding to a (different) LCP.

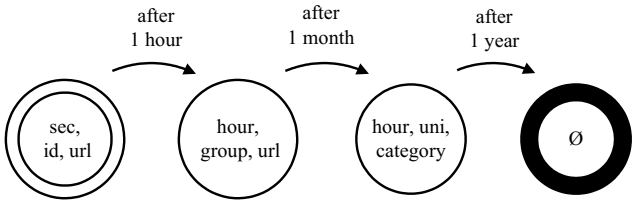

Figure 2: ***Example of a degradation policy noted as a DFA***

3. DEGRADATION OF ABILITIES

In stead of only facilitating natural degradation of values, we also want to support *functional* degradation of *abilities*. In this section we show that traditional database operators (e.g., select, project, join) can be disabled by means of data degradation in opposite to traditional access control techniques (like simply disabling the operators given specified access rights).

To motivate this approach, we slightly enhance the definition of context data:

Context data = event + *link to the real world*
Context data = event + {*time, location, id, ...*}
Context data = 'door opened' + {*10:33, 3090, Peter, ...*}

Events alone (without additional context information) are not particularly privacy sensitive, but even without additional information they can be useful for application. For example, the number of times the windows are opened in a building can be an indication for the effectiveness of the climate control mechanisms in that building. With additional context information, like the time and location of an event, both the usability and privacy sensitivity of the data increases.

Each additional dimension of context information brings it own additional usage of the data. Although we concentrate in this paper only on SPJ-queries (*select-project-join*), we list here some abilities of the time dimension:

- *stick-ability* ⇒ absolute time knowledge (e.g., 2006-09-05 14:00:45)
  - Time is a universal dimension, enables to 'stick' the event on the universal time axis, making it possible to 'select on time' and 'project time'.

    Example:

    ```
    select event, time
    from facts
    where time > 2006-09-22
    ```

- *group-ability* ⇒ "context" knowledge (e.g., month, evening, ...)
  - Exact time is replaced by "contextual" value, enables to perform 'group by' and 'select on context' queries without knowing the exact date an event happened.

    Example:

    ```
    select count(event), time as context
    from facts
    group by time
    ```

- *join-ability* ⇒ concurrent event knowledge (e.g., time represented by a join key)"
  - Discover concomitant events (even among different AmI spaces or ContextDBs), enables the join operator in queries without disclosing the exact time.

    Example:

    ```
    select event
    from facts f1, facts f2
    where f1.time = f2.time
    ```

- *order-ability* ⇒ time as a sequence number (e.g., 1, 2, 3, 8, 12)

– Keep ordering between events, predict next events to occur, enables the order by operator.
Example:

```
select event
from facts
order by time
```

With the *stick* ability, time is simply stored as a time stamp linked to the event description. Since this attribute is available, the sort, group by, and all relevant operators make sense in SQL queries. Our goal is to *degrade* the stick ability to one or more other abilities by *degrading* the time value. The set of SQL operators is then still available, but they do not have any use anymore (a 'group by' on sequence numbers is not useful). We can do similar things with the location and time dimensions. The abilities formed by all dimension can be combined.

3.1 Formalization

As mentioned before, we will only focus on SPJ-queries. To start with the most simple queries, we show that for queries using only the join, select or project operator in isolation, it is possible to find an alternative query on degraded data which returns the same result as the original query (e.g., we show *adequacy*). We start to give a formal description of the notion of *adequacy*

3.1.1 Adequacy

The goal of degrading data is, that applications, given a predefined query, still can use this degraded data to answer the query. Since degraded data contains less information than the not degraded data, this data is less privacy sensitive. To ensure that the application still retrieves correct and complete answers (the result contains no false answers and the result contains all true answers), a degraded database must be adequate:

Definition 1. Given a predefined query Q and a set of relations $\{R\}$, a degraded database $V(\{R\})$ is *adequate* (with respect to Q) if there is a query Q' such that $Q(\{R\}) = Q'(V(\{R\}))$ (see Figure 3).

Note: if we speak about a set of queries for which one degraded database $V(\{R\})$ will be used, than the degraded database $V(\{R\})$ is adequate if $\forall Q \in \{Q_0, Q_1, \ldots, Q_n\} \exists Q' : Q(\{R\}) = Q'(V(\{R\}))$.

3.1.2 Project operator

Given the query $\pi_{\{x,z\}}(R)$ on a relation $R(x, y, z, u)$, a degraded database $V_{\pi_{\{x,z\}}}$ can be found by removing the attributes $\notin \{x, z\}$:

π		π	
x	y	z	u
1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

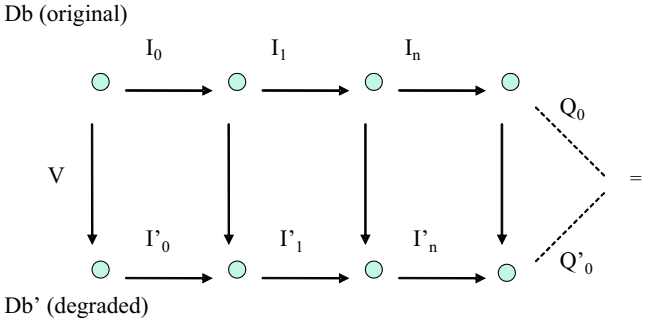

Figure 3: ***After n inserts I in the original database, a query Q on this data should return the same result as an alternative query Q' on n degraded inserts I'.***

Definition 2. Degradation for the project list $\{x, y\}$ is defined as:

$$V_{\pi_{\{x,y\}}}(\{(x_1, y_1, z_1, u_1), \ldots, (x_n, y_n, z_n, u_n)\}) = \{(x, y, z, u,) : \{(x_1, y_1, z_1, u_1), \ldots, (x_n, y_n, z_n, u_n)\} \bullet \{x, z\}\}$$

Hence, all columns from the table which are not touched by the query can simply be removed, since they play not a role in answering the query.

3.1.3 Select operator

Given the query $\sigma_{C(y)}(R)$ on a relation R (with $C(y)$ a condition applying to attribute y), a degraded database $V_{\sigma_{C(y)}}(R)$ can be found by simply removing all tuples for which $C(y) ==$ false:

	σ		
x	y	z	u
1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

Constant operand. For now, we assume that select conditions have the form $x \otimes c$, using an operator $\otimes \in \{<, >, =\}$. In some conditions, c is a constant (like a constant time value or location). If the operand is a constant, a value x which is tested against c using the operator will always return the same result. If $x \otimes c$ is false *now*, it also will be in the future, and thus will never be in the result set of the query.

Definition 3. A tuple $(x, y, z, \ldots)$ is considered as forever false with respect to x and a constant c, if the expression evaluates to false at insertion time.

Similar to forever false, tuples could be forever true.

The degradation function $V_{\sigma_{C(y)}}$, given a query Q using operator $\otimes$ on attribute A and constant c, can be specified as:

if for a tuple $t \in R(\ldots, A, \ldots) : x \otimes c == \text{false}$, with $x \in t$ an instance of A, tuple t will not be stored in the degraded database.

Definition 4. Degradation for a select condition C with constant operand c on attribute x is defined as:

$$V_{\sigma_{C(y)}}\left(\{(x_1, y_1, z_1, u_1), \ldots, (x_n, y_n, z_n, u_n)\}\right) = \{(x, y, z, u,) : \{(x_1, y_1, z_1, u_1), \ldots, (x_n, y_n, z_n, u_n)\} \; | x \otimes c \bullet \{x, y, z, u\}\}$$

Variable operand. In some queries, the select condition uses a variable operand whose value may change over time. This may result in the fact that some tuples can be true now, but are false in the future.

Example: $C(time) = time < now() - 1 \text{ hour}$

At insertion time, above expression will evaluated to true. After one hour however, the expression would evaluate to false. To handle this 'true now false in the future' condition we can again remove the tuples which are false at insertion time. The tuples which are true at insertion time will be inserted in the database. Through 'lazy evaluation' at query time, we can remove the tuples which have become false.

Example: $C(time) = time > now() - 1 \text{ hour}$

At insertion time, above expression will evaluate to false and will for a certain amount of time not be part of the query result. This tuple is thus not needed for the degraded data set to be adequate and should be removed. However, to be still adequate in the future, the tuple should be inserted in the degraded data set when the expression would evaluate to true. Additional techniques (like access control, secure hardware or distributed protocols) might be used to tackle this problem.

3.1.4 Join operator

Given the query $R \bowtie_u R$ on a relation R, a degraded database $V_{\bowtie_u}$ can be found by replacing u with an irreversible join key which can be used to make the join on the (generalized) representation of value u.

x	y	z	$\bowtie$ u	$\bowtie_{key}$ u_{key}
1	2	3	4	a
5	6	7	8	b
9	10	11	12	c
13	14	15	16	d

Definition 5. There is a function $f_k : X \rightarrow X'$ which is injective with respect to k, that is, $f(a) =_k f(b)$ implies $a =_k b$ (or $a =_k b$ implies $f(a) =_k f(b)$, for any $a, b \in X$ such that for all relations (denoted as n-tuples) the degradation function $V_{\bowtie_u}$ is defined as:

$$V_{\bowtie_u}\left(\{(x_1, y_1, z_1, u_1), \ldots, (x_n, y_n, z_n, u_n)\}\right) = \{(x_1, y_1, z_1, f_k(u_1)), \ldots, (x_n, y_n, z_n, f_k(u_n))\}$$

where y is the to be degraded component. The injective function f must replace the real value of the join attribute (the to be degraded attribute) with a join key. Predicate k defines for which generalization of that attribute the join key must be generated (and thus to which extend f_k must be injective).

With respect to privacy, generation of a join key should happen in such a way that it is computational hard to obtain x from y if $f(x) = y$. To enforce this last property on a finite domain X, a one-way key hash function can be used, for which the key is kept secret. Note however that a hash function by definition is not an infinite function, although (when used properly in practical situations) it can be assumed to be a near injective function, with low chance of duplicate hash values. Other possibilities are using block encryption [23], other forms of access control or distributed key-sharing protocols [9].

3.2 Degradation of data for a SPJ-query

In the previous sections we showed that for single operators it is possible to provide a degradation function which transforms the data such that only that information is kept which is necessary to answer the query adequate now and in the future. In this section we show that those techniques can be combined to provide a degraded data set of any *select-project-join* query. We make the assumption here that there are methods to store and process the 'true now false in the future', 'false now true in the future' and join keys.

π x	σ y	π z	$\bowtie$ u	$\bowtie_{key}$ u_{key}
1	2	3	4	a
5	6	7	8	b
9	10	11	12	c
13	14	15	16	d

Definition 6. For a single SPJ-query $\pi_X\left(\sigma_{C(y)}\left(R \bowtie_u R\right)\right)$ a degraded data set can be obtained by creating a composition function $V_{\pi_X} \circ V_{\bowtie_u} \circ V_{\sigma_{C(y)}}(R)$.

3.2.1 Proof of adequacy

Given the definition of degradation functions for *select*, *project* and *join*, we provide a simple proof that the composition of those functions provide an adequate set of degraded data. For simplicity, we assume that select condition C uses a constant operand.

The target in the following proof is to find an alternative query Q' on the degraded data which produces the same result as a query Q on the original data. We start with query Q, in which k is an generalization function as described in the previous section[1]:

$$\pi_{\{x,z\}}\left(\sigma_{C(y)}\left(R \bowtie_{k(u)} R\right)\right)$$

This query can be rewritten as:

$$\pi_{\{x,z\}}\left(\sigma_{C(y)}(R) \bowtie_{k(u)} \sigma_{C(y)}(R)\right)$$

[1]We use the notation $a =_k b$ as an abbreviation of $k(a) = k(b)$ and $f_k(a)$ as an abbreviation of $f \circ k(a)$

The select condition can be rewritten using set notation which results in:

$$\pi_{\{x,z\}}\{(x,y,z,u) : R | C(y) == \text{true} \bullet \{x,y,z,u\}\} \bowtie_{k(u)} \sigma_{C(y)}(R)$$

Indeed, this set notation is equal to the definition of $V_{\sigma_{C(y)}}$:

$$\pi_{\{x',z'\}}(\{(x',y',z',u') : V_{\sigma_{C(y)}}(R) \bullet \{x',y',z',u'\}\} \bowtie_{u'=_k u} \sigma_{C(y)}(R))$$

We therefore can replace the join on data with the select condition on the original data, with a join on the degraded data:

$$\pi_{\{x',z'\}} \left(V_{\sigma_{C(y)}}(R) \bowtie_{k(u')} V_{\sigma_{C(y)}}(R) \right)$$

Now we rewrite the join using set notation:

$$\pi_{\{x'_0,z'_0\}}(\{\left(x'_0,y'_0,z'_0,u'_0\right) : V_{\sigma_{C(y)}}(R), \left(x'_1,y'_1,z'_1,u'_1\right) : V_{\sigma_{C(y)}}(R) | u'_0 =_k u'_1 \bullet \{x'_0,y'_0,z'_0,u'_0,x'_1,y'_1,z'_1,u'_1\}\})$$

Given that an injective function f_k is used to generate a key, the following set comprehension will produce the same result:

$$\pi_{\{x'_0,z'_0\}}(\{\left(x'_0,y'_0,z'_0,u'_0\right) : V_{\sigma_{C(y)}}(R), \left(x'_1,y'_1,z'_1,u'_1\right) : V_{\sigma_{C(y)}}(R) | f_k(u'_0) == f_k(u'_1) \bullet \{x'_0,y'_0,z'_0,u'_0,x'_1,y'_1,z'_1,u'_1\}\})$$

Again, this is equal to the definition of the degradation function for joins:

$$\pi_{\{x''_0,z''_0\}}(\{\left(x''_0,y''_0,z''_0,u''_0\right) : V_{\bowtie_{k(u')}}(V_{\sigma_{C(y)}}(R)), \left(x''_1,y''_1,z''_1,u''_1\right) : V_{\bowtie_{k(u')}}(V_{\sigma_{C(y)}}(R)) | u''_0 == u''_1) \bullet \{x''_0,y''_0,z''_0,u''_0,x''_1,y''_1,z''_1,u''_1\}\})$$

We can replace the join on the original data with a join on the degraded data:

$$\pi_{\{x''_0,z''_0\}} \left(V_{\bowtie_{k(u')}}(V_{\sigma_{C(y)}}(R)) \bowtie_{k(u'')} V_{\bowtie_{k(u')}}(V_{\sigma_{C(y)}}(R)) \right)$$

Now we have to rewrite the projection on the set $\{x,z\}$:

$$\{\{x''_0,y''_0,z''_0,u''_0,x''_1,y''_1,z''_1,u''_1\} : \left(V_{\bowtie_{k(u')}}(V_{\sigma_{C(y)}}(R)) \bowtie_{k(u'')} V_{\bowtie_{k(u')}}(V_{\sigma_{C(y)}}(R)) \right) \bullet \{x''_0,z''_0\}\}$$

This results in alternative query on degraded data using the defined degradation functions:

$$\pi_*(V_{\pi_{\{x''_0,z''_0\}}}(V_{\bowtie_{k(u')}}(V_{\sigma_{C(y)}}(R)) \bowtie_{k(u'')} V_{\bowtie_{k(u')}}(V_{\sigma_{C(y)}}(R))))$$

□

4. APPLICATION ORIENTED DEGRADATION

Now we have the techniques to degrade and transform the data in order to make the data less privacy sensitive, we can use them in *policies* in which can be specified to what form the data must be stored and how long it must be retained. In this context, application can communicate their requirements by queries on the data (we assume that it is known which data is available from the sensors in the AmI-space). After degrading and transforming the data in such way that only that information remains such that the queries still can be answered, it can be up to the donors of the data (e.g., the users of the applications) to give their consent and to specify until how long their data may be retained in the system.

To clarify this, see the example given in figure 4. Here we have a *base relation R(id,location,time)*, that is, tuples

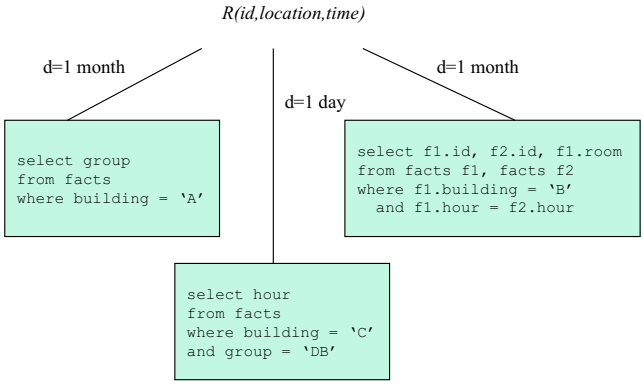

Figure 4: ***Three queries are specified on a relation R(id,location,time) and retention periods δ after which the data should be removed such that it will not be part of the query answer anymore.***

are coming in from sensors with accurate ids of persons being monitored, the exact location and time (the monitored events themselves are left out of consideration in this example). Applications using this data want to have the *ability* to know which groups (that is, at least one person from a group) have entered a particular building (Q_1), they want to know which pairs of persons where in a particular building *within the same hour* and in which room (Q_2) and they want to have the ability to know in which hour(s) a member of a particular group has entered a building (Q_3).

For these queries to be answered, it is not necessary to store and keep all data accurate and privacy sensitive, so we are looking for degraded sets of data which still can answer the queries *adequate*, but are less privacy sensitive. To do this we first create out of base relation R three sets of data $\{R_1, R_2, R_3\}$ which can answer Q_1, Q_2 and Q_3:

$$R_1\{id, location, time\}$$
$$R_2\{id, location, time\}$$
$$R_3\{id, location, time\}$$

For each set we give a degradation function:

$$R'_1 = V_1(R) = V_{\pi_{\{id\}}} \circ V_{\sigma_{\text{building}=A}}(R)$$
$$R'_2 = V_2(R) = V_{\pi_{\{id,id,loc\}}} \circ V_{\sigma_{\text{building}=B}} \circ V_{\bowtie_{hour(time)}}(R)$$
$$R'_3 = V_3(R) = V_{\pi_{\{time\}}} \circ V_{\sigma_{\text{building}=C \wedge \text{group}=DB}}(R)$$

This results in the following relations:

$$R'_1\{id\}$$
$$R'_2\{id, id, location, time\_key\}$$
$$R'_3\{time\}$$

The attributes in those relations are more accurate than required by the queries. We use here the techniques as described in section 2 to degrade these values and attach a retention delay to it after which the values will finally be removed from the system. Those *life cycle policies* are now

relative simple:

$$S_1 = \{s_1\} = \{(group, \emptyset, \emptyset)\}$$
$$S_2 = \{s_2\} = \{(id, id, room, time\_key)\}$$
$$S_3 = \{s_3\} = \{(\emptyset, \emptyset, hour)\}$$
$$\Sigma_1 = \{t_1\} = \{\text{after 1 month}\}$$
$$\Sigma_2 = \{t_1\} = \{\text{after 1 month}\}$$
$$\Sigma_3 = \{t_2\} = \{\text{after 1 day}\}$$

$$lcp_1 = \{S_1, \Sigma_1, \{\delta(s_1, t_1) = \emptyset\}, s_1, \emptyset\}$$
$$lcp_2 = \{S_2, \Sigma_2, \{\delta(s_2, t_1) = \emptyset\}, s_2, \emptyset\}$$
$$lcp_3 = \{S_3, \Sigma_3, \{\delta(s_3, t_2) = \emptyset\}, s_3, \emptyset\}$$

Finally, this results in the following sets which will be available for querying:

$$R_1''\{group\}$$
$$R_2''\{id, id, room, time\_key\}\}$$
$$R_3''\{hour\}$$

The only thing left is, that we need to specify the (trivial) alternative queries Q_1',Q_2' and Q_2' on the degraded data which should return the same results as the original queries on relation R:

Q_1' = select group from R_1''
Q_2' = select id1, id2, room from R_2'' r1, R_2'' r2
where r1.time_key = r2.time_key
Q_3' = select time from R_3''

4.1 Performance issues for updating materialized views

To some extent, degrading the base relation into a sets of degraded data is similar to creating materialized views from the base relation. Much previous research has been done on 'maintaining materialized views', of which the most deal with the relevancy of updates for the view, that is, is (for example) an insert independent of the query for which the view has been build [3]. According to Gupta et al [14], when there is only partial or no information (there is no base relation from which the view can be recalculated), those 'query independent of update' algorithms [3, 4] can determine if the view needs to be updated and if so, some algorithm can be used for that. Those kinds of views are called self-maintainable views.

In the context of our privacy work, all views should be self-maintainable, which is quite easy if we consider that there are only inserts. According to the work of Levy [20], knowing wetter or not a view is adequate (answer complete and answer correct) can be expressed in terms of 'query dependency'. This means that all performance issues can be concentrated on this topic: how much performance does it take to check if an incoming tuple touches the view (cf. section 3.1.3 in which we described that a tuple must be tested on true or false at insertion time).

Maintaining multiple views may be very costly in terms of performance: if $n - k$ tuples are query dependent, the tuple must be inserted in k separate views generating k I/Os (instead of just one I/O in the original base relation). Further research must resolve if this really is an issue, and if there are perhaps solutions in the literature for this. Indeed, the tuple must be inserted in k separate files, but this does not necessarily mean that the data is also k times replicated. In the optimal case, the tuple is split into k distinct parts, each part inserted into a different file. Dependent on the organization of the database (see for example MonetDB [5], in which a storage model based on vertical fragmentation has been used), this might not be costly at all.

5. FURTHER WORK

The work as presented in this paper is clearly work in progress, and can be seen as a first step into further research in comprehensive retention and collection policies. For example, in section 2 we specified life cycle policies which support the degradation of values in multiple steps, given multiple retention periods. Those policies can be specified by each user individually, making the degradation process *user-oriented* instead of *application-oriented*. In this scenario users have even more control over what kind of data they want to provide to application, so that applications have to negotiate with users. However, individual policies introduce additional difficulties which must be solved. In order to maximize the performance of our approach, optimization techniques can be investigated with a comprehensive performance study.

As mentioned in section 3.1.3 and 3.1.4, we still need some access control techniques in order to update the degraded data set to keep it accurate. Additional research has to be done to find the best suitable techniques in terms of privacy protection and performance. One possibility is to slightly relax the adequacy condition in exchange for less access control and more limited retention techniques.

6. CONCLUSIONS

In our approach, we reuse well known materialized view techniques with a new purpose: minimizing the amount of data stored in the system to answer queries to comply with the limited collection and retention principles. Since base relations will not be available, those views must be self-maintainable (or at least with a minimized amount of additional information needed to update the views), introducing additional difficulties. However, by clearly communicating to users which data will be collected and stored for which period, the asymmetric information difficulties between donors and collectors of data will be strongly reduced. Access control methods can still be used to restrict access to the remaining data, but at times access control is not sufficient to resist against attacks on the database, the amount of disclosed data and the privacy sensitivity of it at least is reduced.

7. ACKNOWLEDGEMENTS

This research is funded by NWO (Nederlandse Organisatie voor Wetenschappelijk Onderzoek; Netherlands Organization for Scientific Research), under project 639.022.403 (CADMAI: Towards Context-Aware Data Management for Ambient Intelligence), BRICKS (Basic Research in Informatics for Creating the Knowledge Society) and INRIA (Institut National de Recherche en Informatique). We particularly like to thank Peter Apers and Maarten Fokkinga from the University of Twente, and Philippe Pucheral and Luc Bouganim from INRIA for their involvements.

8. REFERENCES

[1] R. Agrawal, J. Kiernan, R. Srikant, and Y. Xu. Hippocratic databases. In *28th Int'l Conf. on Very Large Databases (VLDB), Hong Kong*, 2002.

[2] N. L. G. Anciaux, H. J. W. van Heerde, L. Feng, and P. M. G. Apers. Implanting life-cycle privacy policies in a context database. Technical Report TR-CTIT-06-03, Enschede, January 2006.

[3] J. A. Blakeley, N. Coburn, and P. Larson. Updating derived relations: Detecting irrelevant and autonomously computable updates. *ACM Transactions on Database Systems*, 14(3):369–400, 1989.

[4] J. A. Blakeley, P.-A. Larson, and F. W. Tompa. Efficiently updating materialized views. In *SIGMOD Conference*, pages 61–71, 1986.

[5] P. Boncz and M. Kersten. Monet: an impressionist sketch of an advanced database system, 1995.

[6] J.-W. Byun and E. Bertino. Micro-views, or on how to protect privacy while enhancing data usability. Vision paper CERIAS Tech Report 2005-25, Center for Education and Research in Information Assurance and Security, West Lafayette, IN 47907-2086, 2005.

[7] S. Chawla, C. Dwork, F. McSherry, A. Smith, and H. Wee. Toward privacy in public databases. In *Theory of Cryptography Conference*, 2005.

[8] CSI/FBI. *CSI/FBI computer crime and security survey*. 2005.

[9] T. Dimitriou and I. Krontiris. A localized, distributed protocol for secure information exchange in sensor networks. In *IPDPS '05: Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium (IPDPS'05) - Workshop 12*, page 240.1, Washington, DC, USA, 2005. IEEE Computer Society.

[10] C. Doom. Get smart: How intelligent technology will enhance our world. Technical report, Computer Sciences Corporation: Leading Edge Forum, 2001. A report available from www.csc.com.

[11] K. Ducatel, M. Bogdanowicz, F. Scapolo, J. Leijten, and J. Burgelma. *Scenarios for ambient intelligence in 2010*. Institute for Prospective Technological Studies (IPTS), Seville, February 2001.

[12] L. A. Gordon, M. P. Loeb, W. Lucyshyn, and R. Richardson. *CSI/FBI computer crime and security survey*. Computer Security Institute, 2006.

[13] M. Gruteser, G. Schelle, A. Jain, R. Han, and D. Grunwald. Privacy-aware location sensor networks. In M. B. Jones, editor, *HotOS*, pages 163–168. USENIX, 2003.

[14] A. Gupta and I. S. Mumick. Maintenance of materialized views: Problems, techniques and applications. *IEEE Quarterly Bulletin on Data Engineering; Special Issue on Materialized Views and Data Warehousing*, 18(2):3–18, 1995.

[15] R. J. Hilderman, H. J. Hamilton, and N. Cercone. Data mining in large databases using domain generalization graphs. *J. Intell. Inf. Syst.*, 13(3):195–234, 1999.

[16] J. I. Hong and J. A. Landay. An architecture for privacy-sensitive ubiquitous computing. In *MobiSys '04: Proceedings of the 2nd international conference on Mobile systems, applications, and services*, pages 177–189, New York, NY, USA, 2004. ACM Press.

[17] ISTAG. Orientations for wp2000 and beyond. final report, 1999.

[18] X. Jiang, J. I. Hong, and J. A. Landay. Approximate information flows: Socially-based modeling of privacy in ubiquitous computing. In *UbiComp '02: Proceedings of the 4th international conference on Ubiquitous Computing*, pages 176–193, London, UK, 2002. Springer-Verlag.

[19] M. Langheinrich. Privacy by design - principles of privacy-aware ubiquitous systems. In *UbiComp '01: Proceedings of the 3rd international conference on Ubiquitous Computing*, pages 273–291, London, UK, 2001. Springer-Verlag.

[20] A. Y. Levy. Obtaining complete answers from incomplete databases. In *VLDB '96: Proceedings of the 22th International Conference on Very Large Data Bases*, pages 402–412, San Francisco, CA, USA, 1996. Morgan Kaufmann Publishers Inc.

[21] A. Machanavajjhala, J. Gehrke, D. Kifer, and M. Venkitasubramaniam. l-diversity: Privacy beyond k-anonymity. In *CLDB*, 2006.

[22] R. Mayrhofer. Context prediction based on context histories: Expected benefits, issues and current state-of-the-art. In T. Prante, B. Meyers, G. Fitzpatrick, and L. D. Harvel, editors, *Proceedings of the 1st International Workshop on Exploiting Context Histories in Smart Environments (ECHISE2005)*, May 2005. part of the Third International Conference on Pervasive Computing (PERVASIVE 2005).

[23] A. J. Menezes, P. C. van Oorschot, and S. A. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 1996.

[24] PhilipsResearch. Ambient intelligence, changing lives for the better. `http://www.research.philips.com/technologies/syst_softw/ami/background.html`.

[25] G. Riva, M. L. Pierpaolo Loreti, F. Vatalaro, and F. Davide. Presence 2010: The emergence of ambient intelligence. In *Being There: Concepts, effects and measurement of user presence in synthetic environment*, pages 60–81. Ios Press.

[26] L. Sweeney. Achieving k-anonymity privacy protection using generalization and suppression. *International Journal on Uncertainty Fuzziness and Knowledge-based Systems*, pages 571–588, 2002.

[27] L. Sweeney. k-anonymity: A model for protecting privacy. *International Journal on Uncertainty Fuzziness and Knowledge-based Systems*, pages 557–570, 2002.

[28] W3C. Platform for privacy preferences (P3P) project. `http://www.w3.org/P3P/`, June 2005.

[29] H. J. W. van Heerde, N. L. G. Anciaux, L. Feng, and P. M. G. Apers. Balancing smartness and privacy for the ambient intelligence. In *Proceedings of the 1st European Conference on Smart Sensing and Context (EuroSCC 2006), Enschede, The Netherlands*, volume 4272 of *Lecture Notes in Computer Science*, pages 255–258, Berlin, October 2006. Springer Verlag.

[30] X. Xiao and Y. Tao. Personalized privacy preservation. In *ACM Conference on Management of Data (SIGMOD)*, 2006.