Implanting Life-Cycle Privacy Policies in a Context Database

Technical Report, January 2006

University of Twente, Enschede, The Netherlands

Nicolas Anciaux, Harold van Heerde, Ling Feng, Peter M.G. Apers

Abstract

Ambient intelligence (AmI) environments continuously monitor surrounding individuals' context (e.g., location, activity, etc.) to make existing applications smarter, i.e., make decision without requiring user interaction. Such AmI smartness ability is tightly coupled to quantity and quality of the available (past and present) context. However, context is often linked to an individual (e.g., location of a given person) and as such falls under privacy directives. The goal of this paper is to enable the difficult wedding of privacy (automatically fulfilling users' privacy whishes) and smartness in the AmI. Interestingly, privacy requirements in the AmI are different from traditional environments, where systems usually manage durable data (e.g., medical or banking information), collected and updated trustfully either by the donor herself, her doctor, or an employee of her bank. Therefore, proper information disclosure to third parties constitutes a major privacy concern in the traditional studies.

On the contrary, AmI is based on autonomous and invisible data collection with weak durability requirements (from donors' point of view), which puts regulation of context data life-cycle on the hot seat. More precisely, we propose to bind Life-Cycle Policies (LCP) to context data regulating its progressive degradation. This paper makes the following contributions. (i) It introduces the Life Cycle Policy (LCP) model to regulate the content of context databases; (ii) it investigates the problem of correctness of the LCP model when used to implant one-way degradation (i.e., ensure that degraded information can no more be recovered from the current database content); (iii) it implants LCP on top of a traditional DBMS, to provide a practical understanding of the model and show the feasibility of the proposed techniques. Finally, it presents new challenges linked to our approach and concludes the paper. We are convinced that providing LCP on autonomous systems paves the way to new privacy solutions.

Keywords

Privacy, Context Management, Ambient Intelligence, Multi-granularity Data Model, Life-Cycle Policy.

1. INTRODUCTION

1.1. Background

Undeniably, information systems are more and more aware of their surroundings thanks to widely spread smart devices able to continuously nourish the computing infrastructure with information describing the real world [20]. Not only lightweight devices like cell phones, PDA, RFID tags localizing objects [38][9] and employees (Omron[1] company) but also chips penetrate such commonly used objects as clothes [27], televisions [28], GPS-equipped cars, floors [25] etc. and even target at the human body [39][31]. This new combination of

[1] See http://ubiks.net/local/blog/jmt/archives3/003741.html

technologies is pushing towards so called pervasive computing having real-time knowledge of the surroundings. This knowledge is called *context*[2] throughout the paper.

Context gives the ability to a pervasive computing environment to become Ambient Intelligence (AmI) [19], i.e., be endowed with enough *smartness* to (i) minimize user-machine interactions (automation is pushed to the extreme), and (ii) deploy this environment in daily life areas (like streets, supermarkets, homes, etc.) with non-traditional stationary desktop-based computing interfaces. Thus, continuously sensed context reflecting individuals' location, behavior, mood, or habits etc. is needed to reach those smartness requirements, i.e., to *adapt* computing applications to fit the environment; *infer* information or draw conclusions from rules and observations; *learn* in the sense of using experiences to improve performance, and to *anticipate* what to do next [14]. An obvious but important remark is that the smartness of AmI is tightly coupled to quality (i.e., accuracy) and quantity of the available past and present context, as exposed in [32][40] at the first workshop [15] dedicated to usage of context histories in smart environments.

1.2. Motivation and goal

Although context information might be considered less sensible than traditional data like healthcare folders or banking information, it falls under privacy regulation when linked to an individual (e.g., location of a person), given the definition of "personal data" in the worldwide privacy acts [36][35][24][17][26]. Moreover, AmI spaces are planned to be developed everywhere and leads in the extreme to sense and archive everyone's actions and moves everyday [22][14]. Also, with sufficient accuracy and volumes, context may become of main interest [20] for malicious usage, e.g., to monitor employees in companies, detect behavior of inhabitants and/or company staff to prepare a robbery or for marketing purposes, or check people behaviors before contracting insurance contracts. This inevitably constitutes a critical privacy threat.

Nevertheless, enforcing privacy protection in the AmI generates the difficulty in controlling and exploiting the content (volume and accuracy) of context histories. In fact, privacy has widely been recognized as being application killer in the AmI (will prevent AmI to be accepted), forming a crucial problem. The goal of this paper is to investigate this issue and provide sensible solutions to reconcile smartness and privacy in the AmI world.

An obvious but important remark is that controlling the content of context histories cannot be simply delegated to *access control*. Indeed, as opposed to traditional sensitive data like healthcare, financial, or insurance information which are subject to privacy protection, context does not require *durability* requirement from users' point of view. That is, context information hurting users' privacy and discarded to anyone will not be useful for its owner. Retaining this data in the system should be proscribed for the following reasons. First, some internals (e.g., system or database administrators) often have access privileges to the whole contents for administration purpose, which constitutes a major privacy threat while half attacks are actually conducted by insiders according to the computer crime and security survey [12]. Second, retaining more data increases motivations (and benefits) of privacy attacks conducted, and currently no solution can fully guarantee that access control mechanisms cannot be bypassed. Third, protecting data by access control has a strong impact on system performance (e.g., views complexity in databases). Particularly, in an AmI world, the amount of consistently sensed context information is huge. For these reasons, controlling the content of context histories cannot be simply delegated to disclosure mechanisms, but is a problem in itself. We do not address access control in the paper. We consider it as an orthogonal problem aiming at properly regulating disclosure of the content of histories to third parties, while our target is to regulate the content of histories itself.

---

[2] In [13], the database storing location profiles values that are subject to change frequently is called a *context*. By analogy, we call here *context* data describing the current state (very volatile) of the physical surrounding (e.g., people or objects location, activity, status, temperature, heart rate, mood, etc.). However, for sake of simplicity, we extend the usage of the term *context* to data describing past surrounding states.

1.3. Solution requirements

Providing automatic control over histories' life-cycle is a major challenge, notably due to the following AmI properties.

Smartness property: The smartness requirement in the AmI imposes to make available any context information which is defined as non-private by its owner. This mainly leads to devise rich and customized (e.g., per user) regulation of context histories, precluding uniform or coarse grain processes.

Usefulness property: Context information might always be useful to make autonomous decisions, which makes it difficult to calibrate lifetime at the system level (as opposed to traditional data, e.g., credit card numbers or a mail address which might be deleted after use).

Distribution property: AmI spaces are composed of many devices, including sensors, network components like routers, databases, and user devices. Many of those might constitute context histories about surrounding individuals, thus any of them should be able to regulate the accumulated content given privacy policies. It is worth pointing out that although we focus in this paper on database recipients, our solution should be adaptable to any third party components. This clearly induces simple regulation processes possibly achieved by non-powerful devices.

Performance property: Performance is highly demanded at data management level to process real-time sensed/acquired huge amount of data, store large context histories, and answer complex queries (required to take autonomous decisions). Performance issues of such a database have been studied in [7], inducing particular data management techniques, which of course proscribes too heavy privacy techniques (e.g., encryption intensive operations).

1.4. Limitation of existing solutions

Existing privacy preserving solutions do not fit those main requirements. Many privacy studies in database systems focus on information disclosure. Although limiting data disclosure and providing multi-leveled views [6] are useful and existing solutions meet the smartness and performance of AmI, it cannot be substituted to life-cycle management.

Recent privacy studies in databases address content life-cycle to certain extend. First, the well-known Platform for Privacy Preference (P3P) recommended by the W3C [10][11] uses the *notice and consent*[3] practice and data lifetime to regulate content. The privacy aware system [21] aiming at enabling privacy in ubiquitous environments and Hippocratic databases [4][5] enabling privacy in corporate databases make both use of this practice. While this clearly shows the importance of controlling content, such approaches hurt the smartness and usefulness properties. First, the notice and consent practice leads to apply the server privacy policy (even difficult to set properly given the usefulness property), which would lead to discard an important amount of non-private context information, e.g., context regarding users holding weaker privacy whishes would be governed by stricter parameters defined at the server and users holding stricter policies would simply not be monitored. Second, content life-cycle is regulated by deleting expired data (after a given period or when the associated purpose is achieved). This coarse grain attempt towards data degradation is justified for data becoming completely useless after usage (e.g., credit card number), but does not suit the AmI.

[3] The privacy practice is expressed by servers and users. In case the server policy, weaker, does not comply with those expressed by users, stricter, a notification is send. Users can either consent to the weaker policy or reject it and do not use the system.

Data downgrading has been further investigated to provide privacy, in balance with data usefulness, in data mining [37] and statistical databases [1]. Interesting techniques have been proposed, based on randomization [3] or data suppression and generalization [29][30][34][33] to obtain k-anonym datasets. However, the downgrading process is uniform (identical rules applied to the complete dataset) and performed only once to generate a public release, which hurts the smartness property. In addition, techniques are often calibrated for given mining algorithms or statistics computations, and require the whole original dataset [29][30] to produce the public release, which strongly impacts the feasibility of a transposition to the AmI.

1.5. Our Work

The first goal of this paper is to enable both privacy and smartness in AmI environments, by use of data degradation. In addition, the degradation model we propose offers the twofold: (i) it simplifies access control and security measures (thus reducing the cost of query evaluation); and (ii) it saves storage space, which also increases query performance.

Our strategy to achieve this goal is to offer to individuals and organizations a control over the life cycle (i.e., progressive degradation) of their context information. We consider life-cycle context data management as a crucial aspect of privacy in the AmI, and focus on a database dedicated to store context corresponding to a given AmI space (e.g., a building). The problem we address is then to devise a general model to regulate the content of context histories at the database level, which comply with the AmI properties and prevent from retrieving accurate values after degradation has been performed.

Specifically, we introduce a life cycle privacy protection model to enable users to declare their degradation whishes. These policies will be bound to their context data, so that any AmI component and system can interpret it and perform required data degradation (the distribution problem raises many further issues discussed in Section 8). As an example, assume an AmI space monitors the location of surrounding individuals in an office environment. Her location context can be described using a triplet, including *ID* as a unique identifier of an employee, *Time* (in seconds) when the location is monitored, and *Location* as a unique room identifier. A possible life cycle policy specified by an employee to preserve her privacy could be the following: keep the accurate location (e.g., room identifier) for 10 minutes so that some colleagues can reach her; 10 minutes later, degrade her exact room identifier to the floor where the room is located, so that her exact location is not traced. Then, if she decides to go home incognito and early (let's say, leaves the building by the backdoor before 5 pm), she may degrade the exact time from second to day (so that AmI can not break her right to hide the exact time when she left), otherwise Time is degraded 8 hours later. In any case, one week later the tuple should be degraded as follows: ID is turned to her working group identifier (e.g., DB group), Time is turned to day, and Location is turned to the corresponding building identifier. One month later, tuple will be deleted.

We believe that such a control over users' own context histories enables both smartness and privacy protection for the AmI. On the one hand, since individuals can express their precise and personalized wishes, the whole context they regard as non-sensible or useful for AmI applications to offer them sufficient smartness and benefits might be retained. Of course, applications offer such smartness given the permission of policies. On the other hand, privacy is also provided, the system being able to progressively degrade context information and only retain the desired "souvenir" of individuals' past status.

In this study, we consider attacks conducted by *snooping* at databases storing context histories. We assume that communication between database and other recipients are secured and trusted, but data files, log files, and main memory can be observed. At the time of the observation, previous states of the context history should not be retrieved given the current content of the database. We call this the "one-way" property of the degradation (more detailed in Section 3.2). When controlling data life cycle in real time systems, compliance with this property is difficult to obtain because of the cohabitation of accurate context values with degraded ones, and because the database has to be aware of life-cycle settings to properly achieve degradation.

In this paper, we give the following contributions. First, we introduce the Life Cycle Policy (LCP) model to regulate the content of context histories, based on a cubic representation of context information and on automata specifying expected degradation of this information. Second, we show by examples the interest of the LCP model regarding the privacy requirements from both the organization holding the AmI space and surrounding individuals moving in this AmI space. For each class of LCP, we analyze compliance with the one-way property of the degradation. Third, we present a brute force instantiation of our model, having the interest of making the model more understandable and showing its feasibility in practice. Finally, we give the list of open problem introduced by this approach.

The rest of the paper is organized as follows. Section 2 introduces the AmI architecture that we consider allover the paper. Section 3 presents the LCP model. Section 4, 5 and 6 present classes of LCP adapted respectively to the privacy requirements of an organization holding an AmI space, those of individuals moving a public AmI space, and LCPs including user specific subparts. Section 7 gives a brute force instantiation of the LCP model on top of Postgres. Section 8 presents new challenges induced by the LCP model and Section 9 concludes the paper.

2. GENERAL ARCHITECTURE

Figure 1 shows our AmI privacy protection architecture. The acquisition devices extract surrounding context related to a set of individuals called *donors*[4] to nourish a database called *Context-DB*. The Context-DB, storing and managing the huge amount of context information, is responsible for providing context to applications to make them smarter, i.e., to help them make autonomous decisions, in the spirit of the AmI vision [19], regarding the set of surrounding individuals called *users*.

For sake of simplicity, here, we consider one database storing the whole context monitored in the surrounding AmI environment. However, we recognize that several AmI spaces can be deployed contiguously (e.g., one AmI space per flat in a building), and a given AmI space will probably be endowed with several databases, we do not tackle into the issue in this study. However, note that other privacy solutions, as the Confab toolkit [18] designed for facilitating development of privacy sensitive ubiquitous computing applications and P4P platform enabling privacy for the paranoids [2], argue for a decentralized architecture (i.e., most information is stored on donors' device) but do not consider histories. Even though we recognize the interest of keeping data on user devices, we consider this as incompatible with the vision of the AmI based on autonomous decision to reduce user interactions. Moreover, AmI spaces target at daily environments, where many users do not hold powerful computing devices. Considering a central context database in a surrounding AmI environment also makes sense for performance issues.

[4] This terminology is taken from [5].

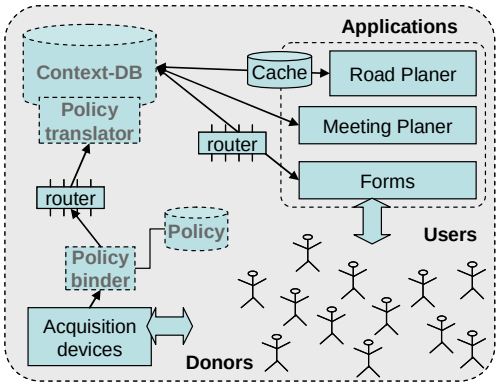

Figure 1. General architecture.

To conform to the distribution property of the AmI, we consider that sensed context is first processed by a *policy binder* in charge of binding the LCP regulating its life cycle. This policy binder is placed close to or even within the acquisition devices. Hence, each component of the AmI (e.g., routers, application cache, user devices) is aware of the defined LCP and thus might apply the LCP. At the database level, we consider then incoming data bound to its corresponding LCP.

Moreover, a *policy translator* is available at each component to translate the policy into a language, which is understandable at the component (e.g., SQL for a relational database). Then the component will be in charge (possibly with the help of the translator) of managing properly the context life cycle following the LCP.

3. THE LIFE-CYCLE POLICY (LCP) MODEL

Our design of the context life cycle policy is driven by the following key considerations. To comply with the distribution property, the LCP must be self-contained, i.e., containing the whole required knowledge to be interpreted and simple to apply. To comply with the smartness property, LCP must be rich enough to avoid useless information loss (regarding privacy). In addition, the usefulness property claims for user defined LCP. Finally, LCP should be easily transmissible to third parties to enable (controlled) data replication.

In the following, we introduce our context degradation policy based on a cubic representation, and our view of the degradation policy as a path in this cube using the automata model.

3.1. Context State

We model context information sensed in an AmI space as a *context triplet* (*ID*, *Time*, *Value*), consisting of the *ID* of the donor, the *Time* when this context was acquired, and the context concrete *Value* itself (e.g., location coordinates). This triplet can take values in different *context state*.

Accuracy / *Dimension*	*Level 1*	*Level 2*	*Level 3*	*Level 4*	*Level 5*
Acquisition Time	Second	Minute	Hour	Day	Month
Donor's ID	Employee	Group	Dept.	University	
Location	Coordinate	Room	Floor	Building	Area
Activity	Detail	Class	General	Status	

Table 1. Dimensions of Location and Activity cubes.

The context triplets can have different levels of accuracies based on domain generalization techniques. Previous work on domain generalization [23] can be a support here. For sake of simplicity, we consider that those

different accuracies can be classified given the privacy of the information they represent, as illustrated in Table 1. Note that although generalization domains might be a graph (non-linear), we assume that any element of the graph can be ranked regarding the privacy concern of information attached to it, depending on specific application scenarios. The different combinations of accuracies thus form a cube (see Figure 2). Each coordinate in the cube corresponds to a context state at a certain accuracy level. For example, coordinates (*3*, *3*, *2*) designates element E_3 of the location cube in Figure 2, which contains context triplets storing the donor's identifier as a *department* number, the date in *hour*, and the location as a *room* number. In the current study, we consider that each kind of context (e.g., location or activity) corresponds to such a context accuracy cube with rich generalization for the three dimensions, namely, *ID*, *Time*, and *Value*.

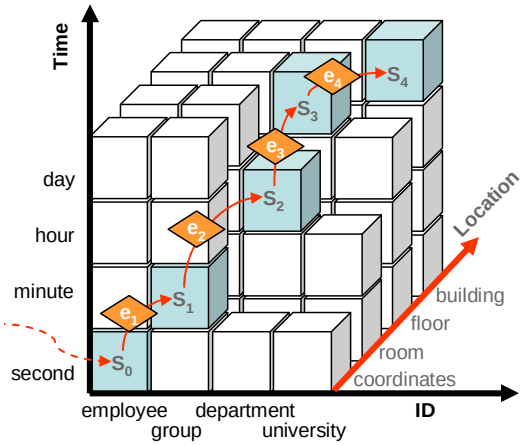

Figure 2. A cubic representation for *Location* context.

A *binary partial order relationship* $\geq_a$ can be further defined in the cube to compare the accuracy levels of two context states ($s_i \geq_a s_j$), if and only if each of the three dimensions of s_i has a higher or equal accuracy level than that of s_j.

Note that the cubic representation of the data is already used, e.g., in data warehouse [8] to represent the (complex) result of a given query. The main difference with our representation is that each dimension takes here different data accuracies linked to a given domain of values (more or less generalized) or to a given data type (more or less precise), ordered from the more accurate (e.g., exact *coordinates* for location) to the less accurate (e.g., *building*), instead representing an ordered set of discrete value (e.g., *years*) or interval (e.g., *age* between *0-10*) having the same accuracy.

We propose to represent each database table used to store context as a multidimensional table, each dimension corresponding to given attributes accuracies, with decreasing privacy concerns. Here, the cube must provide a schema as rich as possible to offer a broad panel of possible LCP states and minimize the gap between donors' wishes and schema possibilities.

3.2. Context State Degradation

Triggered by events, the accuracy of a context triplet can be consistently down degraded, forming the life cycle of a certain type of context information. Figure 2 plots an evolution example for context triplets representing people's location. According to the happening places, we categorize three kinds of events, namely, (i) *universal events* which can be generated and thus be available in any AmI space (e.g., a time predicate); (ii) *internal events* which originate from a specific AmI space, and are usually monitored by sensors or related to a component like a database access or a printer error; and (iii) *external events* which are monitored in another AmI space, thus originating externally.

Formally, we can define the life cycle of a context triplet for a certain type of context as a *deterministic finite automata* ($S, \Sigma, \delta, s_0, s_f$), where S is a set of *context states* regulating the form of a the corresponding context triplets (*ID, Time, Value*); Σ is a set of *events*; δ is a set of *transition functions* $S \times \Sigma \rightarrow S$, satisfying that for a transition $\delta(s_i, e_k) = s_{i+1}$, $(s_i \geq_a s_{i+1})$; $s_0 \in S$ is the *context starting state*; and $s_f \in S$ is the *context final state*, which can be empty value $\varnothing$ corresponding to the deletion of context triplets.

For example, the following automata

$S=\{s_0, s_1, s_2, s_3, s_4, s_f\}=\{(1,1,2), (1,1,3), (1,4,3), (1,3,3), (2,4,3), \varnothing\}$,

$\Sigma=\{e_1, e_2, e_3, e_4\}=\{10$ minutes later, leave through backdoor before 5pm, 8 hours later, 1 week later, 1 month later$\}$,

$\delta(s_0, e_1)=s_1, \delta(s_1, e_2)=s_2, \delta(s_1, e_3)=s_3, \delta(s_2, e_4)=s_4, \delta(s_3, e_4)=s_4, \delta(s_4, e_5)=s_f$,

$s_0 = (1,1,2), s_f = \varnothing$.

describes the life cycle policy example given in Section 1 to preserve the privacy in case one decides to leave the office building earlier. Its pictorial representation is shown in Figure 3.

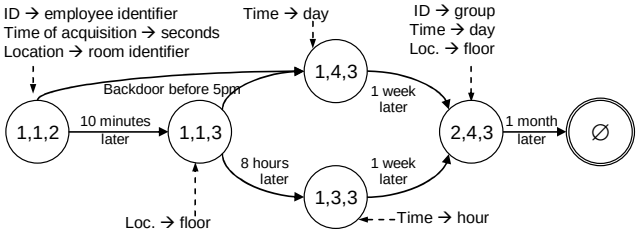

Figure 3. LCP automaton example.

The final state of context data Location will store ID as the working group of the donor, Time as a day, and Value as the building identifier, which will be retained durably in the database (or AmI component), and considered as non private by the donor.

The one-way property of the model

We concentrate in this paper on the usage of the LCP model. In particular, the correctness of the model in providing privacy lies in its *one-way property* guarantee. That is, from an already degraded context triplet, the system (even the DBA of the Context-DB) is not able to derive previous accurate values. However, this one-way property could be potentially violated, and then violates users or organizations' privacy.

In particular, compliance with this property induces constraints on the automata itself, impacts the logging process, and requires some particular techniques in the particular case of degradation along *Time* and *ID* fields.

4. ORGANIZATION ORIENTED LCP

We consider here LCP defined by an organization (e.g., a company, a country) to preserve its own privacy, preventing useless context retention that could be subject to attacks from its competitive organizations.

4.1. Motivating example

To achieve its privacy goal, an organization has to minimize the available (retained) data within its own information system in order to avoid potential spying. Using LCP, an organization can parameterize its own information system to only retain context information which is strictly required in providing the services to improve its efficiency.

For example, a company could require (i) phone call redirection, (ii) automatic filling-in daily timetable forms, (iii) room availability forecasting for the next week, (iv) statistics in terms of visibility of different teams (i.e., number of days per week a team is represented by one of its members in the organization), and so on.

To provide these services with privacy in mind, the following LCP (pictured in Figure 4) could regulate employees' location information acquired by the AmI space.

Following this LCP, the AmI space retains (i) accurate location states (employee *ID*, precise acquisition *Time*, and *Room* identifier) for a few minutes (allowing phone call redirection), (ii) then the exact date of acquisition is degraded to *Hour* of acquisition (enough to allow automatic fill-in of daily timetables), (iii) one day later, employee's *ID* is degraded to her *Team* identifier (enabling room availability forecast for the next week), and finally, (iv) one week later, identifier of *Room* is deleted (still allowing day-per-week visibility of the team at work). In this particular case, the last context state is considered as non dangerous for the organization's privacy, and can thus be retained durably in the system to enable further statistic computations and long term historical analysis.

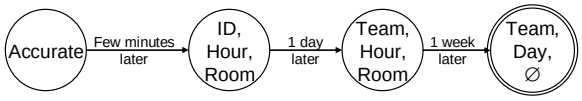

Figure 4. Organization oriented LCP.

Although this LCP is shared by all the employees of the company, it reduces the amount of context information available in the AmI space, which could be accessible in case of a spying attack conducted by a competitive.

4.2. One-way property analysis

The correctness of the LCP model requires complying with the one-way property of data degradation (see Section 3.2). However, this one-way property could be potentially violated by a few factors. In the following, we analyze two possible risks coming from LCP itself and system log file, and present our solutions.

Risk 1: *The LCP itself might induce unexpected information disclosure*. As a LCP automaton is directly bound to a context state, from an already degraded context state, there is a risk of revealing parts of its previous more accurate one (i.e., its donor (*ID* field), acquisition date (*Time* field), or context value (e.g., *Location* field)). To resolve this problem, the definition of a LCP must observe the following two principles. First, context states in a LCP must strictly follow the descending order in accuracy along with each degradation. Second, information disclosed by each context state transition via its triggering event should not be more accurate than the current context state it applies on. In other words, the definition of the event should be computed based on the current context state to be degraded or using an irreversible function upon accurate values. The conformity of LCP automata to the above two principles will be checked at the instantiation time by the policy binder (see Figure 1).

For the event, let's take a close look at the degradation of the *Time* field based on time delays. Time events have the particularity to preserve a partial ordering of tuples among states on the Time field. More formally, transitions based on time delays leads to the following situation. Let Δ be a snapshot of a context cube at a given time. Let P be the policy based on time delays regulating the content of the cube, specifying a set of n consecutive states $\{S_i\}_{i=0..n-1}$ and a set of $n-1$ time delays $\{d_i\}_{i=1..n-1}$ specifying the delay to respect before transferring one tuple from a state to the next one. Let t_{i+1} be a tuple in the cube belonging to a context state S_{i+1}. An important remark is that $\forall\ t_i$ managed by P, $t_i \in S_i$ *with* $S_j \geq_a S_{i+1} \Rightarrow t_{i+1}.Time < t_i.Time$. In case of degradation of the *Time* value (whatever be the degradation model, e.g., truncate *Time* value or express it as larger time interval), this partial ordering of *Time* values between consecutive states might lead to refine the time value of the "youngest" tuples belonging to state S_{i+1} given the "oldest" time values contained in states S_j. Let be $Lost_i$ the

time value expressing the maximal lost of accuracy between time values stored in the two consecutive states S_i and S_{i+1}, and *Time_Refinement* the possible refinement of time values stored in tuples in state S_{i+1}. We can determine *Time_Refinement* for each tuple in S_{i+1} given the time delay the tuple was transferred from S_i to S_{i+1}, expressed as d_{trans}=*Current_Date-Transfer_Date*, the current date being *Current_Date* and the date *Transfer_Date* being the date the tuple was transferred from S_i to S_{i+1}. The refinement of the *Time* value available in tuples in state S_{i+1} given the oldest tuples is S_i is bounded in an interval decreasing with time. Thus we have $0 < Time\_Refinement < Lost_i - d_{trans}$ if $Lost_i > d_{trans}$*(Cur_Time)* and $Time\_Refinement = 0$ if $Lost_i \geq d_{trans}$. Interestingly, we see that after a delay equal to $Lost_i$, *Time* values in tuples in S_{i+1} can no more be refined. However, it is interesting to find a way to avoid this problem. Therefore, we propose to add a random delay to transition d_{i+1} leading to S_{i+1} uniformly chosen between $[-Lost_i/2; Lost_i/2]$. Referring to the LCP of Figure 4, the maximal lost $Lost_3$ in term of time accuracy between S_3 (*Time* accuracy is *Hour*) and S_4 (*Time* accuracy is *Day*) is 24 hours. Thus, the *Time* values of tuples recently transferred to S_4 might be refined given the *Time* values of the oldest tuples present in S_3. To prevent this, a random value in hour chosen between [-12, 12] can be added to the "*1 week later*" delay.

Risk 2: The logging process might recover some accurate values. To encompass the one-way degradation property, the logging process must be implanted carefully, to make sure that accurate values might not be recovered from the log files after degradation. First, undo logs are usually required to provide transactional atomicity (i.e., undo aborted transactions) while allowing a *steal* cache management policy (i.e., report modifications to disk before commit) while long transactions are performed. However, the interest of an undo log in our context is limited because only two processes write to the database. One is inserting incoming tuples issued from sensors (insertion are performed by the policy translator, see Figure 1) and the second is degrading tuples content as expressed by LCPs. Both processes operate by short transactions involving a few statements (when not a single). Hence, undo logs might be disabled in our settings. Second, redo logs are required to enforce transaction durability while enabling a *no-force* cache management policy (i.e., changes operated by a committed transaction do not require to be reported to disk immediately, but when appropriate regarding the load on the disk controller). In our context, final states correspond to non-private data which might be stored durably in the database. Those states can thus be logged in the redo. However, logging only final states (and not intermediate states) leads to reduce the smartness of the system in case of media failure, in particular when a very slow degradation process occurs. Instead of using logs to achieve durability, we propose to make use of database redundancy, i.e., we propose to duplicate the intermediate states (including their corresponding LCP) in a second database implementing degradation (without logging), the second database being able to recover intermediate states in case of media failure.

To conclude, policies must be checked at instantiation level against conformity to the one-way property, and the logging mechanisms must be adapted to comply with this property.

5. USER ORIENTED LCP

Besides organization oriented LCPs, user-leveled service acceptance based on the well-known notice and consent strategy (as promoted in worldwide laws and directives) leads to defining individual LCP per user.

Indeed, the list of services a given user consents to determines the LCP regulating her context information. To a certain extend, this LCP might serve as a (quasi) identifier of this user, giving a (fuzzy, because LCPs linked to data are consumed when applied) joining key to gather accurate and degraded context values belonging to this (set of) user(s). This obviously generates an inference problem which is inherent in the degradation model. We will address this in this section.

5.1. Motivating example

In the spirit of notice and consent, LCPs are set by default to prevent from storing any information about the donors. For example, in a road system continuously monitoring location of cars, context states would be dropped immediately when acquired or received.

Services available in the AmI space notify users of the context information they need for offering a given benefit, and ask users to consent adding (driven by the service requirements) intermediate context state(s) to their current LCPs. Users, balancing the loss of privacy and the benefits offered by the service, would possibly consent to additional intermediate state(s). On this compromise basis, services can progressively ask for more data in exchange of additional benefits, which might lead to a rich and highly personalized LCP. Note that each user might consent to a specific pool of services.

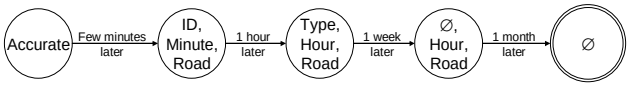

Figure 5. User oriented LCP.

For example, in the car environment, some users could consent to a LCP shown in Figure 5. Such an LCP leads to (i) storing the accurate context state for a few minutes to estimate car speed and traffic overload which are necessary for advising personalized (car *ID* is stored) driving direction efficiently, e.g., avoiding traffic jams, localizing colleagues, etc.; (ii) degrading the accurate location to *Road* and storing it for one hour to enable personalized forecasts (one hour history is stored), e.g., to advise user's colleagues on car pooling possibilities. (Here data is retained for one hour because we assume that people susceptible to make their cars available for car pooling wait for maximum one hour to fill their cars before moving on); (iii) degrading car *ID* to car *Type* and keeping it for one week to enable per type car forecasts (one week history), e.g., providing general car pooling service (e.g., estimating the best places to be given a lift) for the next week; (iv) deleting the *ID* field from the context state triplet to enable general forecasts for the next month, e.g., enabling to plan road directions and avoid traffic jams in advance; and (v) finally removing the whole context state triplet.

5.2. One-way property analysis

Compared to the organization oriented LCPs, which are shared by all the users within an organization, such degradation policies can be unique per user. As such, it might serve as a quasi-identifier of its owner, raising a particular problem regarding the one-way degradation property while progressively degrading the ID field.

Taking the LCP example in Section 5.1 for example, the data file stored at the database level would be similar to the one shown in Table 2, where we represent a LCP by a list of context states s_i (i is an integer) including an empty state $\varnothing$. For ease of explanation, context state transitions are simply represented by an arrow.

In Table 2, four tuples are stored, of which three (i.e., tuple *2*, *3*, and *4*) are in an accurate state, and tuple *1* has been degraded. Accordingly, the LCP associated with tuple *1* is also consumed with only the rest $s_3 \rightarrow s_4$ remained to regulate its further degradation. From the table, it is clear that tuple *4* belongs to *car1*, tuple *2* belongs to *car2*, but 'car' ID of tuple *1* is unclear except the fact that it is a *Van*. However, if one joins on purpose the tuples on the *remaining LCP* field, s/he may suspect that tuple *1*, tuple *3*, and tuple *4* may be from the same donor (the same carID in this case), since they may share the same LCP. This first enables to qualify *car1* and *car2* (regulated by the same LCP) as potential *ID* for tuple *1*, and second enables to discard *car3* (regulated by another LCP). Additionally, the generalization knowledge inherently contained in each LCP might be used to refine the join result. Assuming that *car1* in tuple *1* is planned to be degraded to *Van*, but *car2* in tuple *3* leads to another generalized value, say, *Truck*, so only *car1* remains as a candidate for the accurate *ID* value of tuple *1*.

Thus, while a few car owners share the same LCP and the same generalization knowledge, the degradation of *ID* values could hurt the one-way property.

Row ID	Values of the context triplet (ID, Time, Value)	Current state	Remaining LCP
1	(Van, 15-10-2005 11, street1)	(type, hour, road)	$\{\mathbf{s_3 \rightarrow s_4}\}$
2	(car3, 15-10-2005 12:37, street2)	(id, minute, road)	$\{s_1 \rightarrow s_5 \rightarrow s_6\}$
3	(car2, 15-10-2005 12:36, street3)	(id, minute, road)	$\{s_1 \rightarrow \mathbf{s_3 \rightarrow s_4}\}$
4	(car1, 15-10-2005 12:35, street4)	(id, minute, road)	$\{s_1 \rightarrow \mathbf{s_3 \rightarrow s_4}\}$
…	……	……..	……

Table 2. Data file with per user LCP.

To cope with this issue and specifically prevent from associating tuples via *remaining LCP* field, we propose to decompose an original LCP into several segments, as illustrated in Figure 6, where shaded context states have degraded IDs compared with their precedent ones.

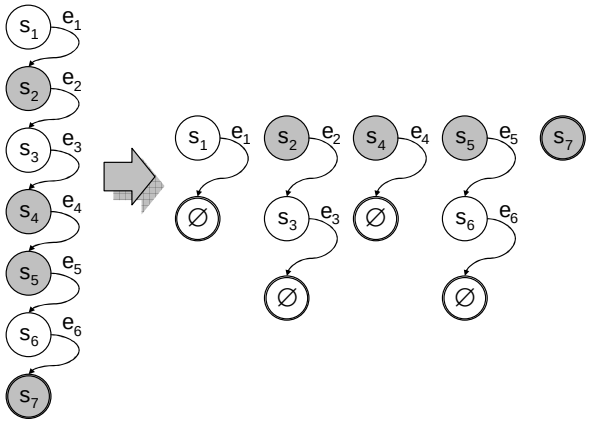

Figure 6. Decomposition of LCP.

Such a decomposition is equivalent to adding one version (context state) of the tuple following a degradation step aiming at generalizing the *ID* value, and associating it with the subpart of the LCP required regulate tuple life cycle until the next ID degradation. In the car example, from tuple 4, an insertion of a degraded version tuple *5* containing (*Van*, *15-10-2005 12*, *street4*) and $s_3 \rightarrow s_4$ as its regulating LCP, and substituting the LCP in tuple *4* by $s_1 \rightarrow \varnothing$. In this way, we prevent from associating tuples *1* and *4*.

Note that such a decomposition is based on a basic property of data generalization. That is, the information offered by a generalized domain is always included in the one revealed at more accurate levels. More precisely, we assume a context state s_i can be generalized to any other state s_i' without hurting individuals' privacy. The privacy requirements of the users can thus be respected. Applied to the considered car example, this would lead to the data file of Table 3.

In fact, such a decomposition is only required when ID value is degraded. The remaining part of the policies (i.e., the degradation steps between two degradations of the ID) does not require any decomposition.

Row ID	Values of the context triplet (ID, Time, Value)	Current state	Remaining LCP
1	(Van, 15-10-2005 11, street1)	(type, hour, road)	$\{s_3 \rightarrow s_4\}$
2	(car3, 15-10-2005 12:37, street2)	(id, minute, road)	$\{s_1 \rightarrow s_5 \rightarrow s_6\}$
3	(car2, 15-10-2005 12:36, street3)	(id, minute, road)	$\{s_2 \rightarrow \varnothing\}$
4	(car1, 15-10-2005 12:35, street4)	(id, minute, road)	$\{s_1 \rightarrow \varnothing\}$
5	(Van, 15-10-2005 12, street4)	(type, hour, road)	$\{s_3 \rightarrow s_4\}$
6	(Truck, 15-10-2005 12, street3)	(type, hour, road)	$\{s_3 \rightarrow s_4\}$
…	……	……..	……

Table 3. Data file after decomposition.

Here, some assumptions must be made regarding the decomposition. First, we require here that complete forms of LCP are not available in the database (e.g., encrypted at the policy binder, decrypted and decomposed at the policy translator with possible buffering of future states). Second, this technique implies that a happening order exists among consecutive transition events in the LCP. For example, the decomposition in Figure 6 assumes that happening of e_2 implies happening of e_1. This requirement holds for general time-based events.

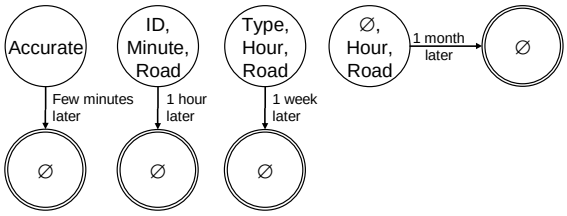

Figure 7. Extreme decomposition.

An interesting remark is that at the extreme, our decomposition is equivalent to inserting several versions of the data in a Hippocratic database [5]. This would be typically the case when decomposing the LCP pictured in Figure 5, as shown in Figure 7.

To conclude, the decomposition technique enables per-user policies to comply with the one-way property when the transition events in a LCP exhibit a certain order. Moreover, this technique is promising in the sense that with decomposition, some common sub-parts of policies shared by many individuals could be dealt with uniformly by the policy binder, which increases the opacity of the privacy requirements per user. To a certain extend, it will lead to k-anonymized LCP patterns.

6. CUSTOMIZED LCP

Both organization and user oriented LCPs can be refined by their owners to reflect special privacy whishes. In particular, a certain owner can complement her LCP with her "individual" personal requirements with extra context states and transitions.

6.1. Motivating example

To illustrate the shape of such an LCP, let's consider the example presented in Section 1. This LCP can be considered as containing an initial subpart either issued by an organization or in a user oriented fashion, plus a user specific subpart coping with particular preoccupations, as shown in Figure 8.

In this case, the third context state of the initial LCP subpart, referred to as (*Group*, *Hour*, *Floor*), will not be reached when the owner of the LCP leaves the building before 5pm. Context information will however be

degraded to the additional user's specific state referred to as (*Group*, *Day*, *Floor*). Such a customization of the LCP would enable the owner to keep private the hour when she left her work, in case she left earlier than expected.

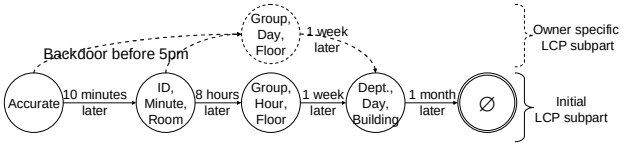

Figure 8. Customized LCP.

Two interesting remarks can be made regarding such a policy. First, while the system knows during one week that the user left incognito by the backdoor (the tuple belongs to a different context state), the exact hour when she left is effectively hidden. Second, the interest of the user to define the additional state referred as (*Group*, *Day*, *Floor*) might be discussed compared to simply forcing to skip not enough degraded intermediate state(s). Indeed, this states offering information than does not exactly corresponds with those required by services based on (*Group*, *Hour*, *Floor*), it might be useless for any application. In that case, direct degradation to the next degraded state fulfilling the particular user's privacy requirement (here, the state referred as (*Dept*, *Day*, *Building*) might be more appropriate. However, we choose here to consider this user defined intermediate state to maximize context information accuracy in the spirit of the usefulness property mentioned in Section 1.

6.2. One-way property analysis

Adding such an LCP subpart to reflect each owner's specific privacy requirements will result in a direct acyclic graph (DAG), complicating the LCP decomposition approach (described in Section 5.2) for one-way property guarantee in case of the ID degradation. This is because the exact degradation path is determined by real-time generated events, and the policy translator cannot prepare the LCP decomposition in advance at tuple's insertion time. For example, in Figure 8, the context state (*ID*, *Minute*, *Room*) can be either degraded to (*Group*, *Day*, *Floor*) or to (*Group*, *Hour*, *Floor*) given the happening event (*the employee leaves through the backdoor* or *at 5pm*).

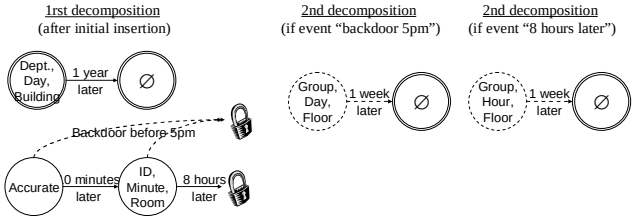

Figure 9. Customized LCP.

To address this issue, we propose to rely on a twofold solution. First, the policy translator only prepares the part of the policy that can be decomposed (i.e., the linear parts of the LCP, containing ID degradation before the first intersection), and opaquely encodes (e.g., encrypts) the remaining part of the LCP. Second, when one of the events revealing the branch to follow happens, the database system is in charge of re-inserting the context state tuple through the policy translator, which will then repeat the operation, i.e., decomposing the proper branch of LCP until the next intersection. Considering the LCP example in section 6.1, this solution would lead to the two decomposition phases pictured in Figure 9.

Note however that such a process is not required or limited to one re-insertion through the policy translator. First, if ID is not degraded, there is no need for decomposition. Second, for context states which are surely to be reached (e.g., the context state (*Dept*, *Day*, *Building*) in the example), the decomposition can still be done in advance. Third, observing that customized user-based events usually involve user's ID itself (e.g., "employee

N231 leaves by the backdoor before 5pm"), after the first ID degradation, user specific degradation may not happen, and in many practical cases only one single re-insertion is necessary.

7. LCP MODEL INSTANCIATION

In this section, we present a brute-force instantiation of the LCP model on top of the Postgres RDBMS. The target of this implementation is to investigate the feasibility of the technique and to make the LCP model more understandable, and not an in-depth performance evaluation.

Our brute-force approach is to pre-compute all degraded fields for each newly incoming context state according to donor-specified life cycle policies, and store them together with the most accurate original ones. When degradation happens, we delete the accurate value from the database by setting *NULL* to the corresponding columns, and keep the less accurate ones. For performance consideration, we use a *fact* table to accommodate different-leveled context states within one tuple. An example of a fact table showing the life cycle of a donor's tuple according to the 3-step degradation policy in Section 4.2 is given in Figure 10. The table contains 4+4+4 attributes representing the three dimensions of the Location cube (i.e., *ID, Time*, and *Value*). Each of the dimensions has 4 different accuracy levels, namely, *second*, *minute*, *hour*, *day* for *ID*; *GUID*, *team*, *dept*, *university* for *Time*; and *coordinate*, *room*, *floor*, *building* for *Value*. When a user's location is sensed, a new tuple (tuple *1*) is inserted into the table with all its columns being filled in. The degradation of the context state from the most accurate one (*ID, Minute, Room*) to (*ID, Hour*, *Room*) leads to modify the column values of *sec* and *min* to *null*. A further degradation to (*Team, Hour, Room*) empties the *GUID* field. The final context state (*Team, Day, ∅*) which will remain durably in the database will have an empty value *null* for the columns of *GUID, sec, min, hour, coor, room, floor*, and *building*.

To examine the cost of the degradation strategy, we conduct a set of simulation experiments. To know when and which tuples need to be degraded and how to perform the degradation, we adopt an additional *script table* to record a list of SQL update statements for emptying columns (as shown in the above example tuple) due to accuracy degradation. These statements will be executed later whenever the corresponding triggering events happen. In our simulation study, we simulate the events' happening using a random number, attached with each original context tuple. The overall workload of the database system therefore consists of users' normal query requests, plus insertion of new context tuples (consistently sensed by the AmI space) as well as their degradation updates. We interleave database query requests with insertion and update operations as follows.

Every new tuple insertion is followed by Q number of queries. After inserting U tuples and $Q*U$ queries, $S*U$ number of updates is performed to degrade the inserted tuples (where S is the average number of LCP degradations). In total, N number of new tuples is inserted in the database, and the total execution time consumed is T. The number of inserts per second can now be calculated as N/T, and the number of queries per seconds as $N \times Q/T$. Considering that querying is one of the most fundamental operations in any database system, the performance measure we used is the number of queries per second (i.e., *query throughput*) which can be executed with a given number of inserts per second (i.e., arriving rate of context data).

Tuple No	ID				Time				Value			
	GUID	team	dept	uni	sec	min	hour	day	coor	room	floor	building
1	123	2	31	1	21-11 14:30:29	21-11 14:30	21-11 14	21-11	2	3	2	1
	↓											
	123	2	31	1	*null*	*null*	21-11 14	21-11	*null*	3	2	1
	↓											
	null	2	31	1	*null*	*null*	21-11 14	21-11	*null*	3	2	1
	↓											
	null	2	31	1	*null*	***null***	***null***	21-11	*null*	*null*	*null*	*null*

Figure 10. An example fact table accommodating the life cycle of a context tuple.

We compared the performance with that of a traditional database system without degradation policies, whose main workload consists of querying and inserting into the fact table. In our tests, we set *U=5*, *N=5000*, $S \in \{1,2,3,4\}$ and vary the value of *Q* over [0…64]. We ran the tests on the databases filled with 30K, 200K, 400K, and 750K tuples. From the results presented in Figure 12, we make a few observations. First, the performance of the system is negatively influenced due to the extra updating workload for degradation. This is not surprising. The higher the data insertion rate and the more complex the degradation policies, the worse the performance (Figure 12 (a)(c)). Second, a larger database incurs a higher cost than a small-sized database (Figure 12 (b)). Efficient indexing and querying techniques, confronted with a high volume of streaming data insertion, are thus very desirable to make the LCP model really practical. Third, comparing our LCP decomposition approach with non-decomposition one, although the former incurs more insertions, there is hardly any difference in performance (Figure 12 (d)), which confirms that this approach to comply with the one-way property almost delegate the additional cost to the policy translator, without inducing additional load for the database itself.

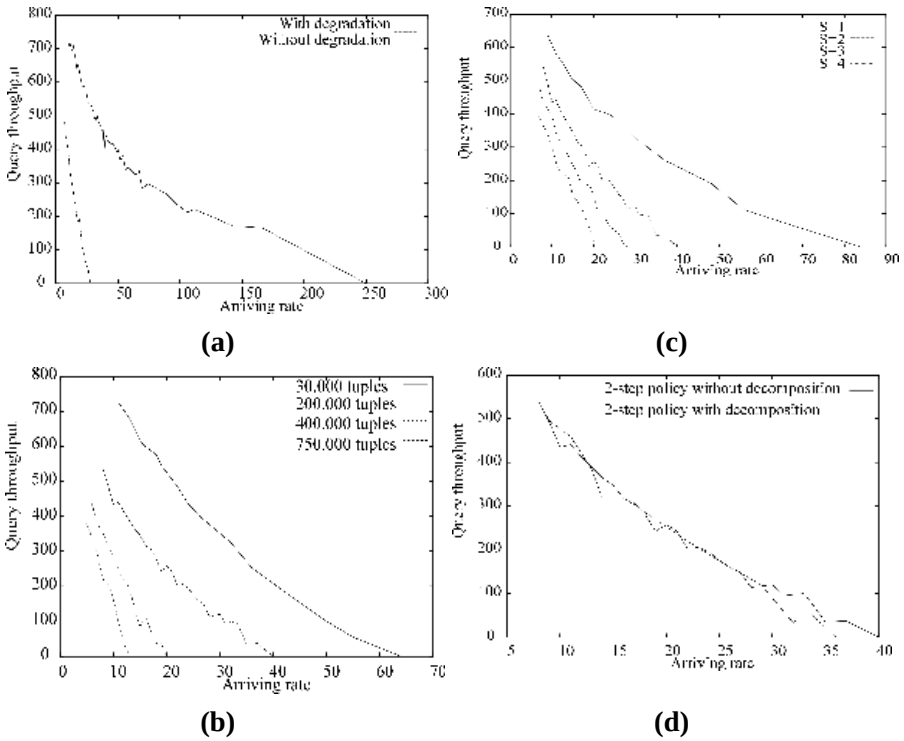

Figure 11. Costs of the degradation.

8. NEW CHALLENGES

Having shown our LCP approach to the important privacy problem for the AmI, we now proceed to a summary of further interesting research challenges and open problems which we consider being of primary interest.

Multi-usage. The current LCP study is based on a linear generalization model for a context cube. However, since generalization/specification hierarchies are often application dependent, the generalization trees might have several branches, potentially one required per application. Our model needs to be adapted to rich generalization schemas so as to cover a broad range of applications. One potential way to resolve this issue could be based on duplicating information, which must follow different generalization paths for different applications. Also other degradation techniques might be envisioned (e.g., progressively deleting bits) for specific attributes (e.g., IP addresses collected by network access points in the soft meeting planner settings).

Trustworthiness of the approach. Although we admit that enabling privacy can not solely rely on our approach, the LCP model offers a resistance to a posteriori (i.e., after LCP appliance) snooping attacks which constitute a major threat in the AmI. Note that a posteriori privacy violation can be performed against individuals by malicious organisms before credit or insurance acceptance, or against companies before financial operations, insider trading, etc. For the irreversible degradation, such a protection can be achieved by only applying the LCP model once against context data. However, it is worth mentioning that we only addressed here privacy-enabled systems but not privacy-enforced systems. That is, without any additional security feature, our proposal requires trust in the underlying data management system that really degrades the data. In particular, attacks can also be conducted by altering (even randomly) data involved in LCPs, replacing some LCPs with other valid (and weaker) LCPs or by a previsously defined (weaker) LCP defined by the same user. How to enforce proper degradation process, e.g., by means of cryptography or secure hardware, is a very interesting and difficult open issue. Another perhaps more tractable problem would be to detect misuse at the Context-DB level, based on database audit techniques (like monitoring database events, incoming queries, and produced results). Indeed, in our current settings, nothing can prevent a malicious third party holding sufficient access privileges to continuously query the fresh (accurate) subsets of the context DB in order to constitute the complete accurate history of the AmI space.

Distribution. As mentioned in Section 1 and 2, AmI spaces incorporate many components. Moreover, several contiguous AmI spaces might coexist. Such a distribution nature introduces many challenging issues. One of them is related to event detection (for triggering context degradation in the LCP model). Although some local events (monitored within the internal AmI space) might be broadcast to any internal recipient responsible for managing context data associated with the LCP, it is difficult to assume that external events (monitored in another AmI space) will be. In fact, events themselves constitute parts of context information, and as such may be regulated by LCPs leading to their own progressive degradation (which can occur rapidly). This will definitely prevent intensive external broadcast. Besides, through events, some information which should not be disclosed to external AmI spaces might be revealed. We are currently resolving this issue, notably in term of possible mapping of external events to internal/universal ones. In addition to event detection, distribution also incurs some other difficult issues like management of knowledge of LCP and knowledge of the appropriate generalization hierarchy for any surrounding (visitor) user, etc.

Querying multi-accuracy data. The LCP model leads to the management of multi-accuracy information leading to further interesting research issues, including query language and processing techniques to cope with this multi-accuracy data. Notably, a query language is necessary to enable applications to express queries involving different granularities of data provided by the context cubes (e.g., in an SQL-like language, *SELECT Id[department] FROM Location_cube WHERE Value[room] = "thisRoom";*). Also, accelerating computation techniques on data samples, in the same spirit as [7], could benefit from the different accuracy levels to present to applications results with more or less precision given the current load of the database system (assuming less accurate results, based on more degraded data, are faster to compute). More generally, query processing and

index techniques should be devised to tackle special properties of multi-accurate data in insertion-intensive DBMS.

One-way data degradation vs. dynamic computation. While we promote here one-way (irreversible) data degradation, it would also be interesting to investigate some other degradation schemes based on dynamic computation of current context states. For instance, one might imagine going backward along a generalization hierarchy. This could be studied by adopting a multi-accuracy access control strategy. Indeed, while traditional systems propose access control policies based on selections and projections, we argue that this approach is not flexible enough to fully satisfy donors' particular wishes when applied to pervasive environments like AmI, where control should be given at different data accuracy, or offer micro-views as shown in [6], and should also be able to evolve with the time. In the spirit of the LCP model, the provided accuracy may either degrade or upgrade under certain circumstances, e.g., current location can be considered as private and become accessible one week later.

Static vs dynamic life cycle models. Addressing the limitation of static models would lead to delegate further intelligence to AmI components (e.g., database), to enable to compute dynamically (i.e., at runtime) the most appropriate next context state given the current database content. For example, we could imagine to define degradation policies by means of k-anonymization wishes (e.g., each ten minutes, anonymize my context 5 time more, taking into account that I would prefer degrading in favor of ID, then Time, and then context accuracy), delegating to the system the determination of the most appropriate degraded state, given the current database content.

9. CONCLUSION

AmI environments continuously monitor surrounding individuals' behavior to make existing applications smarter, i.e., make decision without requiring user interaction. While this smartness ability is tightly coupled to the quality and quantity of the available (past and present) information, context linked to surrounding individuals falls under privacy directives. This paper proposed a new approach to deal with this paradox between the smartness and privacy for the AmI, based on progressive degradation of the accuracy of context information. We believe that providing control over context data's life cycle is crucial for the AmI, which invisibly and consistently acquires data with a weak durability requirement (from donors' point of view).

We defined the baselines of a Life-Cycle Policy (LCP) model enabling such data degradation in a database managing context information. We showed by examples the interest of the proposed LCP model regarding the privacy requirements from both an organization holding the AmI space and surrounding individuals moving in this AmI space. We also analyzed the potential problems regarding the one-way property (i.e., while performed, degradation must be irreversible, even by the system) of the model, and showed the feasibility of our approach on top of the traditional database system - Postgres. As a strategy to tackle the important privacy problem in the AmI, the LCP model itself also leads to a number of further research issues which were also described in the paper. We are convinced that this LCP model might pave the way to new solutions to respect individuals' privacy in autonomous systems.

We are currently integrating the LCP model into our *Soft Meeting Planner* prototype, which demands serious privacy and smartness in planning an appropriate soft meeting among different people.

10. REFERENCES

[1] Adam N. R., Worthmann J. C. Security-control methods for statistical databases: a comparative study. ACM Computing Surveys 21(4), 1989.

[2] Aggarwal G., Bawa M., Ganesan P., Garcia-Molina H., Kenthapadi K., Mishra N., Motwani R., Srivastava U., Thomas D., Widom J., and Xu Y. Vision Paper: Enabling Privacy for the Paranoids. VLDB 2004.

[3] Agrawal R., and Srikant R. Privacy-Preserving Data Mining. SIGMOD Conference, 2000.

[4] Agrawal R., Kiernan J., Srikant R., Xu Y. An Implementation of P3P Using Database Technology. EDBT, 2004.

[5] Agrawal R., Kiernan J., Srikant R., Xu Y. Hippocratic Databases. In Proc. of the Int'l Conference on Very Large Databases, 2002.

[6] Byun J-W, Bertino E. Vision paper: Micro-views, or on how to protect privacy while enhancing data usability. CERIAS Tech Report 2005-25, Purdue University, IN 47907-2086, 2005.

[7] Chandrasekaran S., Franklin M. J. Remembrance of Streams Past: Overload-Sensitive Management of Archived Streams. VLDB, 2004.

[8] Chaudhuri S., Dayal U. An Overview of Data Warehousing and OLAP Technology. SIGMOD Record 26(1), 1997.

[9] Chawathe S., Krishnamurthyy V., Ramachandrany S., Sarma S. Managing RFID data. In Proceedings of VLDB Conference, 2004.

[10] Cranor L., Langheinrich M., Marchiori M. A P3P Preference Exchange Language 1.0 (APPEL1.0). W3C Working Draft, 2002.

[11] Cranor L., Langheinrich M., Marchiori M., Presler-Marshall M., Reagle J. The Platform for Privacy Preferences 1.0 (P3P1.0) Specification. W3C Recommendation, 2002.

[12] CSI/FBI. Computer crime and security survey, 2005. http://i.cmpnet.com/gocsi/db_area/pdfs/fbi/FBI2005.pdf

[13] Dittrich, J-P., Fischer P. M., Kossmann D. AGILE: Adaptive Indexing for Context-Aware Information Filters. SIGMOD Conference, 2005.

[14] Doom C. Get Smart: How Intelligent Technology Will Enhance Our World. Computer Sciences Corporation, 2001.

[15] ECHISE. First International Workshop on Exploiting Context Histories in Smart Environments, collocated with PERVASIVE, 2005.

[16] European Parliament. Directive 2002/58/EC of the European Parliament and of the Council of 12 July 2002 concerning the processing of personal data and the protection of privacy in the electronic communications sector, 2002.

[17] European Parliament. Directive 95/46/EC on the Protection of Individuals with Regard to the Processing of Personal Data and on the Free Movement of Such Data. 24 October 1995. www.cdt.org/privacy/eudirective/EU_Directive_.html

[18] Hong J. I., Landay J. A. An Architecture for Privacy-Sensitive Ubiquitous Computing. MobiSys, 2004.

[19] ISTAG. Orientations for WP2000 and beyond. Final Report, 1999. ftp://ftp.cordis.lu/pub/ist/docs/istag-99-final.pdf

[20] Lahlou S., Langheinrich M., Rücker C. Privacy and trust issues with invisible computers. Communication ACM 48(3), 2005.

[21] Langheinrich M. A Privacy Awareness System for Ubiquitous Computing Environments. Ubicomp, 2002.

[22] Langheinrich M. Privacy by Design - Principles of Privacy-Aware Ubiquitous Systems. Ubicomp, 2001.

[23] Lederer S., Mankoff J., Dey A. K. Who wants to know what when? privacy preference determinants in ubiquitous computing. CHI Extended Abstracts, 2003.

[24] OECD. The International Legal Framework for Data Protection and its Transportation to Developing Transitional Countries. December 28, 2004. http://www.internetpolicy.net/privacy/20041228privacy.pdf

[25] Orr R. J., Abowd G. D. The Smart Floor: A Mechanism for Natural User Identification and Tracking. In Proceedings of the Conference on Human Factors in Computing Systems (CHI), 2000.

[26] Pipe R. China Launches Privacy Protection Law Project. 2005. http://www.privacyexchange.org/japan/Chinaprivacy.pdf

[27] Post E. R., Orth M., Russo P. R., Gershenfeld N. Ebroidery: Design and fabrication of textile-based computing. IBM Systems Journal, 39(3&4), 2000.

[28] Potonniée, O. A decentralized privacy-enabling TV personalization framework. European Conference on Interactive Television, 2004.

[29] Samarati P., Sweeney L. Generalizing data to provide anonymity when disclosing information (abstract). ACM Symposium on Principles of Database Systems, 1998.

[30] Samarati P., Sweeney L. Protecting privacy when disclosing information: k-anonymity and its enforcement through generalization and suppression. IEEE Symposium on Research in Security and Privacy, 1998.

[31] Shnayder V., Chen B., Lorincz K., Fulford-Jones T. R. F., Welsh M. Sensor Networks for Medical Care. Harvard University Technical Report TR-08-05, 2005.

[32] Spence M., Driver C., Clarke S. Sharing Context History in Mobile, Context-Aware Trails-Based Applications. ECHISE, 2005.

[33] Sweeney L. Achieving k-anonymity privacy protection using generalization and suppression. Int. Journal on Uncertainty, Fuzziness, and Knowledge-based Systems, 10(5), 2002.

[34] Sweeney L. k-anonymity: a model for protecting privacy. Int. Journal on Uncertainty, Fuzziness and Knowledge-based Systems, 10(5), 2002.

[35] United Nations. Guidelines on the use of computerized personal files. Economic and Social Council, resolution 1990/38 of 25 May, 1990.

[36] USC. The Privacy Act, 5 U.S.C. § 552a, 1974. http://www.usdoj.gov/04foia/privstat.htm.

[37] Verykios V. S., Bertino E., Fovino I. N., Provenza L. P., Saygin Y., Theodoridis Y. State-of-the-art in privacy preserving data mining. SIGMOD Record 33, 2004.

[38] Wang F., Liu P. Temporal Management of RFID Data. Industrial session, VLDB, 2005.

[39] Warneke B., Last M., Liebowitz B., Pister K. S. J. Smart Dust: Communicating with a Cubic-Millimeter Computer. IEEE Computer 34(1), 2001.

[40] Wilson D. H., Wyatt D., Philipose M. Using Context History for Data Collection in the Home. ECHISE, 2005.