Context Sensitive Privacy Management in a Distributed Environment

Grzegorz Gołaszewski and Janusz Górski

Gdańsk University of Technology, ul. Narutowicza 11/12
80-233 Gdańsk, Poland
{grzo,jango}@eti.pg.gda.pl

Abstract. The paper presents a mechanism for privacy management developed for a distributed environment with the assumption that the nodes are subjected to severe resource constraints (processing power, memory). The basic idea is that the private data are filtered out in accordance with users' privacy policies before they become visible to other users. The decisions are highly localized which reduces the load related to privacy management on the computing nodes. The mechanism is hidden in middleware (the platform) and is transparent to the applications running on the nodes. The paper describes the problem and its solution in abstract terms and then presents the technical system which has been developed to demonstrate the proposed solution.

Keywords: privacy management, context dependence, distributed systems, mobile systems, embedded systems, Angel.

1 Introduction

Privacy protection is presently one of the major concerns related to applications of modern information technologies. Wireless, mobile and embedded applications pose multiple new threats related to privacy, including new types of data which become widely available (e.g. location), potential for automatic gathering of data without user's consent (e.g. by means of various types of sensors) and automatic linking of these data with users' identities (by means of RFIDs, mobile phones, smartcards and other personally carried tokens). One of the problems related to such dynamic and distributed environments is that privacy is highly context dependent meaning that the same data item can be subjected to different privacy requirements (from the data owner's viewpoint) depending on the context within which the data is potentially disclosed.

This paper addresses the problem of context dependence of privacy requirements with respect to environments where users have access to mobile and stationary terminals which can display their private data. The terminals are cooperating with gateways by calling services through which private data can 'leak' to terminals. The gateways are under control of (possibly multiple) users and run applications serving the specific application objectives (for instance, health monitoring, lifestyle support, home automation etc.). The gateways gather data sensed by wireless sensors, including body parameters, environmental parameters etc. and process them according to the needs.

In such an environment, the problem we are dealing with in this paper is that data displayed on a terminal may undergo different privacy restrictions depending on the context of the terminal owner is currently in (basically, the contexts differentiate between different groups of subjects being able to see the data displayed on the terminal). These restrictions have their roots in the privacy policies of data owners.

In the paper we assume that such privacy policies are known. In the presented application of the method we provide an explicit mechanism by which the data owners can define their privacy policies. However, the presented mechanism would work also in the situation where the policies are acquired in a more indirect way.

The mechanism we present has two distinguishing features. Firstly, it controls interfaces through which the applications running on the gateways and the terminals cooperate. And it builds into each interface a sort of 'valve' which control leaking of the private data depending on the current context. Secondly, it is localized in the architecture of the deployed software in a way which keeps most of the software 'unconscious' of the presence of the mechanism (making it transparent to the running applications).

The paper is structured as follows. Section 2 introduces the problem, the related assumptions and constraints. Section 3 describes our solution to the problem and section 4 gives details on how this solution was demonstrated within the context of the Angel[1] project. In section 5 we compare our work with the work of others. Section 6 concludes the work.

2 Problem Statement

2.1 The Problem

Private data belong to its *owner. Privacy* is understood as the right of the data owner to decide about disclosure, storage and processing of his/her private data. This right may be implemented in different ways: by providing the data owner with means to indicate potential receivers of the data, to accept the purpose of data collection before the data being actually stored/processed and to designate the situations in which the data can be disclosed etc. For instance, Tom may implement his privacy right by deciding that his location should be confidential except that it may be disclosed to an ambulance service in case of medical emergency.

We assume that different situations for which the owner sets restrictions on his/her private data disclosure can be enumerated and we call them *contexts*. The context a given person is actually in is called her/his *current context.* In the above example, Tom defines restrictions related to his private data with respect to the context "medical emergency". Other contexts are possible for Tom, like "at home", "jogging", "having guests" etc. Tom decides that for these contexts his location remains confidential.

We assume that each data owner defines his/her *privacy policy*. For each possible current context, the policy specifies the receivers authorized to obtain the owner's

[1] 6th FR STREP *Advanced Networked embedded platform as a Gateway to Enhance quality of Life* (ANGEL), Contract number IST-033506.

private data. For instance, the privacy policy of Tom includes the requirement: *Tom's location can be disclosed to* ***ambulance service*** *in the* ***medical emergency*** *context.*

We assume the following model of the system:

- the system includes multiple *gateways*,
- each gateway is controlled by one or more system *users,*
- each gateway hosts a number of *applications*,
- a distinguished application called *platform* is permanently deployed on each gateway,
- applications can disclose data to data *receivers* through *interfaces*,
- data receivers are applications or *user terminals,*
- system behaviour is modelled as a sequence of system *states,*
- for each interface, the potential scope of data disclosed by the interface at a given state to different receivers can be determined statically (i.e. is known before the system starts).

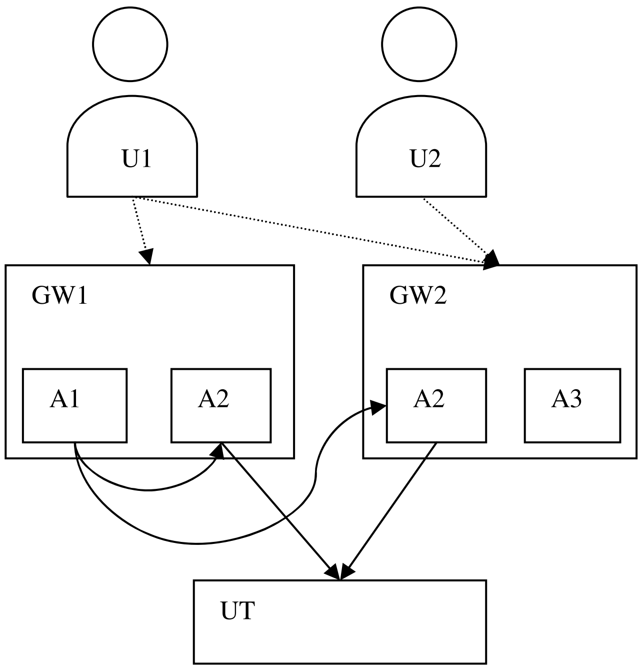


Fig. 1. An example of the system

In Fig. 1 an example system is presented. It consists of two gateways (GW1, GW2), three applications (A1, A2, A3) and a user terminal (UT). A1 and A2 are deployed on GW1 and A2 and A3 on GW2. GW1 is controlled by one user U1 and GW2 is controlled by two users, U1 and U2, which is represented by dotted arrows. Potential data flows are represented by solid arrows. These arrows originate in interfaces of the corresponding applications.

The problem statement is as follows.

> *To ensure that for each system state, data disclosure on system interfaces observes the restrictions imposed by the privacy policies of the data owners.*

Let us imagine that Tom is using a system which integrates mobile devices, including Tom's PDA (equipped with GPS) and the mobile terminal in an ambulance. In an emergency situation, Tom's PDA connects to the ambulance to call for help. In such context, the problem is how to enforce Tom's privacy policy, and make sure that Tom's location will remain confidential in all other situations except his health emergency.

2.2 Assumptions and Constraints

Before we present our solution to the above problem, we introduce several additional assumptions which constraint the problem solution space.

- For each data owner, the corresponding privacy policy distinguishes different private *data types*.
- For each interface, the scope of private data possibly released through the interface is determined at system configuration time. At run time the interface either releases all data or nothing (i.e. partial disclosure is impossible).
- For each gateway, the applications deployed on the gateway may attempt to disclose (through corresponding interfaces) only the private data of the users of the gateway.
- For each system state, the current context of each user can be determined.
- For each user, the contexts are mutually exclusive (i.e. at any time, the user is in exactly one current context); however, for different users, their current contexts can differ.
- Platform does not disclose private data. It is used for system management purposes and provides support for other applications.

In addition, we impose the following constraints on the solution sought for the problem:

- the privacy policies are defined referring to contexts and data receivers (i.e. the system structure remains transparent at this abstraction level),
- the effort required from application developers to implement the privacy policies should be minimized.

3 The Solution

3.1 General Idea

To control the users' private data disclosure we propose a mechanism that 'filters out' (depending on the current context) the data being displayed through system interfaces in accordance with the user's privacy policy requirements. This filtering process follows the changes in current contexts of the users and guarantees that the private data is not disclosed if so required by the user's privacy policy.

This filtering is effected on each interface through which private data are delivered to data receivers. The data filtering decisions are controlled by the *canAccess* (partial) function which has the following signature:

$$canAccess\colon O \times R \times C \times I \nrightarrow Boolean\,, \tag{1}$$

where *O* denotes a set of data owners, *R* represents a set of data receivers, *C* is a set of contexts and *I* is a set of interfaces. The function returns a Boolean value which indicates whether a given interface *i* can disclose data of the owner *o* to the receiver *r* in the current context *c*. In other words, *canAccess* decides about 'opening' or 'closing' a given interface in a specific context and for specified owner and receiver.

The function *canAccess* is derived from the privacy policies specified by the users. At system runtime, the function is continuously evaluated to adjust private data filtering to the changing contexts of the users.

To provide for efficiency of evaluation of *canAccess* values, several more specific data structures and algorithms have been proposed.

3.2 Privacy Policies

We assume that the privacy policy of a system user is represented by a *privacy matrix.* The matrix specifies the private data types to be visible to different receiver categories in different contexts. This specification is given by each data owner by setting corresponding controls in the *privacy matrix* ('√' means 'disclose' and '–' means 'do not disclose'). An example privacy matrix is shown in Table 1.

Table 1. Privacy matrix

	ReceiverCategory1		ReceiverCategory2		ReceiverCategory3	
Context1	*DataType1*	√	*DataType1*	√	*DataType2*	√
	DataType2	-	*DataType3*	-	*DataType4*	√
	DataType3	√	*DataType4*	-		
Context2	*DataType1*	-	*DataType1*	√	*DataType2*	-
	DataType2	√	*DataType3*	√	*DataType4*	√
	DataType3	-	*DataType4*	-		

Table 1 shows a privacy matrix as it is seen by a single user. Initially, the matrix includes one *default* context which is the current context of the user. Additional contexts can be added/removed by the user (the contexts are enumerated in the first column of the table). For each context, the relevant receiver categories and private data types are given. The user can declare her/his privacy policy by setting corresponding controls in the table.

In the example presented in Table 1, the user decided that while being in *Context1*, data types *DataType1*and *DataType3* are to be disclosed to *ReceiverCategory1* whereas access to *DataType2* is to be prohibited. In the same context, the data visible to *ReceiverCategory2* is just *DataType1*.

For instance, if *DataType4 = Health information*, *ReceiverCategory2 = Friends* and *ReceiverCategory3 = Family*, the choices represented in Table 1 mean that Tom does not want his friends to know his health details whereas he accepts that family members can have full access to this information.

More formally, we can represent the privacy policies as a function *PrivPol* with the following signature:

$$PrivPol\colon O \rightarrow \mathbb{P}\,(C \times RC \times DT \times Boolean)\,, \tag{2}$$

where *RC* is the set of possible receiver categories and *DT* is the set of possible private data types.

The mapping of private data receivers to receiver categories is specified (by the private data owner) by means of the auxiliary *mapping* function.

$$mapping\colon O \nrightarrow RC \times R\,. \tag{3}$$

Note that if a gateway *g* is controlled by users *u1* and *u2* (e.g. a gateway collecting and displaying the house environment parameters where *u1* and *u2 are inhabitants of the house*), a possibility of privacy policy conflict arises. Consider a situation where *dt* is a private data type of both, *u1* and *u2,* and the users specify in their privacy policies conflicting requirements related to releasing *dt* through interfaces of *g,* respectively for context *c1* (for *u1)* and *c2* (for *u2*). Then, if in a given state the current context for *u1* is *c1* and for *u2* is *c2*, we have conflicting requirements for interfaces of *g*.

In general, such conflict can be resolved by one of the strategies listed below:

- release *dt* if both, *u1* and *u2,* allow for this in their privacy policies (seeking for consensus),
- release *dt* if either *u1* or *u2* allow for this in their privacy policies (in case of more users, this can be generalized as a threshold based decision),
- let the data receiver to specify the intended owner of *dt* and then decide in accordance with the owner's policy (conflict elimination).

The third strategy eliminates the conflict by narrowing the scope – relating the decision about releasing *dt* to just one owner (either *u1* or *u2*). In such case, if *u1* says 'disclose *dt*´ and *u2* says 'do not disclose *dt*', the request 'give me *dt* of *u1*' issued to an interface of *g* will return a data of type *dt* whereas the request 'give me *dt* of *u2*' will return an empty value.

In the mechanism described below we are following this conflict elimination strategy. Nevertheless, implementation of other strategies is also possible.

3.3 System Configuration

We assume that the following information related to system configuration is available.

For each application deployed in the system, a list of its interfaces and for each interface the list of *private data types* exposed by the interface are determined at system configuration time. And for each interface, the list of all possible *receiver categories* is given.

The above information is represented by the *appInfo* function, which for each application specifies a set of triples, where each triple include: an interface exposed by the application, private data types exposed by the interface, and receiver categories envisioned as data consumers for the interface:

$$appInfo\colon A \rightarrow \mathbb{P}\,(I \times \mathbb{P}\,DT \times \mathbb{P}\,RC)\ . \tag{4}$$

In addition, it is assumed that the following information is maintained at system runtime:

- identification of gateway users:

$$own\colon Gw \leftrightarrow O\ , \tag{5}$$

  where *Gw* is a set of all gateways,
- identification of the applications deployed on each gateway:

$$conf\colon Gw \leftrightarrow A\ , \tag{6}$$

  where *A* is the set of all applications deployed in the system.

Note that the above information can be used to automatically calculate the privacy matrix of each user in the form shown in Table 1. Initially, the user defined entries of the matrix are set to "do not disclose" default value.

3.4 Making Privacy Related Decisions

The privacy related decisions are made by continuously evaluating the value of the *canAccess* function in order to decide about private data release through system interfaces. The function is evaluated from the information submitted and maintained in the system (described in sections 3.2 and 3.3).

First, an auxiliary *canAccess'* function is calculated from *PrivPol*, *appInfo* and *conf* functions. It has the following signature:

$$canAccess'\colon O \times RC \times C \times I \nrightarrow Boolean\ . \tag{7}$$

Values of *canAccess'* are generated following the folowing rule: in a given context, an interface can disclose data to a given receiver category only if the receiver category has the permission (given in the privacy policies of the data owner) to obtain, in this context, all data types potentially disclosed by this interface.

For example, assume that Tom is doing physical exercises (the current context of Tom). Assume also that the treadmill he uses acts as a gateway under his control and exposes interface *i* used for exercise monitoring. The treadmill publishes data such as speed, distance, but also pulse and heart rate. Therefore the data published by *i* belong to both *Exercise information* and *Health information* data types. Then, if Tom wants his coach to gain access to his *Exercise information*, he has to give him/her access to his *Health information* as well. To enable Tom to separate the *Exercise information* and *Health information* data these data types should be released through separate interfaces. In present implementation of the privacy management mechanism such decision is to be made at application development time.

In case a user redefines his/her privacy policy or the system configuration changes, *canAccess'* function has to be recalculated.

From *canAccess'* and *mapping* functions we calculate values of *canAccess* following the principle that a receiver *r* can obtain data from an interface *i* if *r* belongs to at least one receiver category *rc* authorized by the data owner *o* to see his/her private data through this interface.

Following the example of Tom's exercise monitoring, let us assume that Tom uses a treadmill which exposes his *Exercise information* through interface *i*. In his privacy policy Tom lets the receiver category *Coach* to view this data. At the same time Tom denies seeing this data by the category *Friends*. In such situation Helen, a friend of Tom, would receive empty response if she tries to call the interface *i* from her user terminal. However, John, a friend of Tom and simultaneously his trainer would receive Tom's exercise data. Also Tim, who belongs to the coaching team, will have access to this data, even if he is not a friend of Tom.

3.5 The Platform

To release applications from managing privacy and to localize privacy-related decisions we use the concept of the *platform*. Platform is an application that is permanently deployed on each gateway and is charged (among others) with making privacy related decisions. The process goes as follows.

When an application receives a request to release private data through its interface, it forwards this request to the platform with the following descriptor:

<data receiver ID, called interface ID, data owner ID>.

For instance, *<John, i, Tom>* means that *John* requested from *i* data belonging to *Tom*. The platform uses the received descriptor along with information on the current context of *Tom* to evaluate *canAccess* function and responds with a Boolean decision on if to serve the request or to ignore it.

For efficiency reasons, *canAcess* can be pre-calculated and its relevant part can be stored on each gateway together with the current context information for the gateway user. In case this information changes (e.g. privacy policy change, system configuration change etc.) the new values are re-distributed to the gateways. Note that it is sufficient to restrict to the current contexts of the users controlling given gateway and maintain only this part of *canAccess* which relates to the privacy policies of these users.

The result of the above solution is that the privacy management is almost transparent to applications. The only localities where privacy management needs care are application interfaces where a simple coding protocol is to be followed: call the platform before releasing data through the interface (asking for a permission).

4 Proof of Concept

The mechanism described above was developed within the context of Angel project. The project objective was to develop and to demonstrate the wireless sensor network

(WSN) based platform integrating WSN enabling technologies (in particular Zigbee [1]) and Internet technologies for health and lifestyle supporting applications.

4.1 Privacy Requirements

In Angel the privacy protection requirements were derived following a three step process:

1. analysis of the relevant source documents like Directive 95/46/EC [2], Directive 2002/58/EC [3], OECD Privacy Guidelines [4], HIPPA Privacy Rule [5] etc. to derive *generic privacy requirements*,
2. mapping the generic requirements on the Angel application scenarios to derive *system level privacy requirements*,
3. mapping the system level privacy requirements on the Angel platform to derive the *platform level privacy requirements*. These mappings were maintained to provide for traceability of privacy requirements.

An example platform level privacy requirements are as follows: PR1: platform should enable identification of patient's health information, PR2: platform should allow for private data purge, PR3: platform should ensure confidentiality of privacy-related data over all communication routes, PR4: platform should ensure integrity of privacy-related data over all communication routes.

4.2 The System

The concept of *Angel system* has been defined comprising of three main logical components: *smart node*, *Angel gateway* and *Angel service terminal*.

Smart node is a small device with limited computational power, wireless networking capability and low energy consumption. Smart nodes are used for collecting data in two kinds of networks: Body Sensor Network (BSN) and Home Area Network (HAN). BSN comprises of wearable sensor nodes (like heart-rate sensors or pedometers), while HAN integrates nodes measuring habitat parameters, e.g. light intensity, temperature, humidity etc. In both cases, the nodes communicate gathered data via WSN protocol to an Angel gateway.

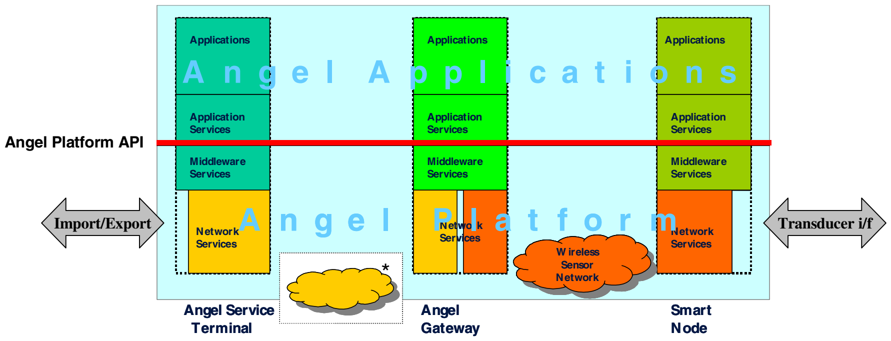


Fig. 2. The Angel Reference Architecture [6]

Gateway is a device with more computational power, equipped with both, WSN interface and IP interface. The data is collected on the gateway by dedicated applications. To disclose the gathered data, Angel applications use Web Services.

Data is disclosed to Angel service terminals, which are (logical) devices capable of receiving data from the gateways and provide related services. As system components are logical, it may happen that a terminal and a gateway reside on the same physical device (e.g. PDA) although it is not necessary. For instance, an IP television set-top box equipped with a WSN interface simultaneously hosts a gateway and a user terminal. In such case the set-top box is capable of collecting data and forwarding it to other Angel service terminals through Internet, as well as displaying the collected data on the TV-set connected to it.

All Angel enabled devices host a software layer providing common services for Angel applications (the middleware). Angel reference architecture is shown in Fig. 2 [6].

4.3 Privacy Subsystem Architecture

Our proof-of-concept implementation of the privacy management mechanism is shown in Fig. 3.

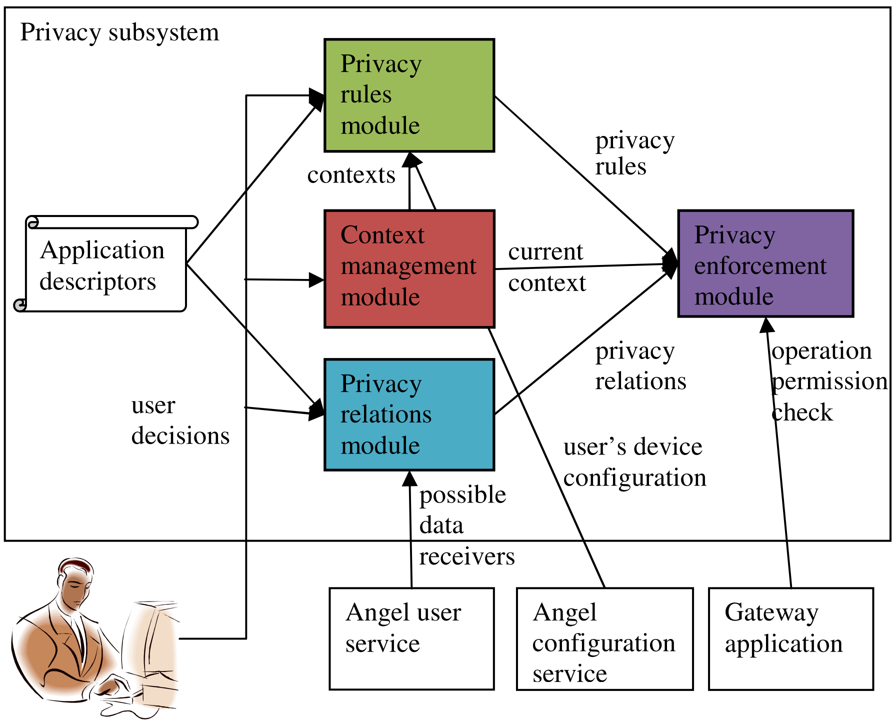


Fig. 3. Angel privacy subsystem logical architecture

Privacy rules module is responsible for constructing a privacy matrix, allowing user manipulation of the matrix, determining privacy rules (*canAccess'* function (7)) and distributing them to gateways.

Context management module is responsible for managing user contexts. This includes managing current context change (in the present implementation the context switching was left to the users; nevertheless it can be delegated to an automatic context detecting mechanism).

Privacy relations module manages receiver categories and their relation to individual Angel users.

The above three modules feed their output to *Privacy enforcement* module, which is responsible for enforcing privacy policies.

Note, that although the security subsystem is not depicted in Fig. 3, the privacy subsystem relies on it in several aspects: users and data receiver authentication, secure communication, and so on. However, these aspects are not discussed here, as they are outside the scope of this paper.

4.4 Privacy Related Information

To perform their tasks, the privacy subsystem modules need to be fed with proper information. Data carrying this information is received from other Angel subsystems, from users' input or from system configuration files.

The *appInfo* function (4) is provided in the form of *application descriptors*. An application descriptor is an XML file describing privacy properties of a single Angel application. A sample application descriptor is presented below (in this implementation of an application descriptor, interfaces are called 'methods').

```
<?xml version="1.0" encoding="utf-8" ?>

<gw-application id="101" name="DemoGWApp"
version="1.0">
     <description>Application for testing
purposes</description>

        <published-methods
           <method name="getUserInformation1">
                <data-categories><data-category>User
Information 1</data-category></data-categories>
                <receiver-purpose-groups>
                     <receiver-purpose-group>
                          <receiver-role>User
Information 1 Role</receiver-role>
                          <processing-purpose>User
Information 1 Purpose 1</processing-purpose>
                     </receiver-purpose-group>
                <receiver-purpose-group>
          </method>
...
```

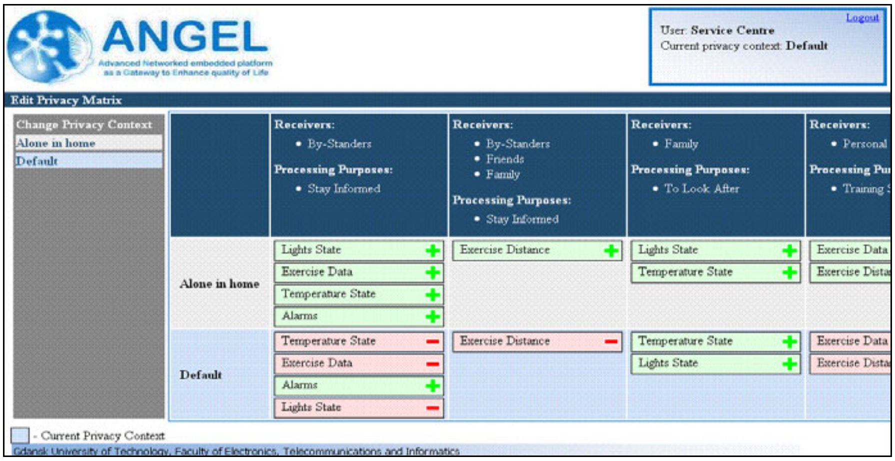
Fig. 4. Example application descriptor

System configuration information (*own* (5) and *conf* (6) functions) is provided by other Angel subsystem: *Angel configuration service*.

Angel user service provides a list of all receivers registered in Angel. The information is provided to *Privacy relations module* to enable mapping of receivers to receiver categories.

4.5 Privacy Subsystem Deployment

Privacy rules and *Privacy relations* modules are deployed on a distinguished Angel service terminal called Service Centre (SC). User interface to these modules is provided by a web portal, as shown on Fig. 5.


ANGEL
Advanced Networked embedded platform as a Gateway to Enhance quality of Life
Logout

User: **Service Centre**
Current privacy context: **Default**

Edit Privacy Matrix

Change Privacy Context
Alone in home
Default

	Receivers: • By-Standers **Processing Purposes:** • Stay Informed	**Receivers:** • By-Standers • Friends • Family **Processing Purposes:** • Stay Informed	**Receivers:** • Family **Processing Purposes:** • To Look After	**Receivers:** • Personal **Processing Pu** • Training S
Alone in home	Lights State + Exercise Data + Temperature State + Alarms +	Exercise Distance +	Lights State + Temperature State +	Exercise Data Exercise Dista
Default	Temperature State − Exercise Data − Alarms + Lights State −	Exercise Distance −	Temperature State + Lights State +	Exercise Data Exercise Dista

☐ - Current Privacy Context

Gdansk University of Technology, Faculty of Electronics, Telecommunications and Informatics

Fig. 5. Privacy management interface (privacy matrix)

Context management module is deployed on both, SC and Angel gateways. On SC it is integrated with *Privacy rules* module. On gateways it is integrated into middleware library and a dedicated Angel application (deployed on each gateway) has been created to provide its user interface. On gateways, however, the *Context management* module does not provide for defining new contexts – only switching of the current context is supported. Current context switching is broadcasted to all gateways owned by a given user from SC. In case the switch was originated on a gateway, the change is first communicated to the SC and then propagated to other relevant gateways.

Privacy enforcement module is deployed on gateways as a part of the Angel platform. It performs two main tasks:

- collects privacy configuration data,
- provides Angel applications with privacy decisions.

Privacy configuration data is distributed to gateways from SC (*canAccess'* (7) by *Privacy rules* module and *mapping* (3) by *Privacy relations* module) either on demand or when the data changes.

Provision of privacy decisions is done through a set of library functions:

- *isLocalAllowed* – a method called by a gateway application before serving any request (received through a Web Services call). It checks whether serving the indicated request is allowed from the privacy viewpoint.
- *isRemoteAllowed* – a method called before a gateway application calls any remote Web Service. It checks whether the call is acceptable from privacy perspective.

4.6 Case Study

The presented solution has been tested in the Angel privacy demonstrator. The objective was to demonstrate privacy management in the context of a realistic scenario of Angel system application. The privacy demonstrator scenario was as follows.

"*Tom has subscribed to Angel. He already installed sensors to monitor his home environment (light and temperature sensors were used in this demonstrator). He acquired some personal devices (a step-counter was assumed in the demonstrator) to support him while exercising. Tom is happy with the system, but he is also very concerned with his privacy and want to have control over his private data disclosure. This in particular involves his body related parameters and the home environment parameters*"

The demonstrator consisted of Service Centre, a fixed gateway (NXP STB810 [7]), wireless temperature sensors, wireless light sensors and wirelessly controlled colour lamp. The sensors were installed in Tom's bedroom and in the living room. The measurements were collected by an application deployed on the gateway and, if the temperature exceeded the predefined level, the application signalled an alarm by a red flashing light. The step-counter measurements and an outside temperature were also fed to the system. The application software used Angel middleware library to control data disclosure. Different application software deployed on the gateway displayed received measurements on the attached TV monitor. A similar application for data display was included in the service terminals deployed on a PDA device and on a desktop computer.

The following receiver categories were distinguished in Tom's privacy policy: Friends, Family, Trainer, Doctor and By-Stander (anybody near the TV set who can see the screen). The demonstrator considered various privacy related scenarios involving Tom and a related group of people, e.g. his wife, his friends, his trainer and his physician and visitors of his house. The demonstrator was tested and positively assessed during Angel consortium and evaluation meetings involving representatives of users (hospitals), technology providers (telecom industry, sensor producers), academic researchers, project officer and independent reviewers.

5 Related Work

There have been numerous proposals related to the privacy management in distributed environments and some of them address the issue of context dependence.

In PeCAN [8], context is a set of user beliefs (represented as atomic assertion and rules) applicable at a current state. The context is determined in relation to the visited web site, the role assumed by a user (father, researcher, etc.) and a transaction the user is involved in (browsing, buying, etc.). In DPM [9] context is determined by sensor measurements. Privacy preferences and policies are stored in the P3P format. TLC-PP [10, 11] assumes a predefined set of privacy contexts (called situations) and additionally a predefined set of ‘rewards’ (possible gratification for disclosing private data). In PRIME [12] access to private data is influenced by context, defined as information about entities of interest, as well as any ambient parameters concerning the environment, where a transaction takes place. Similar approach is chosen by Prime Life [13]. In MUPPS [14] context is understood as a characterisation of user’s privacy situation (which includes transaction type, time, user location etc.) when accessing a particular website. In PERSIST [15, 16] user context is taken into account and is understood as a collection of attributes describing the user (including location, time, but also services being used by user).

Most of the above solutions are focusing on Internet users in various e-commerce scenarios or similar applications. Our solution is more focusing on the situations where the users are embedded in intelligent ambients equipped with wireless devices with sometimes highly limited resources. Therefore our priority was to find an effective mechanism applicable in the environment subjected to severe efficiency constraints.

The DPM solution [9] also focused on enforcing privacy on disclosure of WSN-gathered data, but it operates in a system where privacy related decisions are centralized. In our case, however, the privacy related decisions are distributed.

PERSIST [15, 16] covers a broad scope, including privacy, trust and identity management. Privacy related decisions are based on basic variables’ values (the values of all these variables determine the current context) and Semantic Web techniques are used to support these decisions. This approach is general but may result in a considerable load related to application of reasoning engines. In our approach, the solution is more specific but the gain is in potential efficiency increase.

6 Conclusion and Future Work

Our approach was developed within the broader context of the approach addressing trustworthiness of the Angel system and platform [17]. In the presented solution we abstract from the way the user contexts were determined. In this sense our solution can be interfaced with any mechanism of context determination. In the demonstrator described in the paper, the decision about contexts selection was left to a user. We also assume that users’ privacy policies refer to contexts explicitly while defining restrictions on private data.

Users’ privacy policies refer to (user defined) contexts and set restrictions on private data disclosure to different receiver categories. The policies remain unchanged as long as the set of receiver categories is not changed. And again, we abstract from the way of determining such policies which means that the proposed solution can be interfaced to any method which results in a similar way of privacy policy representation. In the implementation presented in this paper, the policies were specified by the users by means of a dedicated graphical interface.

Privacy policies and the information of the current context of users forms the base for the mechanism controlling private data disclosure. The mechanism takes into account system configuration, and in particular its distribution into autonomous computing nodes. Each node is capable of disclosing data through its interfaces, and the mechanism installs a sort of a 'valve' on each interface. Closing and opening this valve is used to control private data release in accordance with the users' privacy policies. An important feature is that these decisions can be localized (to the computing nodes) with the possibility to keep at a given node only this information which is needed to make a decision. This provides for efficiency in terms of resources consumption which is highly relevant for mobile devices. A decision on private data release at a given gateway can be made based on locally stored data, without additional communication with remote nodes. And the amount of locally stored data needed to facilitate such decisions is limited. The locally stored information needs to be updated only if the system configuration or users' privacy policies change.

The proposed mechanism is highly transparent to the applications running on the nodes. It is hidden in the middleware (the platform) and the only what is required from an application is to call a proper middleware function before releasing data through an interface. As the platform is deployed on each node the mechanism is available on every node.

The distinguishing features of the presented solution can be summarised as follows:

- it is neutral to the context determination method; the only restriction is that the contexts are known and used as the parameter for the proposed mechanism,
- it assumes that users' privacy policies explicitly set restrictions on private data disclosure to different receiver categories in a context dependent way,
- it assumes that at any system state the current context of each user is determined,
- it provides a distributed mechanism which, for each node, implements a private data disclosure controlling function which enforces consistency with the privacy policies of data owners,
- the mechanism is resource efficient in the sense that on each node only the information necessary to support local decisions is maintained,
- the mechanism is hidden in the platform which makes it transparent to the applications running on the nodes.

In terms of scalability, our mechanism depends mainly on:

- number of data receivers *nr*,
- number of interfaces deployed on a gateway *ni*,
- number of contexts *nc*,

in the sense that the amount of information stored on a gateway is determined by the above factors (the required storage size is proportional to *nr*ni*nc*). The number of users and the number of gateways are not significant, as they influence the storage requirement of Service Centre (SC), which is not resource limited.

In reality however, *nr* and *ni* will not be a meaningful barrier for scalability, as these numbers are related to the number of applications deployed on a gateway. And it is reasonable to assume that the gateways with more applications deployed will be equipped with more resources at system configuration time.

In our implementation, context detection and switching was left to the users. In such case the number of different contexts managed by a user can not be too large and (which is even more important) the contexts have to be well understood by the users. In case of using an automated context management scheme that offers more support for the user, the number of contexts could grow. This could present a problem for scalability, if the number of contexts is very large (even though the amount of required data grows linearly with *nc*).

The mechanism presented in the paper can adapt to configuration changes (adding/removing applications deployed on a gateway, adding/removing gateways, changes in the set of receivers, etc.). However, the changes first need to be adequately represented in the configuration data. This poses a limit on applicability of the mechanism in a fully dynamic environment.

The next steps considered for further development of the proposed approach include:

- Investigating suitability of the mechanism for different context management approaches. In particular it is planned to investigate the consequences of removing the assumption that contexts are mutually exclusive.
- Extending the mechanism to Smart Node level. In Angel project, providing the privacy control mechanism on Smart Node level was initially considered, but following the decision that Smart Nodes will not disclose data, it wasn't further investigated.
- More flexibility in privacy policy definition scheme. The scheme for defining privacy policies worked well in the demonstrator scenario. However, in case of large numbers of receiver categories and data types, it could be unmanageable. A solution to this problem might introduce taxonomies of data receiver categories and private data types. This would help users to express the policies in more concise way.

Acknowledgments. The work presented in the paper was partially supported by the 6th FR STREP *Advanced Networked embedded platform as a Gateway to Enhance quality of Life* (ANGEL), Contract number IST-033506.

References

1. ZigBee Alliance, `http://www.zigbee.org/`
2. European Commission: Directive 95/46/EC of the European Parliament and of the Council on the protection of individuals with regard to the processing of personal data and on the free movement of such data. Offic. J. of the Euro. Communities L 281, 31–50 (1995)
3. 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 (Directive on privacy and electronic communications). Offic. J. L 201, 37–47 (2002)
4. OECD Guidelines on the Protection of Privacy and Transborder Flows of Personal Data (1980)

5. Health Insurance Portability and Accountability Act, Privacy/Security/Enforcement Regulation Text, 45 CFR Parts 160 and 164, U.S. Congress Act, with amendments (2003)
6. Bruynen, J., et al.: Deliverable D1.2: Angel System Specification. Technical report, Angel Consortium (2008)
7. Advanced IP STB reference platform STB810, `http://www.nxp.com/applications/set_top_box/ip_stb/stb810/`
8. Jutla, D.N., Bodorik, P., Zhang, Y.: PeCAN: An architecture for users' privacy-aware electronic commerce contexts on the semantic web. Infor. Syst. 31, 295–320 (2006)
9. Hong, D., Yuan, M., Shen, V.Y.: Dynamic Privacy Management: a Plugin Service for the Middleware in Pervasive Computing. In: 7th international conference on Human computer interaction with mobile devices & services, pp. 1–8. ACM, New York (2005)
10. Skinner, G.: The TLC-PP framework for delivering a Privacy Augmented Collaborative Environment (PACE). In: International Conference on Collaborative Computing: Networking, Applications and Worksharing, pp. 289–293. IEEE Computer Society, Washington (2007)
11. Skinner, G., Han, S., Chang, E.: Integration of Situational and Reward Elements for Fair Privacy Principles and Preferences (F3P). In: IEEE International Conference on Industrial Technology, pp. 3078–3082. IEEE Computer Society Press, Los Alamitos (2006)
12. Ardagna, C.A., Cremonini, M., De Capitani di Vimercati, S., Samarami, P.: A Privacy-Aware Access Control System. J. of Comp. Sec. 16, 369–392 (2008)
13. Ardagna, C.A., De Capitani di Vimercati, S., Paraboschi, S., Pedrini, E., Samarati, P.: An XACML-Based Privacy-Centered Access Control System. In: The 1st ACM Workshop on Information Security Governance, pp. 49–58. ACM, New York (2009)
14. Saleh, R., Jutla, D., Bodorik, P.: Management of Users' Privacy Preferences in Context. In: IEEE International Conference on Information Reuse and Integration, pp. 91–97. IEEE Computer Society, Washington (2007)
15. Liampotis, N., Roussaki, I., Papadopoulou, E., Abu-Shaaban, Y., Williams, M.H., Taylor, N.K., McBurney, S.M., Dolinar, K.: A Privacy Framework for Personal Self-Improving Smart Spaces. In: International Conference on Computational Science and Engineering, vol. 3, pp. 444–449. IEEE Computer Society, Los Alamitos (2009)
16. Venezia, C., Taylor, N., Williams, H., Doolin, K., Roussaki, I.: Novel Pervasive Computing Services Experienced through Personal Smart Spaces. In: Tenth International Conference on Mobile Data Management: Systems, Services and Middleware, pp. 484–489 (2009)
17. Górski, J., Gołaszewski, G., Miler, J., Piechówka, M., Baldus, H.: Trustworthiness: safety, security and privacy issues. In: 14th IEEE International Conference on Electronics, Circuits and Systems, ICECS 2007, Morocco, pp. 641–644 (2007)