Information Sciences 180 (2010) 2979–2990

Contents lists available at ScienceDirect

Information Sciences

journal homepage: www.elsevier.com/locate/ins

Activity-oriented access control to ubiquitous hospital information and services

Xuan Hung Le [a], Sungyoung Lee [b,*], Young-Koo Lee [b], Heejo Lee [c], Murad Khalid [a], Ravi Sankar [a]

[a] Department of Electrical Engineering, University of South Florida, 4202 E. Fowler Avenue, Tampa, FL, USA
[b] Department of Computer Engineering, Kyung Hee University, Seocheon, Gihung, Yongin, Geyonggi 446-701, South Korea
[c] Department of Computer Science and Engineering, Korea University, Anam-dong Seongbuk-gu, Seoul 136-701, South Korea

ARTICLE INFO

Article history:
Received 10 August 2009
Received in revised form 24 February 2010
Accepted 19 April 2010

Keywords:
Access control
Ubiquitous hospital information system and services
Security
Human activity

ABSTRACT

In hospital information systems, protecting the confidentiality of health information, whilst at the same time allowing authorized physicians to access it conveniently, is a crucial requirement. The need to deliver health information at the point-of-care is a primary factor to increase healthcare quality and cost efficiency. However, current systems require considerable coordination effort of hospital professionals to locate relevant documents to support a specific activity. This paper presents a flexible and dynamic access control model, Activity-Oriented Access Control (AOAC), which is based on user activity to authorize access permissions. A user is allowed to perform an activity if he/she holds a number of satisfactory attributes (i.e. roles, assignments, etc.) under a specified condition (e.g. time, location). Results of AOAC implementation in a realistic healthcare scenario have shown to meet two important requirements: protecting confidentiality of health information by denying an unauthorized access, and allowing physicians to conveniently browse medical data at the point-of-care. Furthermore, the average execution time was 0.078 s which allows AOAC to work in real-time.

1. Introduction

Current hospital information systems require considerable coordinated effort of physicians to find and browse relevant data to support a specific healthcare activity. They have to constantly log in and out of devices, start and stop sets of applications, look for the types of information needed for their clinical care and browse information repeatedly. For example, when a physician discusses with other physicians about a patient's progress, he/she has to manually browse the patient's electronic medical record to see the progress; at a patient's bedside, after checking the patient's health status, the physician has to manually open the diagnosis and treatment record of this patient to update it; when the physician diagnoses a patient with a broken leg, he/she has to manually browse the electronic X-ray images provided by a radiologist. These examples show significant efforts by the physician to search and browse different medical data that he/she needs for his/her activity. In order to avoid wastage of time and resources and to increase efficiency of professional hospital work, a new access control model is required. First, the access control mechanism should be able to dynamically grant permission to a physician to access all data related to a healthcare activity. Second, the mechanism should be able to flexibly permit or revoke permission when the physician is or is not allowed to perform an activity. Third, the access control mechanism should be able to automatically provide the right medical data at the right time so as to reduce the physician's effort to search and browse the med-

* Corresponding author. Tel.: +82 31 201 2514; fax: +82 31 202 2520.
E-mail addresses: xhle@usf.edu (X.H. Le), syle@oslab.khu.ac.kr (S. Lee), yklee@khu.ac.kr (Y.-K. Lee), heejo@korea.ac.kr (H. Lee), mkhalid@mail.usf.edu (M. Khalid), sankar@usf.edu (R. Sankar).

0020-0255/$ - see front matter © 2010 Published by Elsevier Inc.
doi:10.1016/j.ins.2010.04.020

ical data. In order to fulfill these requirements, the access control mechanism needs to take into account the physician's activity. This dynamically grants an appropriate access permission to the physician to access relevant data according to his/her activity. It requires a strong coordination between the physician, the activity he/she is allowed to access, and the medical data relevant to the activity.

Most of current solutions are based on user *identification* (ID) or user's *role* [1–8]. They are application - dependent and do not address the intricate security requirements of hospital applications. Ubiquitous hospital systems require flexible, extensible, context-aware and dynamic authorization enforcement. The policy specification of these models tightly couples ID/role of users with their permissions. This coupling does not support user tasks. In [9,10], accessing time, user location and other contexts were applied as a foundation to authorize access permission. However, this context is general and it does not indicate user activities. Considerable effort must be spent to adapt these approaches in reality to provide information and services to users at the *point-of-care*. In this paper, we propose a new access control model based on user activity, *Activity-Oriented Access Control* (AOAC), to address aforementioned security requirements in hospital applications. A general AOAC model was presented at *RTCSA'07* [11]. In this study, a comprehensive extension of AOAC methodology was accomplished. Furthermore, AOAC was implemented in a realistic scenario for evaluation.

AOAC employs the role concept from RBAC, but extends further than RBAC to provide a flexible and dynamic access control mechanism. In RBAC, authorization is based on user's role such as *doctor* or *nurse*. It does not consider the specific activities a physician may have to perform. Consequently, the physician has to spend considerable effort to browse information he/she needs for each activity he/she is doing. AOAC can reduce effort by recognizing what he/she is doing and dynamically providing such information to him/her. To do this, AOAC controls access permissions to hospital information and services based on the user activity. A user is allowed to perform an activity if he/she holds a number of satisfactory attributes (i.e., roles, assignments, etc.) under a certain environment constraint. For example, Dr. John is assigned to diagnose a patient Carol. To carry out this activity, John needs access permissions to Carol's medical record, X-ray image, and blood test result. As the system knows that the doctor performs "*Pneumonia treatment for patient Carol*", all related medical information would be displayed to him. Such information is of course authorized to the doctor. If an activity (task) is revoked, all corresponding privileges become invalid without human intervention. In the above example, if the doctor's "*Pneumonia treatment for patient Carol*", activity is revoked, all access privileges related to this activity become invalid without human intervention. RBAC requires more effort to make it work, because a role may involve various activities. Although some access privileges are no longer needed for an activity, they are still available to that role. For example, with a role 'doctor', a person may be assigned several treatments. Consequently, he/she is conferred a right to access information related to his/her patients. Once a treatment for a patient is accomplished, access permissions to that patient's information are still available. Therefore, RBAC requires extra effort to disable those access privileges.

The remainder of this paper is organized as follows. Section 2 briefly discusses related work and compares it to AOAC. In Section 3, we discuss hospital activities based on our field study. Section 4 presents requirements of authorization services in hospitals and a typical scenario. The proposed model is described in Section 5. In Section 6, we describe the implementation and results. Finally, Section 7 concludes the paper and outlines our future work.

2. Related work

Access control technology has a long history. In the late 60's, Lampson introduced a formal, mathematical description of access control based on an *access matrix* [1]. In 1973, Lapadula and Bell [2] proposed the first *rule-based access control* model. In 1983, access control technology took a significant step forward when the U.S. DoD published its *Trusted Computer System Evaluation Criteria* (TCSEC) [3]. However, a limitation of these models is that they manage privileges based on individuals, rather than groups of individuals. This introduces high complexity and significant cost to manage a large-scale system.

During the early 90's, a new access control model based on the user's *role* (*Role-based Access Control* – RBAC) [4,5] was introduced to tackle this problem. RBAC is conceptually simple: access to computer system objects is based on an user's role in an organization. The authorizations are not assigned directly to particular users, but to roles. A role denotes a job function describing the authority and responsibility conferred on users assigned to that role. These approaches exploit role information to determine the set of access permissions. Therefore, they are not efficient to apply in ubiquitous hospital environments where access permissions are tightly coupled to the activities, rather than user identities or roles.

Many efforts have been made to extend traditional access control models to support a particular computing system such as [6,7]. Zhu et al. [6] introduced a practical Mandatory Access Control (MAC) model for XML database. A mandatory access control is used by an operating system to constrain the ability of a *subject* (i.e. a process or thread) to access or generally perform some sort of operation on an *object* (i.e. a file or directory). This is only suitable for access control in a low-level computing system. Luo et al. [7] presented a trust degree based access control in grid environments. A trust relationship across grid domains was introduced to decide an access privilege. This model is helpful for grid environments in which each entity is usually unknown to another entity and they normally do not have prior trust relationship.

Lu et al. [8] presented a Task-Activity Based Access Control (TABAC) for process collaboration environments. Although TABAC seems to be similar to our approach due to the term *activity*, it is different in several aspects. First, *activity* in TABAC referred to a process action such as *invoke, receive, reply, assign, wait, if*, or *while*. Meanwhile, our *activity* refers to a human activity, namely a healthcare activity such as medical treatment or diagnosis. Second, TABAC was proposed for workflow

management systems in which it focused on workflow processes while we address human activities for healthcare. Third, TABAC dynamically assigns permissions to each process when the process is being performed. This is not true in healthcare environments because permissions must be assigned beforehand, conforming to hospital policies or national law (e.g. HIPAA). Fourth, TABAC allowed users to get permission through either their ID, their roles, or their task, which makes it become more complicated to control in healthcare domains. This is also a barrier to provide the access dynamism as we have discussed earlier.

Several models have been introduced to control user access to electronic patient records [9,10]. In [9], Motta et al. presented a *contextual role-based access control model* to increase patient privacy and the confidentiality of patient's data, whilst being sufficiently flexible to consider specific cases. However, their concept of context is limited to some environmental information that is not suitable to support user activities. In [10], the authors assumed that information required by specialists is highly dependent on their location. The paper presented a location-aware medical information system that was developed to provide access to resources such as patient's records or the location of a medical specialist, based on the user's location. However, location information does not indicate what a user is doing. Different activities could be carried out at the same location.

There is a difference of context usage between [9] and our proposed approach, AOAC. In [9], an environment context for direct authorization was used. However, AOAC uses environment context as a part of information to deduce user's activity. In [9], it stated that a context denotes environmental information available at access time. That includes current user's information, time, location of access, and security (e.g. login name). Even though such context is very useful, it does not say what activity a user is doing. AOAC employs the similar concept of environmental context with further extension as a 'raw' context. Such 'raw' context is used to deduce a high-level context that is a specific activity.

3. Hospital activity overview

We conducted a field study in two hospitals located in Suwon and Seoul cities that have around 967 and 800 beds, respectively. The study is mostly based on daily observation and short interviews with nurses, physicians, and administration staff.

Hospital electronic information comprises three parts: Electronic Medical Records (EMR), Picture Archiving and Communication Systems (PACS), and Order Chart Systems (OCS). EMR is the main hospital database to store all in- and out-patients medical records. Korean law stipulates that EMR must be maintained for at least 25 years. PACS includes X-ray, and CT-images. OCS provides instant medical information to physicians and nurses during diagnosis or treatment. It includes patient medical records, and prescription orders. After diagnosis or treatment, information in OCS is moved to EMR. Four main divisions can be observed. These include emergency rooms, patient wards, treatment rooms, and examination rooms.

Hospitalization normally includes six steps: admission, examination (i.e. diagnosis), patient care (i.e. treatment), recovery, discharge, and billing. When a patient goes to a hospital, the first step is admission. It involves filling forms, recording patient's personal information, describing reasons for admission, etc. After that, the patient is assigned a unique hospital identification number. Next, the patient is taken to an appropriate department for diagnosis and treatment.

During diagnosis, a physician checks the patient's medical history, signs and symptoms, along with other necessary examination processes. For example, if pneumonia is suspected, the physician orders blood tests and a chest X-ray. Blood tests show an elevation of blood cells, whilst a chest X-ray is useful to show an infiltrate indicative of pneumonia. Diagnosis may not be done on the first visit. The patient may have to visit the hospital more than once to get a final diagnosis. After obtaining sufficient information, the physician identifies the patient's disease. If the patient has a pneumonia disease, then he/she is assigned to a pulmonologist for treatment. The physician and nurse update results to the OCS.

Treatment has to be individualized depending on the causative agent for the pneumonia. Nevertheless, a specialist may begin empirically based on the diagnosis. For the treatment, the specialist may request an emergency bronchoscopy with the help of a pulmonologist to obtain a good specimen directly from the infected lung tissues for histological identification, Gram staining and various culture methods. At regular intervals (e.g. every 8 h), the following procedures could be carried out by a nurse or a phlebotomist: blood is drawn and analyzed; blood pressure, heart rate, temperature, and O_2 saturation are recorded; intravenous (IV) fluid levels are recorded and replaced as needed; other evaluations such as body condition, pain sensations are recorded; other specimens are obtained and sent to the laboratories. Every morning, the specialist visits his/her patients in the ward at a specific time. Mostly, he/she carries out the treatment during this time. This is the *visiting round*. If a further treatment is required, he/she will make an appointment for the patient to visit the treatment room. At each step, results are updated to OCS.

During recovery, a patient may be encouraged to walk for exercise. The patient's movements, meals, medication consumed, etc. are recorded. Finally, when the specialist determines that hospital care is completed, the patient is issued post-hospital care instruction, and processed for discharge. Billing is also carried out and statements may be sent to various parties such as health insurer for payment.

The above description is an overview of general activities in the hospital that was studied. In the AOAC system, more granulated activities and related resources are considered. For example, although we mentioned a very general term 'diagnosis', there are different types of diagnosis. As we observed at the admission desk, a patient must describe to the staff his/her symptoms before being assigned to a suitable physician. For example, Carol, who is a pregnant patient, recently experienced a pain in her abdomen when she fetched a heavy chair. The staff assigned her to an obstetrician. Therefore, to be

granted to perform the activity "*pregnancy diagnosis for Carol*", the physician has to have the roles "*obstetrician*", "*hospital employee*", and "*an assignment of diagnosis for Carol*". The activity is associated with a number of specific data such as Carol's pregnancy progress and symptoms.

4. Requirements for hospital authorization services

Based on our field study, we summarize the following requirements for hospital authorization services:

1. Medical information is highly sensitive. Confidentiality of such information must be kept strictly, yet at the same time it must not hinder patient care by denying legitimate access requested by hospital employees or other authorized personnel.
2. The access control model should be able to reduce coordination efforts of hospital employees to locate relevant documents according to their specific activity. First, the access control mechanism should be able to dynamically grant permission to a physician to access all data related to a healthcare activity. Second, the mechanism should be able to flexibly permit or revoke permission when the physician is or is not allowed to perform an activity. Third, the access control mechanism should be able to automatically provide the right medical data at the right time, so as to reduce the physician's effort to search and browse the medical data

The following sample scenario illustrates these requirements:

Dr. John obtained a medical doctor license from National Medical Council (NMC) before he was employed by the hospital. One day, a new patient, Carol, is hospitalized. After admission, she is assigned to John for diagnosis and treatment.

At 8:00AM, John arrives at his office. The first thing he needs to do, as usual, is discuss patient progress with his colleagues. He logs on to his computer, and browses a number of resources, such as patient progress reports and treatment plan.

After the discussion, John starts a visiting round at the patient ward with a nurse Alice to see Carol. For the treatment, John needs to use a computer attached to the patient's bed for detailed information. He needs to access Carol's medical record, treatment history, detailed progress, latest symptoms, etc. When the treatment is finished, Alice updates all results to the OCS.

This sample scenario might be handled by RBAC as follows. John is assigned the role '*doctor*', '*hospital employee*'. With these roles, he is allowed to access Carol's medical data. In the morning, John logs into the hospital information system via his computer. Then, he has to manually browse Carol's progress and treatment plan. The system verifies each access permission to specific data he is attempting to browse. When he visits his patients at the ward, he needs to log in to the system again. He enters the patient's ID (or the name) to find his/her medical information and browses it. After the treatment, nurse Alice logs into the system from her portable device, searches Carol's progress data and updates the latest treatment progress.

With RBAC, it shows considerable effort by the doctor and the nurse to search and browse the required information. This is because of two reasons. First reason is that RBAC grants access permission to a user according to his/her role (i.e. the user can access different types of data with a certain role). There is no relationship between a role and an activity. Therefore, it is impossible for RBAC to provide appropriate data to a certain activity. It obviously reduces the efficiency of medical care. In many situations, such as emergency, time is a critical factor to save a human life. Supporting user activity to access hospital information and services is essential to increasing the efficiency of healthcare in hospitals. Second reason is that an access permission is granted based on a role that will not change unless the role is revoked. In dynamic activity environments like hospitals, it requires an access permission to be changed once the user accomplishes a task or an activity (e.g. *treatment for a certain patient*). When he/she finished the task, he/she should not be able to access the patient's medical record unless he/she is assigned a new activity. AOAC solves this issue by controlling user's access privileges based on an activity a user is allowed to perform. In addition, AOAC integrates a trigger module to automatically request data to users according to what he/she is doing. It can recognize what a user is performing and invokes corresponding data.

5. Activity-oriented access control

This section presents the proposed *Activity-Oriented Access Control* (AOAC) model. First, the overview of AOAC is described, followed by the AOAC model, privilege delegation mechanism, activity activation rules, and permission activation rules.

The nature of organizational authorizations is determining who is allowed to do what. For example, Dr. John is permitted to carry out treatment of pneumonia for patient Carol. Therefore, by associating a user with activities, we can easily match the AOAC into real environments. In order to accomplish an action, a user needs access permissions to a number of resources. By connecting each activity with access permissions, the proposed model highly supports user activity. Therefore, we abstract the access control model in three levels: user level, activity level, and privilege level, as illustrated in Fig. 1. Each user holds a number of credentials [12,13] specifying his/her attributes, such as hospital role, experience, assignment. A credential can be a certificate/qualification (e.g. medical license), a hospital role (e.g. screening nurse), or an assignment from another user (e.g. a doctor is on leave, so he/she assigns another doctor to diagnose his/her patients during his/her absence). A

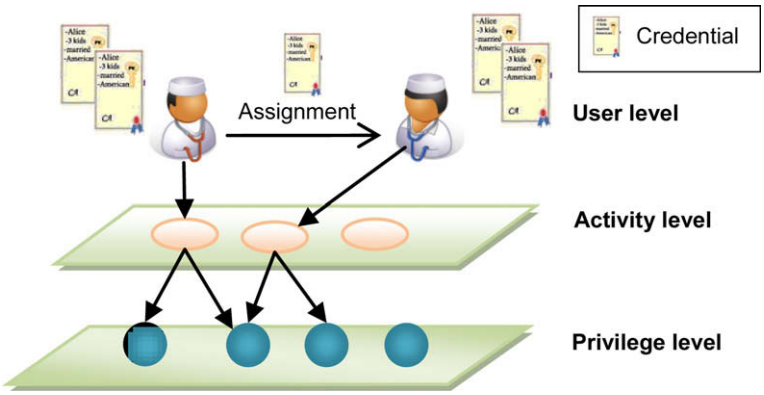

Fig. 1. AOAC abstraction levels.

user is authorized to perform a certain activity if the conditions are satisfied. We define the condition as *activity activation rule*. Each activity is associated with a number of access privileges. Those access privileges are needed to support the user to accomplish this activity. For example, the activity '*prescribe medicine for patient Carol*' requires access permission to medical record, X-ray images, blood test results, and medicine charts. We define this rule as *permission activation rule*.

Fig. 2 shows AOAC model. It is comprised of five administrative elements: (1) *users (i.e. user's attributes)*, (2) *activities* and (3) *permissions* (i.e. privileges), where *permissions* are composed of (4) *objects* and (5) *operations*. *User* is a human interacting with a computing system. *Activity* is a human activity. It differs from the term '*task*' in a workflow system in the sense that is does not model nor control real-world human activities. An activity can be created and modified according to the desire of the user. *Object* is medical data as well as system resource. An operation is an executable image of a program, for example '*read*', '*write*', '*execute*'. *Permission* is an authorization to perform certain operations within the system. *Constraint*, similar to the concept of constraint from RBAC model [4], is defined as a predicate that is applied to a relation between two elements returning a value of '*acceptable*' or '*not-acceptable*'.

In the AOAC model, *Activity Activation Rule* (AAR) is to allow a user to perform a certain activity if he/she holds a number of attributes including roles, user ID, and other credentials (e.g. an assignment). *Permission Activation Rule* (PAR) is to provide access permissions to an activity. Technically, AOAC differs from RBAC in that a *role* in AOAC is considered as an attribute of a user and it alone cannot decide what permission is allowed. It is not a bridge to connect between a user and permission. Permission is only directly connected to an activity which a user is allowed to perform. Therefore, if a user holds a number of roles, it will not cause any conflict of authorization like in RBAC model.

5.1. Privilege delegation via digital credentials

Privilege delegation is a term to indicate that *A* delegates to *B* a particular privilege. In AOAC, we define this as *assignment*, where *A* is an *assigner* and *B* is an *assignee*. Formally:

$$A \rightarrow B : assignment\ (CRED\ (USERS_1, USERS_2), N),$$

where user *A* assigns user *B* an assignment (i.e. task/activity) wrapped in a credential $CRED(\cdot)$, where $USERS_1$ is a subset of users who can grant the credential, $USERS_2$ is a subset of users who can be so granted, *N* is number of further assignment

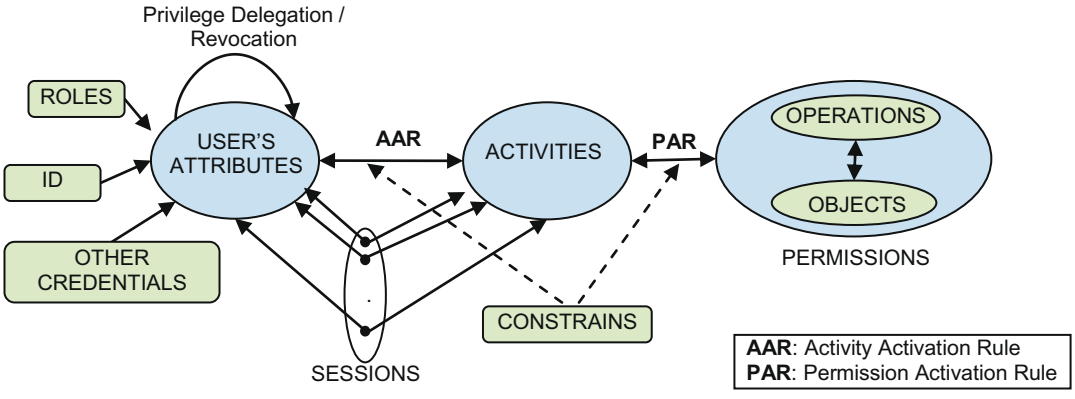

Fig. 2. AOAC model.

that an assigner can delegate to an assignee ($N = 1,2,3,\ldots$); if $N = 1$, then B cannot grant this credential to anyone; if $N = \infty$: anyone possessing the credential *CRED* can grant it to another user (B must be indicated in *CRED's* $USERS_1$).

For example,

$$John \rightarrow Peter : assignment\ (DIAGNOSE\ (\{doctors, nurses\}, \{doctors, nurses\}), 1).$$

Assignment occurs when a user grants a digital credential that directly or indirectly allows a user to perform one or more activities. The credential content may be an assignment of activities (*direct delegation*), or may be an assignment of role, qualification, etc., so that the user may activate an activity (*indirect delegation*). Our delegation approach differs from *Appointment* [5] in several aspects. Firstly, user A may not only delegate the object right to user B, but user B may also delegate the right to another user C, and so on. We call this *multi-step delegation*. Secondly, our privilege delegation may be *restricted* or *unrestricted*. It means that user A can restrict how user B can further delegate the access right to user C. Delegation in *appointment* approach is only concerned with how to grant another user a credential to activate one or more roles without any concern about further delegation or restriction.

Digital credentials are the digital equivalent of paper-based credentials. Digital credentials prove something about their owner (e.g. person position) or deliver some information of their owner (e.g. job assignment). We abstract the credential in a form of *CRED* ($USERS_1, USERS_2$), where $USERS_1$ is a subset of users who can grant the credential, and $USERS_2$ is a subset of users who can be so granted.

It is essential to restrict which users can grant (or be granted) a credential. This contributes two advantages. First, this complies with the hospital policies because, for instance, some sensitive credentials must be only granted by superiors to senior personnel in the hospital. For example, only oncologists are allowed to treat cancer patients, where as others who are not specialized in cancer treatment would not be permitted to work on this assignment. Second, it avoids a mistake or abuse of legitimate users and prevents unauthorized assignments. For example, if a user is not a *screening nurse*, he/she cannot assign patients to doctors for treatment; or if a user is not a doctor, he/she cannot designate a nurse to administer medicine.

The *Privilege Delegation* can be implemented based on X.509 *Privilege Management Infrastructure* (X.509 PMI) [12,13]. X.509 PMI differs from X.509 Public Key Infrastructure (X.509 PKI) in that PMI holds *Attribute Certificates* (i.e. *Attribute Credentials*, which can be used for authorization), whereas X.509 PKI holds a *Public Key* (which is only for authentication). More importantly, a *Source of Authority* (SOA) can assign an attribute certificate, whereas only *Certification of Authority* (CA) can assign a Public Key (being a CA is a very specialized function and there are usually very few of them in an organization, whereas any attribute holder can be a SOA). We can define a number of user attributes in each attribute credential. There are a number of ways we can implement a secure credential delegation mechanism such as PERMIS (*PrivilEge and Role Management Infrastructure Standards*) [13,14]. PERMIS provides a cryptographically secure PMI using public key encryption technologies and X.509 Attribute Certificates to maintain users' attributes.

5.2. Privilege revocation

Privilege revocation is crucial. In many situations, credentials should be revoked. For example, John delegated a credential to Peter for a particular task; if the task is accomplished, or Peter is transferred to another department, the credential should be revoked immediately. There are different reasons and different ways to revoke delegated credentials. Privilege revocation can be done in four ways [5]: by its assigner only; by anyone active in the credential; by the assignee's resignation; or by the rule-based revocation: revocations can be carried out by the system itself. There are different ways to revoke a privilege: time duration on the credential is expired; constraint on the credential is violated; the task is completed; revoked at the end of the assigner session or the assignee's session.

We propose two revocation types: *Single-step Revocation* (SR) and *Multi-step Revocation* (MR). SR is applied for *single-step delegations*. This means that Alice only delegates a credential to John without any further delegation from John. MR is applied for *multi-step delegations*. SR can be considered as a case of the *multi-step revocation* with the number of delegation steps set to one. There are different ways to cascade revocations. One of the simplest ways is based on credential identifiers (IDs) to revoke them. Thus, the original assigner (Alice) attaches a unique ID to each credential. Whenever she wants to revoke, she just needs to indicate the credential's IDs and the system will consider those credentials as invalid ones.

5.3. Activity activation rules

The user must satisfy the conditions of the *activity activation rule* to be authorized an activity. We use prolog-like expression to formulate our *activity activation rule* as follows:

$$ACT \vdash CON_1, CON_2, \ldots, CON_n,$$

where, *ACT* is an activity, CON_i is a condition including a user's attribute such as a privilege to perform an activity or a privilege to access a resource. It is not restricted to only user's properties or characteristics.

For example:

treating_patient (John,Carol) med_doctor (John), screening_nurse (Alice), patient (Carol), treating_assignment (Alice,John) i.e. John is allowed to treat Carol if he holds a medical doctor license, and is appointed by screening nurse Alice to treat Carol.

5.4. Permission activation rules

Whenever a user initiates an activity, corresponding permissions are automatically activated if a number of context constraints are satisfied. We define a *context constraint* as any requirement about contextual information such as time and location. Context constraints play a key role in specifying context-sensitive policies. It is very important to revoke privileges if contextual requirements are not met. *A permission activation rule* is formally defined as follows:

$$PERM \vdash ACT, CC_1, CC_2, \ldots, CC_m,$$

i.e., if the user can activate an activity *ACT* under satisfied context constraints $CC_1, CC_2, \ldots, CC_m$, then he/she is granted the corresponding permissions *PERM*.

For example, if Alice is authorized the activity "*taking_note*", then she is permitted to access EPR of Carol during work time.

$$read\_EPR\ (Alice, Carol) \vdash taking\_note\ (Alice), time\ (8:00, 17:00).$$

6. Implementation

6.1. System design

We have implemented the AOAC model for a realistic sample scenario mentioned in Section 4, to show how the system works and fulfills the aforementioned requirements. The AOAC system design is shown in Fig. 3. *Activity Recognition Manager* (ARM) detects user's activities. User's attributes, activities and activity activation rules are stored on a *Lightweight Directory Access Protocol* (LDAP) server. Permission activation rules are defined by *eXtensible Access Control Markup Language* (XACML) [15]. To be compliant with XACML standard, AOAC integrates a *Policy Enforcement Point* (PEP), and a *Policy Decision Point* (PDP). PEP makes decision of requests and enforcing authorization decisions. PEP incorporates a *Trigger* module to ease access. Once AOAC identifies authorized activity and access privileges, the trigger can deliver that information to users at the point-of-care. PDP evaluates applicable policy and renders and authorization decision. PDP is composed of three sub-components: *User-Attribute Manager* (UAM), which retrieves user's attributes according to user identifier (uid) from user's attributes LDAP server; *Attribute-Activity Manager* (AAM), which matches user's attributes to a set of allowed activities; and *Activity-Permission Manager* (APM), which retrieves all access privileges for given activities from XACML policies. The *Admin Tool* (AT) is used to define activities and policies. Possible activities in the hospital are predicted and defined in advance. In fact, the bootstrapping phase is not much more onerous than other approaches such as RBAC. In RBAC, we need to define a number of roles, and a number of access permissions related to each role. In a similar way, AOAC requires to define a number

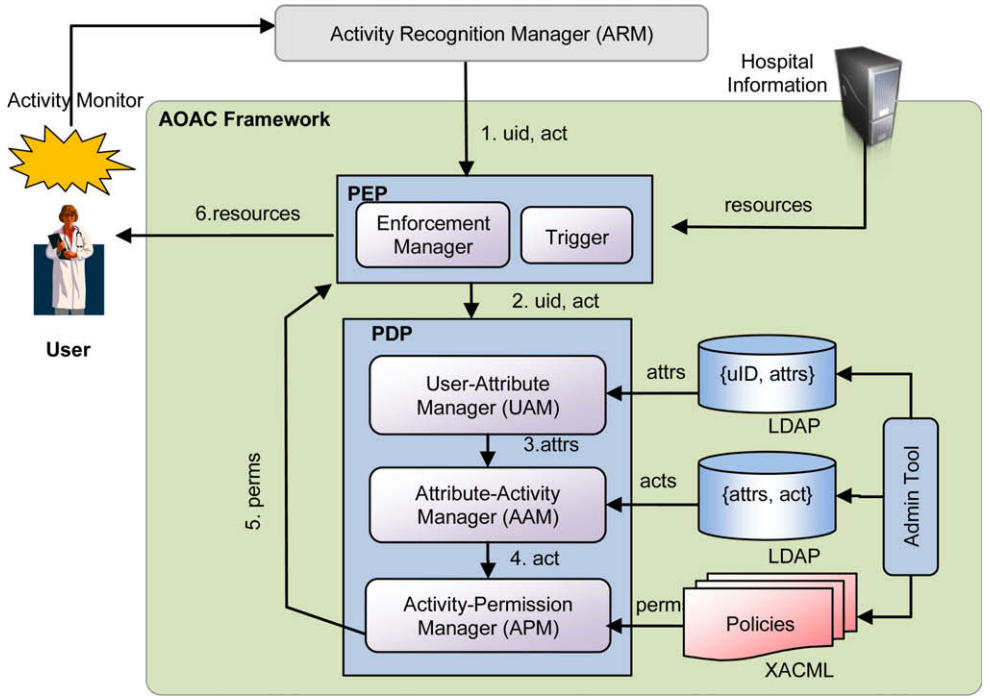

Fig. 3. AOAC system design.

of activities and related access privileges. Usually, all possible activities are defined in advance so that this rarely requires a change. This is similar to RBAC in which all roles must be predefined and the system does not require many changes later.

All of these AOAC's components were implemented in Java JDK 6.03 using *Eclipse 3.2.2*. The LDAP server was installed by *Apache Directory Server* 1.0.2 (http://apache.tt.co.kr/directory/apacheds/stable/1.0/1.0.2). The AOAC policy structure written in XACML format is shown in Fig. 4. In this policy specification, an activity is defined in the subject field. The policy defines what activity (*activity_id*) can access to which data (*recource_id*) with what permission (*permission_id*). To connect AOAC components with XACML policies, query policies and assess user's permissions, we used Sun Inc.'s XACML Library (http://dev.mysql.com/downloads/connector/j/5.1.htm).

The AOAC flowchart is shown in Fig. 5. The system operates using the following steps:

1. The user logs on to the system through a *Single Sign-On* (SSO) authentication. Once authenticated, AOAC queries from < uID, attrs > LDAP server to achieve user's attributes based on user's ID.
2. When AOAC detects that the user is performing an activity, the activity will be matched with the user's attributes. Based on *Activity Activation Rule* (AAR) in LDAP server, AOAC checks if the user is allowed to perform the activity or not. If his/her attributes are not satisfied AAR, AOAC will not take any further step.
3. If the user is allowed to perform the activity, then corresponding access permissions are queried from the policies. Medical data is then sent to the user.

```
<?xml version="1.0" encoding="UTF-8"?>
<Policy PolicyId="AOACpolicy100"
RuleCombiningAlgId="urn:oasis:names:tc:xacml:1.0:rule-combining-algorithm:ordered-permit-overrides">
  <Description>This policy was automatically created by AOAC engine </Description>
  <Target>
    <Subjects>
      <Subject>
        <SubjectMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
          <AttributeValue DataType = "http://www.w3.org/2001/XMLSchema#string">
                                      activity_id</AttributeValue>
          <SubjectAttributeDesignator AttributeId="urn:oasis:names:tc:xacml:1.0:subject:subject-
                    id" DataType="http://www.w3.org/2001/XMLSchema#string"/>
        </SubjectMatch>
      </Subject>
    </Subjects>
    <Resources>
      <Resource>
        <ResourceMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:anyURI-equal">
          <AttributeValue DataType = "http://www.w3.org/2001/XMLSchema#anyURI">
                                      resource_id </AttributeValue>
          <ResourceAttributeDesignator AttributeId =
                          "urn:oasis:names:tc:xacml:1.0:resource:resource-id"
                           DataType="http://www.w3.org/2001/XMLSchema#anyURI"/>
        </ResourceMatch>
      </Resource>
    </Resources>
    <Actions>
      <AnyAction/>
    </Actions>
  </Target>
  <Rule RuleId="CommitRule" Effect="Permit">
    <Target>
      <Subjects>
        <AnySubject/>
      </Subjects>
      <Resources>
        <AnyResource/>
      </Resources>
      <Actions>
        <Action>
          <ActionMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
            <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
                                      permission_id</AttributeValue>
            <ActionAttributeDesignator AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-
                              id" DataType="http://www.w3.org/2001/XMLSchema#string"/>
          </ActionMatch>
        </Action>
      </Actions>
    </Target>
  </Rule>
  <Rule RuleId="FinalRule" Effect="Deny"/>
</Policy>
```

Fig. 4. AOAC policy structure written in XACML.

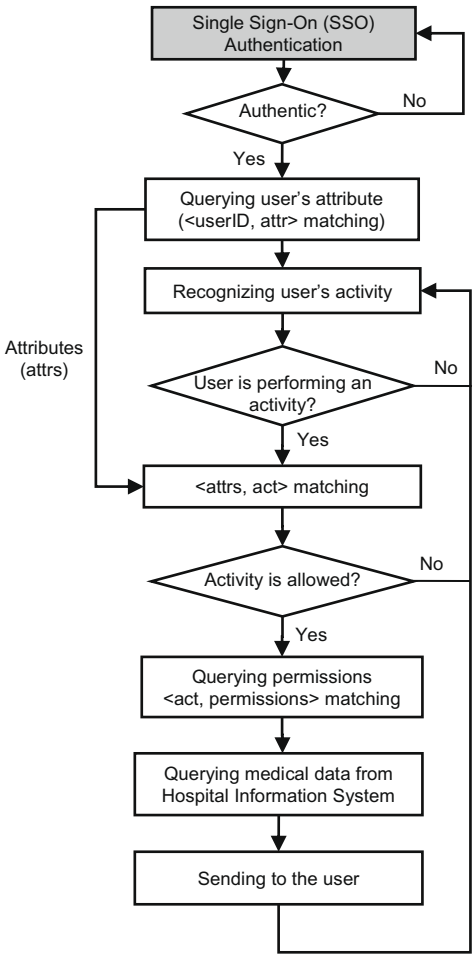

Fig. 5. AOAC flowchart.

There are two ways of activity recognition, as discussed in [16]. First, context awareness can provide all or part of the user's intention. Second, users may explicitly state intentions. In the former approach, we plan to incorporate the *Activity Recognition* (AR) engine from our *Secured Sensor Network-integrated Cloud Computing for u-Lifecare* (SC^3) system [17]. Currently, our activity recognition engines can detect a basic activity using a wearable sensor on the human wrist [18] or using embedded-sensors [19]. Although the results have shown about 90 percent accuracy of detection, we believe that it can reach 100 percent if we involve cyber context information such as where, when and what computing device an user is using to access hospital information. In the latter approach, users explicitly state intentions. The system may request activity description from a user by displaying a dialog-box "*Enter your current activity*". Although the strategy is straightforward, it is not trivial. An extreme scenario is that a physician who is called for immediate help to a patient suffering from heart-attack, rushes to the nearest computer to get vital information only to be met by the dialog-box requesting activity description. To tackle this issue, we implemented an auto-searching dialog-box, in which the system displays a list of similar activity names as the user is typing. An example is illustrated in Fig. 6. When the user is typing "*Treatment*", the box shows a list of similar activities, including "*Taking note of treatment progress for patient Carol*", "*Treatment of patient Carol*", etc.

6.2. Sample scenario

We illustrate the use of AOAC system and how it supports user activity in the ubiquitous hospital environment with the sample scenario mentioned in Section 2.

Dr. John obtained a medical doctor license from the National Medical Council before he was employed by the hospital. A Human Resource (HR) staff person, who is in charge of John's employment, assigns him three attributes. UAM adds to the LDAP server ($\{uID, attrs\}$) his hospital ID, department ID, and a credential specifying his medical doctor license.

One day, a new patient, Carol, is hospitalized. After admission, she is assigned to John for diagnosis and treatment. UAM inserts new attributes to the LDAP server ({uID, attrs}). This is the *treatment assignment* from Alice.

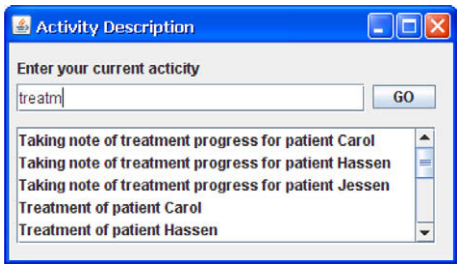

Fig. 6. A list of auto-searched activities is shown up as the user is typing.

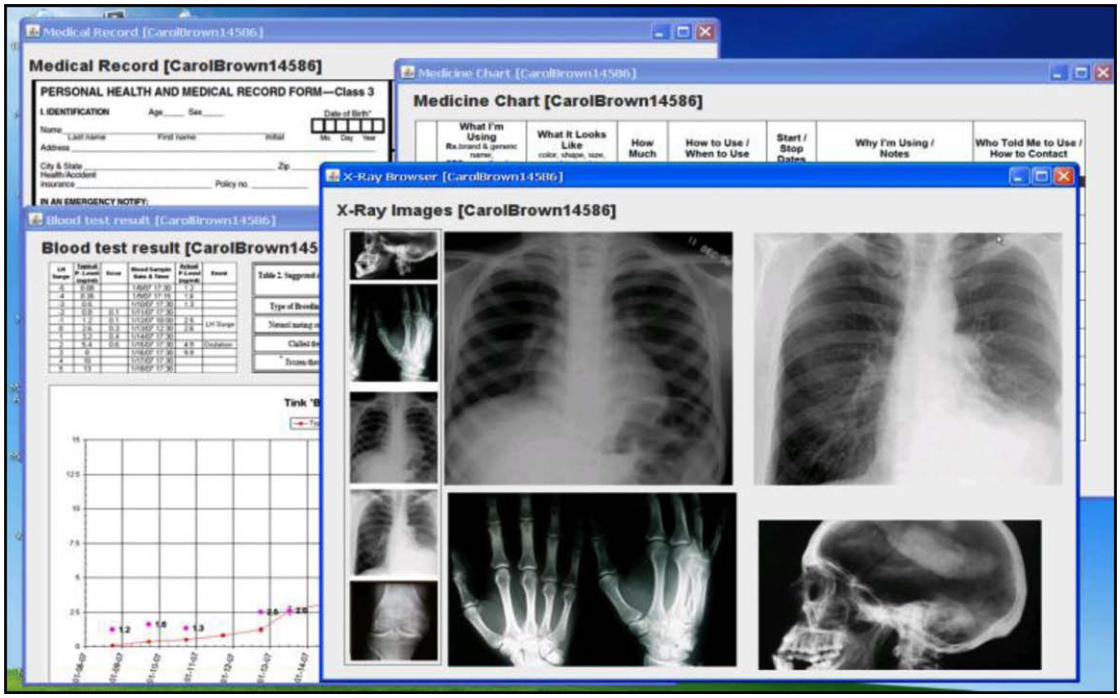

Fig. 7. Data supported for treatment.

At 8:00AM, John arrives at his office. The first thing he needs to do, as usual, is to discuss the patient's progress with his colleagues. After logging in, the monitor displays the dialog-box "*Enter your current activity*". As he types "*Discussing patient progress*", the box displays all similar activities. John selects "Discussing patient progress". The agent sends the activity to PEP. PEP forwards the activity name along with John's hospital ID to PDP. At PDP, after a number of processes as described in Section 6.1, a list of permissions is sent to PEP. Since John is authorized to carry out that activity, he has full permission to access patient's progress reports and treatment plans. The trigger module looks up the list, and then queries proper data from hospital information database. The data is then displayed on the computer as shown in Fig. 7.

After that, John starts a visiting round to his patient ward with a nurse, say Alice, to see his patient Carol. He enters the activity "*Pneumonia treatment for patient Carol*" into the dialog-box on the monitor attached to the patient bed. Since John holds an assignment credential, he is authorized to perform that activity. After authorization, the monitor attached on the bed shows related medical information, including medical record, treatment history, detailed progress, and latest symptoms. When the treatment is completed, Alice enters "*Taking notes on Carol*" to record all the treatment results and progress. In the above experiment, we considered information priority so that the display window containing the most critical information will be displayed at the top level of the screen.

6.3. Discussion

In general, there are different ways to protect the confidentiality of data. One way is to protect at the communication level by a strong encryption algorithm such as *Advanced Encryption Standard* (AES). Another way is to protect at the application

level, in which access of unauthorized user are prohibited by an access control mechanism. In this paper, the confidentiality of medical record is protected at the application layer by the proposed access control mechanism. We extended the sample scenario with different activities, such as medical treatment, diagnosis, admission/discharge, taking notes, medical prescription. When a physician was performing an activity, he/she entered it into the pop-up dialog-box, and then corresponding data was shown. In several cases, we also asked the physicians to intentionally enter an activity that they are not permitted to perform. Because the activity is not allowed, the corresponding access permissions are prohibited. Consequently, none of information was provided to them. That indicates AOAC meets the first requirement mentioned in Section 4. The hospital information maintains confidentiality without hindering patient's care by denying legitimate access request to hospital employees. If the activity was permitted, the system correctly and instantly provided related information and services to physicians and nurses. This shows AOAC meets the second requirement mentioned in Section 4. It delivers appropriate information to physicians at the *point-of-care*. The experiment was repeated 70 times with different scenarios. It ran on a PC server (Pentium IV 3.2 GHz and 1 GB of RAM) with the average execution time of approximately 0.078 s. That included receiving an activity description, processing authorization, and displaying appropriate data to users. This means that AOAC is able to work in real-time. In the first phase of implementation, we have not optimized the code. We believe that if the optimization is taken into account, the performance will be significantly improved. Even when the user performs many activities simultaneously, the execution time will not be significantly increased.

We have considered the situation where many simultaneous activities are executed. However, it is not an issue for the access control mechanism because we have observed that the execution time was not increasing when we increased the number of simultaneous activities. On the other hand, if our experiment was deployed on a faster PC server, the execution time would be decreased significantly. However, when many simultaneous activities are executed, the execution time of Activity Recognition (in case we apply Activity Recognition engine instead of the dialog-box) will be an issue. To address this issue, we are considering a solution of using Cloud Computing, as mentioned in our on-going work (SC^3) [17]. By taking advantage of Cloud Computing, we can increase the performance and decrease the financial cost of the system. Our next version will present the performance of a complete system including the Activity Recognition engine.

In our experiment, we have not considered the case that a user performs more than one activity at the same time because it affects the activity recognition mechanism. In particular, it adds high complexity to the activity recognition mechanism because the activity recognition depends on various sensing devices to gather activity context. When a user performs more than one activity, the sensing devices and recognition engine should be able to differentiate these activities. Our future work will address this issue.

7. Conclusion and future work

This paper presents the Activity-Oriented Access Control (AOAC) model to increase efficiency of the healthcare professionals. Through the implementation, we showed that AOAC is more flexible than RBAC by meeting two important requirements: protecting confidentiality of health information by denying an unauthorized access, and allowing physicians to conveniently brow medical data at the *point-of-care*. Furthermore, the average execution time was 0.078 s which allows AOAC to work in real-time.

We implemented a simple dialog-box for users to explicitly state their activity. However, this is not the 'best' choice. Instead, we plan to integrate our activity recognition engine to completely fulfill AOAC's goal. Currently, we are working on two activity recognition approaches: embodied-sensor based activity recognition [18] and embedded-sensor based activity recognition [19]. With an ontology engine and learning mechanism, the system will be able to work in real-time. In the former approach, a single sensor is attached to each person. It supports gyroscope and three-axis accelerometer measurement. An activity is predicted or inferred based on the gyroscope and accelerometer information. Activity history is also used to increase the accuracy of current activity detection. In the latter approach, a group of sensors is attached to objects and the surrounding environment, for example on a wall, a monitor, an electronic notepad, a medicine tray, a patient bed. Normally, when a person performs an activity, he/she must use a number of objects/tools. Whenever an object/tool is used, the attached sensor will report a signal to the central server. Using a sequence of these signals, we can predict or infer what the user is doing. Therefore, we will eventually be able to integrate our activity recognition engine.

In the above approaches, we implemented a simple activity prediction in a case that the physician begins an activity and then needs information, but the system will not provide those rights, until it realizes that the user is performing the activity. Although we observed that these situations seldom occur in hospitals, we will investigate more detailed prediction algorithm to make AOAC usable in different circumstances. Another piece of future work is to implement the credential manager based on X.509 PMI [12,13] to maintain user's attributes and privilege delegation. That includes secure issuance, distribution, and revocation of credentials. Besides, we will extend the model so that a user can perform more than one activity at a time. In fact, it will bring high complexity to the activity recognition mechanism because it depends on various sensing devices to gather activity context.

Acknowledgement

This work was supported by a grant from the Kyung Hee University in 2009 (KHU-20090437).

References

[1] B.W. Lampson, Dynamic protection structures, in: AFIPS Conference, Las Vegas, Nevada, 1969, pp. 27–38.
[2] L.J. LaPadula, D.E. Bell, Secure Computer Systems: Mathematical Foundation, ESD-TR-73-278, vol. 1, ESD/AFSC, Hansom AFB, Bedford, Mass., 1973.
[3] DoD Trusted Computer System Evaluation Criteria (TCSEC), DoD 5200.28-STD Foundations, MITRE Technical Report 2547, 1973.
[4] D.R. Ferraiolo, S. Sandhu, D.R. Kuhn, R. Chandramouli, Proposed NIST standard for role-based access control, ACM Transactions on Information System Security 4 (3) (2001) 224–274.
[5] J. Bacon, K. Moody, W. Yao, A model of OASIS role-based access control and its support for active security, ACM Transactions on Information System Security 5 (4) (2002) 492–540.
[6] H. Zhu, K. Lu, R. Jin, A practical mandatory access control model for XML databases, Information Sciences 179 (8) (2009) 1116–1133.
[7] J. Luo, X. Ni, J. Yong, A trust degree based access control in grid environments, Information Sciences 179 (15) (2009) 2618–2628.
[8] Y. Lu, L. Zhang, J. Sun, Task-activity based access control for process collaboration environments, Computers in Industry 60 (6) (2009) 403–415.
[9] G.H. Motta, S.S. Furuie, A contextual role-based access control authorization model for electronic patient record, IEEE Transaction on Information Technology in Biomedicine 7 (3) (2003) 202–207.
[10] M.D. Rodríguez, J. Favela, E.A. Martínez, M.A. Muñoz, Location-aware access to hospital information and services, IEEE Transactions on Information Technology in Biomedicine (4) (2004) 448–455.
[11] X.H. Le, S. Lee, Y.-K. Lee, H. Lee, Activity-based access control model to hospital information, in: 13th IEEE International Conference Embedded and Real-Time Computing Systems and Applications (RTCSA 2007), Seoul, Korea, 2007, pp. 488–496.
[12] ITU-T Recommendation X.509, The Directory: Authentication Framework, International Telecommunication Union, Geneva, 2000, ISO/IEC 9594-8.
[13] D.W. Chadwick, The X.509 privilege management infrastructure, in: Proceedings of the NATO Advanced Networking Workshop on Advanced Security Technologies in Networking, Bled, Slovenia, 2003, pp. 1–11.
[14] D.W. Chadwick, A. Otenko, The PERMIS X.509 role based privilege management infrastructure, in: Seventh ACM Symposium on Access Control Models and Technologies, 2002.
[15] XAMCL and OASIS Security Services Technical Committee, eXtendible Access Control Markup Language Committee Specification 2.0, February 2005.
[16] J.E. Bardram, Activity-based computing: lessons learned and open issues, in: Workshop on Activity – From a Theoretical to a Computational Construct (ECSCW 2005), Paris, France, 2005, pp. 1–5.
[17] X.H. Le, S. Lee, T. Phan, V. La, A. Khattak, M. Han, H. Dang, M. Hassan, M. Kim, K. Koo, Y.K. Lee, E.N. Huh, Secured WSN-integrated cloud computing for u-Life care, in: Seventh Annual IEEE Consumer Communication and Networking Conference (CCNC), January 9–12, Las Vegas, 2010, pp. 1–2.
[18] L.T. Vinh, S. Lee, X.H. Le, H.Q. Ngo, H. Kim, M. Han, Y.-K. Lee, Semi-Markov conditional random fields for accelerometer-based activity recognition, Applied Intelligence, Springer, March 2010, pp. 1–16.
[19] J. Sarkar, Y.K. Lee, S. Lee, A smoothed naive bayes based classifier for activity recognition, Journal of IETE Technical 27 (2010) 107–126.