Authorization policies: Using Decision Support System for context-aware protection of user's private data

Arnaud Oglaza, Romain Laborde, Pascale Zaraté
University of Toulouse IRIT UMR 5505, Toulouse, France
{oglaza, laborde, zarate}@irit.fr

***Abstract*—Nowadays privacy in ambient system is a real issue. Users will have to control their data more and more in the future. Current security systems don't support a strong constraint: policy writers are non-technical users and not security experts. We propose in this paper to use Decision Support techniques and more specifically Multi-Criteria Decision Analysis in the process of authorization policy writing. This research area provides techniques to inform and assist non-technical users to write their own authorization policies following the paradigm of Attribute-Based Access Control.**

***Index Terms*—privacy; authorization policy writing; decision support system; attribute-based access control**

I. Introduction

Ambient intelligence defines the world as flooded by electronic devices and computer. These devices are interconnected, context aware and have a certain degree of intelligence, in order to make our daily environment easier. Today, this vision is not restricted to a closed environment like a house or a building but could open itself on every connected entities based on existing network technologies and in particular on the Internet to form what is called Internet of Things.

The success of such a system requires it to be as transparent as possible. As consequence, information will be more or less spontaneously exchanged between various entities to build their context in order to adapt their behavior (information on users, on environment, on system's entities, etc.). However, as emphasized by ITU in his report on Internet of Things [1]: "Invisible and constant data exchange between things and people, and between things and other things, will occur unknown to the owners and originators of such data. The sheer scale and capacity of the new technologies will magnify this problem. Who will ultimately control the data collected by all the eyes and ears embedded in the environment surrounding us?" People will have to control the access to their information in a complex and moving environment. Thus, they will have to write complex authorization policies themselves.

There exist authorization systems that provide both a very expressive policy language and adaptable enforcement architecture like XACML [2][3]. However, the complexity of the system to control (i.e. the nature and the number of security factors to consider) combined to the complexity of the language (e.g., XACML is a verbose policy language) make this solution not conceivable.

Several research works have been conducted to simplify the users' interaction with their electronic security. For example, the P3P project ("Platform for Privacy Preferences" [4]) has defined a standard to simplify users' data confidentiality policies of web sites to allow people to understand how web sites manage their data. These policies are then evaluated according to users' preferences by ad-hoc mechanisms. Reaching the same goal, Kelley et al. [5] have proposed an approach similar to nutritional labels to represent confidentiality policies of web sites. Inglesant et al. [6] have proposed a constrained natural language for the specification of authorization policies. Stepien et al. [7] have worked on a non-technical notation for XACML policies.

Although facilitating the understanding of how entities will use users' data (e.g. P3P) and making authorization policy language more human readable are mandatory, the global authorization policy writing process involving people should consider the following requirements:

- Req1: Users won't spend too much time configuring their devices before using it. The classical authorization policy writing process, which consists in 1) risk analysis, 2) policy specification, then 3) service usage, isn't acceptable any more. It has to be improved in order to take into account non-technical users.
- Req2: Users shouldn't be bothered by too many questions like "application XYZ wants to access to your calendar" when they use their devices. Interaction with users should consider users preferences to limit the interactions.

Decision Support Systems (DSS) is a research area that focuses on informing the decision maker and giving him tools and methods to model and understand the decision and give then point of solution. New trends in DSS design aim at taking into account the context of the user and the decision. For example, EUREKA [8] is a recommendation system for television that 1) analyses what and how people watch television to dynamically set up the users' preferences and 2) proposes adapted TV programs that people are expected to like. We believe DSS can allow users to write access control rules more interactively while controlling the number of interaction, for avoiding users bothering. The goal of this article is to explain how Decision Support Systems and more specifically Multi Criteria analysis can facilitate the writing of

authorization policies.

The rest of this article is organized as follows. Section 2 presents evolution of access-control models. Section 3 introduces Decision Support Systems. Section 4 presents our approach on using decision support techniques for writing authorization policies interactively. Section 5 describes the implementation of our prototype. Discussion and future work are given in section 6. Finally, we conclude in section 7.

II. Access Control Policy Models

Access control models consider three main entities: the subjects, the objects and the permissions. Subjects are users or applications who can perform actions in the system. Objects are resources or services that subjects want to access. Permissions determine how subjects can access resources. Even if these three entities appear in all models, their representation has evolved over time to adapt to requirements of modern systems. From mandatory and discretionary access control to attributes based models, we will see that the more their power of expression increases, the more the difficulty to write policies for non-technical users increases too.

A. Discretionary and Mandatory Access Control

Discretionary access control (DAC) is presented by TCSEC (Trusted Computer System Evaluation Criteria) as "a means of restricting access to objects based on the identity of subjects and/or groups to which they belong. The controls are discretionary in the sense that a subject with a certain access permission is capable of passing on that permission (perhaps indirectly) to any other subject (unless restrained by mandatory access control)". Rules are described by a triplet <user, object, action>. Each object in the system has an owner who determines the access control policy for his objects. This approach is adapted to non-technical users because policy rules are simple. However, such policies can't handle modern security requirements such as contextual information.

Unlike DAC, Mandatory Access control (MAC) doesn't let users choose permissions of resources. An administrator is in charge of building policies. Users and objects are grouped in different levels of security. The objective is to avoid information leaks by prohibiting access for a user of a certain level for an object of an upper level. This kind of policies requires a specification step before using the system for defining the security levels. This task isn't effortless. For example, security levels in the Bell-Lapadula model [9] are a combination of classification levels and compartments.

B. Role-Based Access Control

The concept of Role-Based Access control (RBAC) appeared in the nineties at the same time than multi-users applications [10]. A role represents a job function in the context of an organization. Assigning permissions to roles instead of users has considerably simplified permissions management as the number of users has grown. A single user can have multiple roles and can even switch between them depending on the actions he wants to perform.

However, RBAC doesn't take into account the environment or the situation of a user. As consequence, extensions of RBAC have been proposed to overcome this issue [11]. For example, GRBAC allows defining environment roles [12]. The model is able to capture the state of the world and described it with variables. Conditions are made with these variables and are used to activate or not the environment rules. This gives administrators the ability to elaborate several rules for the same resources but in different context.

With RBAC and its extensions, an administrator can manage a big amount of users and some contextual information. But an expert administrator is required for writing such policies. In fact, creating a role hierarchy can be a hard task when using a role engineering methodology is mandatory [13].

C. Attribute-based Access Control

Roles aren't always efficient in a dynamic, open and context-aware system to structure authorization policies. A new model based on attributes (ABAC) has been proposed. The administrator is no longer required to add user one by one but it is still necessary to specify attributes and to define access rules. Attributes can be used to detail users but also resources or the environment. All these elements can define the context of a situation. Context is any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves [14]. For example, security rules in CABAC (Contextual Attribute Based Access Control [15]) are described by a tuple <Action, UserAttr, ObjAttr, EnvAttr, TransacAttr, Perm>. Flexibility is increased for administrators who do no longer assign permissions to roles but to contextual situations.

This model is interesting as fine-grained constraints on contextual information can be specified. It allows writing complex and accurate rules. But thinking that non-technical users can write their own security rules based on such models is illusory. Thus, we propose to support users in this very complex task through DSS development.

III. Decision Support System

Nowadays, it is common to have to take hard decisions. Making a good choice requires to be well informed about this decision. In order to help the decision maker, DSS propose tools and techniques. The goal of these systems isn't to make decisions instead of the decision maker but to interactively design the decision between the user and the system. The DSS informs the user about his possibilities and guides him during the solving problem process. Several techniques are used for developing DSSs [16]. One of these is based on the analysis of the decision to make on several aspects. These aspects are called criteria and the corresponding area is called Multi-Criteria Decision Analysis (MCDA).

When people have to make a decision, it is generally possible to analyze this decision along several criteria. Multi-criteria Decision analysis proposes tools to analyze and un-

derstand this decision based on multiple criteria. Information and assistance are provided to the decision maker in order to capture his preferences. These preferences are represented by a numeric function called the utility function [16]. Using this function in the context of Recommender Systems allows assigning scores to choices in order to rank these choices from the less desirable to the most desirable. Before building the function, two conditions need to be verified. Preferences of the decision maker must be numerically representable and the multi-criteria function must be decomposable into single-criteria functions. A multi-criteria utility function has the following form:

$$\forall x = (x_1, x_2, x_3), y = (y_1, y_2, y_3) \in X \quad (1)$$

$$x \succcurlyeq y \Leftrightarrow u(x_1, x_2, x_3) \geq u(y_1, y_2, y_3) \quad (2)$$

Where x, y are choices, x_i, y_i are criteria and X the set of alternatives. Each choice implies one or multiple consequences. Decisions are then classified according to the knowledge of the decision maker. The decision maker's knowledge is ordered in three categories from complete knowledge to complete ignorance. When the decision maker has full knowledge, he knows what will be the consequences of any action. In this case, the environment is deterministic and this decision is called decision under certainty. When there isn't a total knowledge of the situation, one action can lead to many consequences. Each of these consequences has a probability of happening. It is called decision under risk. Finally, when the decision maker has no idea of what can happen, each action can have multiple consequences but he can't estimate the probability of occurring. This model is called decision under uncertainty.

As explained previously, it is necessary to be able to decompose multi-criteria function into single-criteria function, easier to encode. Then encoding these functions allows assigning score to each function. Encoding is performed by aggregation function. They are used to combine several numerical values and for the result to take into account user's preferences. An aggregation function can be a very simple mathematical function like a weighted mean or more complex like a Choquet integral that is used to take into account relationships among criteria [17].

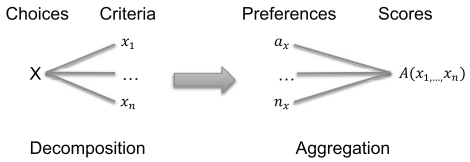

Fig. 1. Decision Support System Diagram

In summary DSS supplies tools to users in order to simplify their decision-making. Everything starts with choices. These choices are decomposed into criteria, easier to deal with. These criteria describe the choice and will allow the system to understand the reason of the user's choice. Then the system uses these criteria and an utility function to numerically represent the user's preferences. Finally preferences are aggregate by a function to build the score that is needed to take the decision. These scores are useful to propose choices to users following their preferences and what criteria are important for them.

IV. Our proposal

As part of the French ANR project INCOME (Multi-Scale Context Management Software Infrastructure for the Internet of Things), we begin to work on an authorization system for ambient system in charge of the protection of users' private data. One of the main constraint is that the authorization system should be adapted to non-technical users. These users aren't security experts and won't use the system if they can't exploit it immediately. At the same time, ambient systems imply to consider contextual information that will lead to write complex security policies. Our idea is neither to provide a new access control policy model, nor a new authorization system. We want to define a DSS that helps non-technical users to take advantage of such systems. In a classic XACML

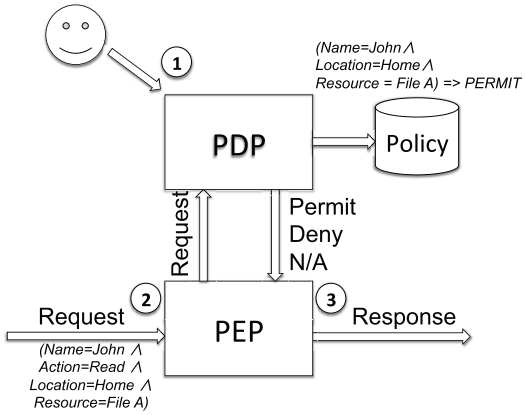

Fig. 2. Example of XACML Model

system (Figure 2), the user must first write policies (step 1). Afterward, every access request to a protected resource is caught by the Policy Enforcement Point (PEP) (step 2). In our example, the request consists in four criteria: the name of the requester, the action to be performed, the current location of the requester and the target resource. The PEP converts all this criteria to a request readable by an XACML system and sends it to the Policy Decision Point (PDP). The PDP looks in its policy database if there is a policy corresponding to the criteria. In our case, one policy exists with three criteria: the name, the location and the resource. Finally, the PDP sends the decision to the PEP who translates it into the application specific language (step 3). When looking at XACML systems, the PDP has three possible decisions: Permit, Deny or Not-Applicable. NotApplicable means there is no rule to take a decision. When the decision is NotApplicable, the PEP doesn't know what to do. To avoid this situation, there is often a default rule in the policy database that denies any unspecified access or, PEP implements response NotApplicable like Deny.

This approach doesn't respect neither requirement 1 nor requirement 2. The user has to analyze security to write complex authorization rules before using its system. In addition, he can't be informed about denied accesses if there is a default rule that denies any access. A DSS can setup an interaction with the user to help him write the missing rule interactively. If we look at our example, the request says that John is at home and wants to read file A. If the owner of file A accepts this request and other requests where John is always at home and wants to perform any action on file A, the DSS has to understand that the criterion *action* isn't relevant for the owner. It won't be interesting to propose a rule for each action John wants to perform on file A when he is at home. What really matters for the owner of the file A is that the requester is John and he is at home when he wants to act on the file. So the DSS has to propose a rule with only three criteria: the requester (John), the location (home) and the resource (file A).

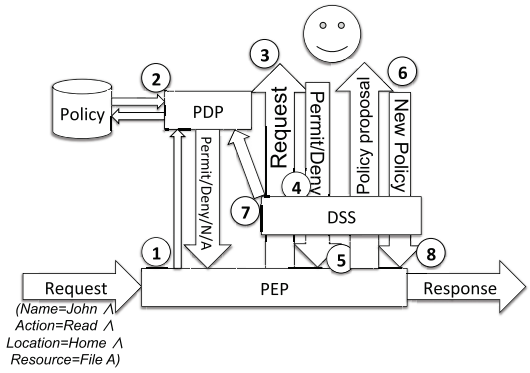

Fig. 3. Architecture of our model

In our approach, the user doesn't have to prefill the policy database. The user can create policies like an ordinary XACML system but the database can also be empty at the beginning. The system starts with the reception of a request (Figure 3) (step 1). If the PDP finds one authorization rule that matches all the criteria of the request or a subset, it will take the appropriate decision (Permit or Deny) like an ordinary authorization system (step 2). But if no rule is found by the PDP, it sends back to the PEP the response NotApplicable. At this point, our approach differs from a usual XACML system. Whenever the PEP receives the response NotApplicable, it hands over to the DSS. First the DSS calculates the score of the request (step 3). This score allows the DSS to choose between two behaviors. The first one is when the score provides enough assurance about the user's preferences. In that case, the DSS proposes to the user to write a new rule. The second behavior appears when the score doesn't give enough sureness. Then the DSS interacts with the user, informs him precisely of the request and asks for a decision.

The first behavior applies when the system is sure enough of the user preferences corresponding to criteria contained in the request. The system will interact with the user to help him write a new authorization rule. The system makes a policy proposal in an understandable sentence for the user who can accept it, change some values of the criteria or refuse it (step 4). If the user accepts to write a new authorization rule, the DSS takes this policy and adds it to the PDP policy database (step 5). Next time a request corresponding to the criteria of this policy will be made, the system will take the decision without questioning the user.

The second behavior applies when the system hasn't enough assurance about preferences concerning the request's criteria. The system must therefore learn more about them and makes an interaction phase with the user. The request is explained in an understandable sentence for the user and he is asked if he agrees or not in sharing the resource (step 6). The decision (Permit or Deny) is sent to the DSS which uses it to update all the criteria used by the request (step 7). Then, whatever the behavior is, when the decision is taken, the DSS sends it to the PEP which translates it to an application compliant form for applying the decision (step 8).

We base our work on A. Martin's research on a generic recommendation system and uses multi-criteria analysis [18]. In this approach, generic criteria are used to describe and manage recommendation objects proposed to users. Each user has profiles, one for each kind of items. An interactive approach is adopted that alternates phases of calculation and phases of dialog with the decision maker to propose him a ranked list of recommendations. The system has been tested through an online bookstore application and has showed interesting results.

In our case user's preferences are described by criteria. All criteria represent contextual information. We define contextual information as all information of a request that can be used for authorization decision and also the decision. There are seven types of criteria:

- What information is requested
- What action is requested
- When the request has been made
- Where is the resource or the owner or the requester
- Who is requesting information
- Why information is requested
- How will information be retained

In order to have a first representation of the user preferences, the system needs an initialization. Users have to answer questions relating the security of their personal data in real-life situations. Users have multiple choices for answering each questions. Each answer will be converted into a score on one or more criteria in a way that, at the end of the questioner, all of the criteria will be initialized. As seen in the introduction, users won't spend too much time configuring the system (Req1), so the questioner shouldn't be too long or the user won't complete it or won't be careful until the end. But the initialization must cover all the aspect of users preferences. The better the initialization is, the faster the system will converge to relevant preferences and will propose suitable policies to the user. And if the system converges quickly, it won't have to bother users with too many interactions with them (Req2). After this first

step, the system is ready to use anytime a request is receive. Each time a request is received, the system builds its score.

Fig. 4. The three different zones

The score of the request is a weighted average of all criteria and combination of criteria. This will allow the system to understand what criterion is really important for the user in a request. There are three different zones in the range of the score for two different kinds of zone (Figure 4). The first kind is the uncertainty zone where the system has to ask the user in order to take the decision, the second kind is a zone where the system is sure enough of the user's preferences and can take the decision itself. A request's score is between 0 and 20, the higher the score is, the less the user is reluctant to share his private information. The first zone between 5 and 15 is the uncertainty zone. In this zone, the system isn't sure enough of what the user wants so it will ask the user to take the decision in order to improve knowledge of preferences. The second zone is between 0 and 5 and is the deny zone. When a score is in this zone, the system becomes sure of the user preferences, the system knows that the user doesn't want to share data in this context so it will propose the first time the user to write a policy and when the policy is added to the PDP policy database, it will take the Deny decision itself without having to interact with the user. The last zone is between 15 and 20. This last zone works like the deny zone except that if the system has to take itself the decision, it will be permitted. In these two zones, when a decision is taken directly by the system, all criteria will be updated (a slight decrease when the request ended in a Permit, a slight increase when the request ended in a Deny) to avoid being stuck indefinitely in this zone. Because user preferences can change, the system has to pass regularly in the uncertainty zone to confirm or affirm these preferences.

When a request is received and no policy fits for its criteria, the system builds the score of the request. If the score is in the deny or permit zone, the system suggests an authorization policy to the user and if the user accepts the policy, it will be added to the policy database and the decision can be taken. If the score of the request is in the uncertainty zone, the system can't take the decision and informs the user of what happen so he can take a decision. The decision is just "yes I agree to share these information with the requester" or "no I don't want to share these information with the requester". After that answer, the system updates all the request criteria's.

The addition of the DSS allows to understand user's behavior and what matters him for taking his decision. Returning to our request's example where John wants to read the file A while he is at home. If John makes several requests while he is at home where he wants to read or write on file A. Suppose the owner of file A always accepts the requests, the criterion corresponding to his home location (where), the file A (what resource) and John (who) are going to increase quickly. Because the action write or read differs from one to another request, the criteria concerning these actions (what action) are going to increase slower than the other and because we use all combination of criteria in the score of the request, here, the combination <who, what resource, where> who is always the same is going to have quickly a higher score than all other combination containing the action. When enough requests will be made to be sure of the user's preference, the system will propose a policy where John can make any action to file A when he is at home. Thus the system won't ask the user to write two policies, one for him to read and the other to write on file A. It reduces the number of interactions needed with the user, the system is more consistent with the requirement 2.

V. Implementation

In order to make experiments, we have implemented a prototype of our system. We have chosen to rely on Android [19]. Android is the world's most used mobile platform designed to be open. However Android doesn't permit dynamic access control during runtime. We had to modify the source code of the operating system in order to allow our system to work. All installed applications in Android have a set of permissions. Permission allows an application to access to one or many resources. All permissions for an application are defined in a file called manifest and are granted at installation. Permissions are only revoked when an application is deleted. Thus, a user can't change permissions once the application is installed. In addition, the only way to deny a subset of the permissions requested by an application is to not install it.

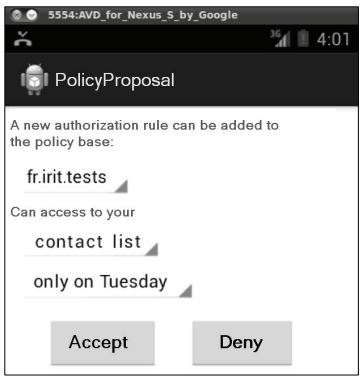

Fig. 5. Example of a policy proposal interaction

Our system needs to control and modify permissions during runtime. Because the Android SDK doesn't provide tools for that, we had to get into the source code of the operating system and change some part of it. Each time a request for a resource

is made by an application, Android checks if this application has the permission in its manifest, the main modification we made was to check before if our system has revoked this permission. This isn't the only modification we made in the source code, we also added calls to application for interactions with the user so when a request for a resource is made and no policy is compatible with it, a new window pops up to inform the user of the request and asking him to allow or deny the sharing . Interaction with user is also needed when the score of the request enters in the deny or permit zone, this time the system helps the user write a policy (Figure 5). For more ease, we also included all the calculus part in the source code so just after the interception of the request, its score is calculated and the system can take the decision or hand it over to the user. When the decision has been taken, the criteria's update phase is also done in the source code.

In the end, only interactions with the user and the XACML part is done on the application side. For the prototype, we have used the Sun XACML implementation. The request is sent to an application where an instance of a PEP is installed. The PEP translates it in an XACML format and sends it to the PDP who is also installed on the application side. For the moment, only three kinds of criteria are exploited: The What is the permission requested, the Who is the PID of the process making the request and the When is the time the request is made. The PID of each process is linked to the name of applications because a same process has a new PID at each new execution. To easily test the prototype, we use the android emulator who provides a lot of tools for debugging and monitoring.

VI. Conclusion and future work

Users will have to control their data more and more in the future. Current security systems do not support a strong constraint: policy writers are non-technical users and not security experts. Our goal is to help these non-technical users to write their own authorization policies. Users should be aware of what is performed on their data. As consequence, security systems should inform and assist users to write their own authorization policies. Our idea is to make benefit of Decision Support Systems to help users in this task. We have introduced in this paper Decision Support Systems and more precisely Multi-Criteria Decision Analysis. We have presented how this research area suits this security issue.

We have presented in this paper our model and a first version of a prototype. This prototype that only considers three kinds of criteria must be completed. Future context aware system requires to handle geographic and temporal criteria. The bigger the number of criteria is, the more the analysis of these criteria will be (deducing their relationship, taking into account their diversity, etc). As a consequence, more powerful Multi-Criteria Decision Analysis are required, such as the Choquet integral. Initialization is a very important step of the system that can help preferences to converge faster and thus bother the user less often. We are currently working on a user study concerning the initialization. We want to analyze what kind of initialization will give us the best first representation of the users preferences. In order to do that, we have prepared a survey where users are first asked to answer questions putting them in real life situations. Each of these questions is related to one or more context element like a type of resource, time or location. After that, we ask users to tell us if they are willing to share some kind of resource. We use sliders that goes to "I dont want to share this resource, no matter what" to "I want to share this resource in any case" . After these two steps, we compare results and if some results are totally different in the two version, a third step begin and asks the user what he really means. This study will help us to know what type of initialization give the best results between the questionnaire and sliders.

References

[1] ITU Internet Reports 2005, *"The internet of Things"*, 7th edition, 2005.

[2] OASIS XACML committee, *"eXtensible Access Control Markup Language (XACML) Version 2.0"*, URL: http://www.oasis-open.org/committees/xacml/, last access may 2013.

[3] Cheaito, M., Laborde, R., Barrere, F., Benzekri, A. *"A deployment framework for self-contained policies"*, (2010), Network and Service Management (CNSM).

[4] W3C (2011). The platform for privacy preferences project (p3p). http://www.w3.org/P3P/, last access may 2013.

[5] Kelley, P.G., Cesca, L., Bresee, J., Cranor, L.F. (2009). Standardizing privacy notices: an online study of the nutrition label approach. CHI '10 Proceedings of the 28th international conference on Human factors in computing system.

[6] Inglesant, P., Sasse, M.A., Chadwick, D., Shi, L.L. Expressions of expertness: the virtuous circle of natural language for access control policy specification. (2008). SOUPS '08 Proceedings of the 4th symposium on Usable privacy and security.

[7] Stepien, B., Matwin, S., Felty, A., Advantages of a non-technical XACML notation in role-based models, in International Conference on Privacy, Security and Trust (PST), pp. 193-200, 2011.

[8] EUREKA. url: http://www.canalsat.fr/pid1358-guide-tv-presentation-eureka.html, last access may 2013.

[9] Bell, D., LaPasula, L., "Secure Computer Systems : Mathematical Foundations", Technical Report MTR-2547, Vol 1, MITRE Corporation, 1973.

[10] Sandhu, R., Coyne, E., Feinstein, H., & Youman, C. (1996). Role-based access control models. Computer, 38-47.

[11] Covington, M., Fogla, P., Zhan, Z., & Ahamad, M. (2002). A context-aware security architecture for emerging applications. (2002). Computer Security Applications Conference. Proceedings. 18th Annual, pp. 249-258.

[12] Covington, M., Long, W., Srinivasan, S., Dey, A., Ahamad, M., & Abowd, G. (2001). Securing context-aware applications using environment roles. Proceedings of the sixth ACM symposium on Access control models and technologies, pp. 10-20.

[13] G. Neumann and M. Strembeck. A scenario-driven role engineering process for functional RBAC roles, proc. of the 7th ACM Symposium on Access Control Models and Technologies, pp 33-42, 20

[14] Dey, A. (2001). Understanding and using context. Personal and ubiquitous computing.

[15] Covington, M., & Sastry, M. (2006). A contextual attribute-based access control model. On the move to meaningful Internet systems 2006: OTM 2006 workshops, pp. 1996-2006.

[16] Bouyssou, D., Dubois, D., Pirlot, M., & Prade, H. (2006). Concepts et méthodes pour l'aide la dcision 3. Lavoisier.

[17] Grabisch, M. Modelling data by the Choquet integral (2003). Information fusion in data mining; pp. 135-148.

[18] Arnaud Martin, Pascale Zaraté, Guy Camilleri. Evolution of multicriteria users' profiles by adaptive machine learning for decision support. In: Multicriteria Decision Making (MCDM 2011), june 2011.

[19] Android. url: http://www.android.com/, last access may 2013.