MASTER IN COMPUTER SCIENCE

Privacy rights management

Benats, Guillaume

Award date:
2011

Awarding institution:
University of Namur

Facultés Universitaires Notre-Dame de la Paix, Namur
Faculté d'informatique.

Année Académique 2010 - 2011

Privacy Rights Management

Guillaume Benats

Submitted in partial fulfillment of the requirements for the degree of Master in Computer Sciences

Abstract

The rapid growth in smartphone usage has driven a global demand for mobile applications. This phenomenon has created new threats to the privacy of smartphone users because the ability to install and run diverse sets of applications on the same device makes it harder for users to understand how privacy configuration policies conflict with which other applications under different situations. Especially when privacy is concerned, detecting such conflicts is challenging because even end-users cannot always precisely describe their privacy requirements under these contexts. To meet the needs of representing privacy requirements and resolving potential conflicts in privacy policies, we propose to use an extension to the P-RBAC model, which allows us to reason about plausible scenarios and weaknesses of mobile systems.
Also, in the same goal of repesentation and in a goal of porting rights expression languages to mobile applications, we will reason about ODRL, the rights expression language selected by the Open Mobile Alliance as a standard for mobile environments. We will study how we can express privacy requirements scenarios of mobile applications in such a language. This work has been evaluated using the case studies we conducted on several Android mobile applications.

La croissance rapide dans l'utilisation des téléphones intelligents, dits *"Smartphone"*, a conduit une demande globale importante dans les applications mobiles. Ce phénomène a créé de nouvelles menaces envers la vie privée des utilisateurs à cause de la capacité à installer et exécuter divers ensembles d'applications sur le même appareil. Cette capacité rend plus difficile la compréhension par les utilisateurs des conflits possibles des politiques de configuration d'une application avec celle d'une autre à un certain moment donné, et de l'impact de ces conflits sur la vie privée de ces utilisateurs. Particulièrement quand la vie privée est concernée, détecter de tels conflits est un challenge en soi car même les utilisateurs finaux ne peuvent pas toujours décrire leurs propres exigences en terme de vie privée dans de tels contextes. Pour rencontrer les besoins de représentation de ces exigences et résoudre les conflits potentiels qui pourraient émerger, nous proposons d'utiliser une extension du modèle P-RBAC, qui nous permet de raisonner à propos de scénarios et faiblesses plausibles de systèmes mobiles.
Aussi, dans ce même but de représentation et dans un autre but de porter les langages d'expression de droits aux applications mobiles, nous raisonnerons sur ODRL, le langage sélectionné par l'*Open Mobile Alliance* comme standard pour les environnements mobiles. Nous étudierons comment nous pouvons exprimer les exigences de protection de la vie privée dans un tel environement avec un tel langage, en se basant sur des scénarios concrets. Ce travail a été évalué grâce à plusieurs études de cas que nous avons conduit sur plusieurs applications mobiles Android.

Acknowledgements

This thesis is the result of a three-month internship at the Open University, in Milton Keynes, United Kingdom where I was integrated in a research project: *PriMMA - Privacy Management for Mobile Applications*. *PriMMA* has for goal to investigate privacy requirements across the general population for a specific set of ubiquitous computing technologies and will produce a reusable framework with demonstrator applications. This internship was supervised by Bashar Nuseibeh, Arosha K. Bandara and Yijun Yu.
The result of those three months was the publication of a paper for the conference *IEEE Policy'11: International Symposium on Policies for Distributed Systems and Networks*[1] which was accepted as a short paper research project which has been presented the 7th of June in Pisa for the conference. The paper was titled: *PrimAndroid: Privacy Modelling and Analysis for Android applications*.

I would like to thank some people for this opportunity. I would like to thank Jean-Noël Colin, my supervisor in the faculty for the opportunity of the internship, his help all along this same internship and the writing part that followed and also for the opportunity to have been able to present our work at the *IEEE Policy'11* conference.
I am also thankful to Arosha Bandara, Yijun Yu and Bashar Nuseibeh for the supervisation of my internship, their sharing of experience, their supervisation and their help all along this year.
As members of *PriMMA* or *Securechange* projects, I would also like to thank Thomas Keerthi, Blaine Price, T. Tun and Lukasz Jedrzejczyk for their availibilty, help and sharing of experience.

Finally, but not the least, I am thankful to my parents and my familly, for the chance to have accomplished those studies, and for their patience all along.

[1]`http://www.policy-workshop.org/`

Contents

List of Figures

List of Tables

Glossary

The definitions from this glossary are mainly restated from [40], [39] and [14]. If any from other sources, those will be explicitely cited as a footnote.

- **Access control:** A system which enables an authority to control access to areas and resources in a given physical facility or computer-based information system.
- **Algorithm:** A set of mathematical instructions that must be followed in a fixed order, and that, especially if given to a computer, will help to calculate an answer to a mathematical problem.
- **Android system:** A software framework, from Google, for mobile devices that includes an operating system, middleware and key applications.
- **Application:** A computer program or piece of software designed to perform a specific task.
- **Authentication:** To prove that something is real, true, or what people say it is.
- **Broadcast:** A message to all entities of a system.
- **Class (Java):** Self-sufficient module.
- **Component:** Modular part of a system.
- **Contacts list:** List recensing information about contacts (full name, address, tel.,...).
- **Context:** Environment of an application in terms of constraints.
- **Constraint:** Condition that a problem must satisfy.
- **Database:** A large amount of information stored in a computer system in such a way that it can be easily looked at or changed.
- **Developer-centric:** System where most of the management is delegated to the developer.
- **Directive:** An official instruction.
- **DRM:** Digital Rights Management.
- **Framework:** A supporting structure around which something can be built.
- **Functional requirements:** What a system should be able to do [2].
- **Geolocation:** Identification of the real-world geographic location of an entity.

[2] `http://dictionary.reference.com/browse/functional+requirements?r=66`

- **Graph:** A mathematical structure used to model pairwise relations between objects from a certain collection.
- **Interoperability:** Being able to exchange and make use of information.
- **iOS:** A software framework, from Apple, for mobile devices that includes an operating system, middleware and key applications.
- **Machine Learning:** The ability of a machine to improve its performance based on previous results.
- **Method (Java):** Name given in Java and other object-oriented languages to a procedure or routine associated with one or more classes.
- **Model:** A description of observed or predicted behaviour of some system, simplified by ignoring certain details.
- **Non-deterministic:** Multiple ways of processing the same input.
- **Non-functional requirements:** Criteria that can be used to judge the operation of a system, rather than specific behaviors.
- **ODRL:** Open Digital Rights Language
- **Ontology:** Formal representation of knowledge as a set of concepts within a domain, and the relationships between those concepts.
- **P-RBAC:** Privacy aware Role Based Access Control.
- **Policy:** Definite course or method of action to guide and determine present and future decisions.
- **Privacy:** The state of being free from public attention.
- **RBAC:** Role Based Access Control.
- **Requirements:** A singular documented need for a system.
- **Resource:** Virtual entity of limited availability that needs to be consumed to obtain a benefit from it.
- **Rights Expression language:** A machine-processable language used to express rights.
- **Specification:** Documented detailed requirements with which a product or service has to comply.
- **SMS:** Short Message Service.

- **Tainting:** A highlighting of certain security risks.
- **UID:** User Identification Number or Unique Identification Number.
- **User-centric:** System where most of the management is delegated to the user.a

1 INTRODUCTION

This Section places the context of this thesis. An introduction to the area is given and the problem questions are stated. We then set the limits of this work before presenting the structure of the whole thesis.

1.1 Problem Statement & Motivations

Nowadays, the notion of privacy is a central concern among ubiquitous computing and mobile systems. User data has become worthy and a lot of this data is often used without user agreement. The rapid growth of mobile systems does not simplify the problem. Moreover privacy is often relayed as a second point and not included in software requirements.

Users privacy requirements can be challenging as it depends on the subject's point of view and on his understanding and caring about privacy. The representation and expression of such requirements can also be challenging and that is where rights expression language are stepping in. But the current diversity in those implies the lack of a standard and, this way, privacy management tools generally use their own representation of user's requirements which can lead to interoperability issues.

Privacy can be an ambiguous word, it is why we have to clarify the idea beyond it. We use a description restated from [13]:

"Privacy, in this context, shall no be understood the traditional way like an intimate sphere to protect, containing a set of private or confidential information that we wish to hide. It shall be understood as the faculty of selfdetermination, autonomy and capacity of the self to execute life choices. This way, it is more precisely about informationnelle self-determination, i.e. right of the individual to know what is it known about him, which datas about him are kept, mastering flows of this data, and contrecarrer abusives usages. Thus privacy does not resume to a confidentiality quest, it is the self-mastering of self-image. Privacy management is an emanation of the right to the respect of privacy taken into the dimension of right to self-determination which is linked to it. It is the right for anyone to manage its own data."

The unwanted usage of personal information has led, in the past years, to new laws and guidelines such as UK Information Commissioner's Fair Information Principles [25] or the OECD guidelines on cross border information flows [17], both of which contains the notion of informed consent. This requires that any collection of personal data is preceded by a notification explaining what data is being collected, how long it will be held and for what purpose. Plus, users must give their explicit consent for the data to be collected. This is also the approach used by mobile applications, or in any case, by Google Android mobile operating system where a user installing an application is shown a list of resources (i.e.: data sources) required by the application and by choosing to proceed, grants permission for the resource to be accessed by the application.

A mobile system like Google Android [20], whilst it does not have the market share of Apple's iPhone, has undoubtedly created a comparable community of developers which implies having over 100K mobile applications available on the Android Marketplace. A common feature of many of these applications is the ability to store, modify and share users' personal data in one way or another.
In fact, a study by Enck et al. [15] showed that Android applications often misuse those data. Indeed, they demonstrated this by selecting 30 random popular Android applications that use location, camera or microphone data and by monitoring how these applications used this data using a custom-developed system called *TaintDroid*. They found that in 105 instances these applications transmitted data to third-party servers or external devices. Out of these 105 instances, only 37 were clearly legitimate. *TaintDroid* also revealed that 15 out of the 30 applications reported users' locations to advertising servers and 7 even collected the phone number or device ID. Such findings confirm our view that mobile applications raise issues of privacy and thus, trust, which need to be better understood.

1.2 Discussion

What are we trying to protect when we talk about privacy management on a mobile phone?

- User information (identity, phone number, home adress, email adress,...)
- User personal data (SMS, pictures, videos, contacts, agenda,...)
- Access to ressources (localisation, phone calls, SMS sending, Internet)

Among other assets probably, also depending whether or not a user considers a particular asset as confidential or not. Some people will say that anybody can geolocate them but some people may think that it is a privacy issue. So, a mobile phone must provide tools to ensure that everybody, in every context, can find balance between privacy protection and information disclosure. It is why a user-centric approach seems, for now, the best way to catch users' requirements and take their usage context into account.

1.3 Research Questions

Based on the above introduction and discussion, the purpose of this work is to adopt a better view of how privacy can be better managed in mobile applications. Research questions can thus be stated to serve this purpose and draw a thread for this study.

1. Question 1: What are privacy issues and what are the assets to protect when talking about privacy management in mobile applications?
    - What are the limitations of current systems in terms of privacy?
    - How can we use tools are our disposal to override those problems?

2. Question 2: How can we use access controls models to reason about privacy issues in mobile applications and how can we adapt those models to our purpose?
3. Question 3: What can rights expression languages offer to such models in terms of expressiveness, privacy evolution and usage of a standard?
4. Question 4: How can we apply those adapted models to a specific environment like Google's Android system? How can we incorporate rights expression languages in such a system?

1.4 Thesis Limits

The world of mobile applications is quite huge nowadays and there is a wide range of privacy issues associated to usage of mobile devices. Regarding this and the limited amount of time we were able to spend on this research, this thesis is limited to studying privacy issues of Android applications, giving propositions of solutions to solve some of them and studying the expression of rights in Android via a known standard rights expression language. Although, privacy issues and proposed solutions can easily be adapted to all mobile systems (actual mobile systems though) and the approach taken is generic. Other mobile systems are also mentionned. Plus, using a rights expression language as a standard for mobile privacy can be quite useful for research on interoperability and expansion of user's control over privacy.
We also consider a user-centric way of privacy management improvement, to stay in Android's spirit but we admit the limitation that this might add overhead to the users, so the prototype has to be intuitive and light. Also, we can consider that a user-centric way for having a fine-grained control over privacy allows users who cares about their data to manage it better, and those who do not, to simply ignore this user management and relay on the base system security.

1.5 Thesis Structure

In Section II, we will define the context and background information related to this thesis, ending up with a study of Android's privacy. In Section III, we will review the state-of-the art of privacy management, rights expression languages and privacy in mobile applications. We will see how Android is dealing with privacy and how its competitors are acting differently.

The Section IV will introduce our contribution by, first, identifying some gaps in the literature presented in Section III. Then we will introduce privacy aware role-based access control as a mean of expressing some scenarios of our mobile system. This will lead to the introduction of a dependency-aware privacy management for Android applications.

The Section V has for goal to express the scenarios of the Section IV in a standard rights expression language like ODRL and will thus allow the expression of any mobile scenario. The last Section, before conclusion, will be the presentation of a prototype for the validation of our ideas. Conclusion will summarize the contribution, put forward limits of our work and proposes some future works.

2 BACKGROUND DEFINITION

Now that the problem is stated and that the subject has been introduced along with the motivations, we need to make a short background of the area of expertise beginning with a small introduction to the law about privacy and going on with a presentation of Android system which will be our main case study in this thesis. Finally, we will propose a rapid preface of what a rights expression language is.

2.1 Definition and Laws about Privacy

The word *privacy* is often ambiguous, misused or misunderstood in a law point-of-view and there are a lot of definitions depending on the context where the term is applicable. It is why we should take a more abstract definition, to generalize the idea. We have chosen *Altman's conceptualization of privacy*:

"Privacy is the selective control of access to the self regulated as dialectic and dynamic processes that include multimechanistic optimizing behaviors."[24]

That definition of privacy contains quite a few interesting words:
We can interpret the *"selective control"* as a way to say that it is a case by case selection of... *"access to the self"*, meaning a case by case choice of personal disclosure. In other words, a way to select which part of ourselves we open, and which we do not.
This process is *"dynamic"* as it is changing through time and contexts. We do not have the same intent of information disclosure with friends than with the cashier.
But this process is also *"dialectic"* as the interpretion of privacy of one person can be totaly opposite to the interpretion of one other. It is determined by how people care about their privacy and how their context is influencing their choices.

The *"optimizing behavior"* is due to the fact that if people care about privacy it is to gain a personal satisfaction and a balance of what they whish to deliver and what they do not. We thus can speak of privacy as a way to optimize personal behavior.

As conventional approaches to protecting users' privacy have been based on guidelines and the notion of informed consent (see [36]), it can be useful to make a rapid quote of current laws in terms of privacy, at least in Europe. The point here is not to pretend to be exhaustive but to have an idea of how law deals with privacy issues.

2.1.1 European Law

We will here only focus on European law as privacy regulations are quite different from one country to one other and also strike multiple domains of activity.

In Europe, the law regarding the processing of personal data and the free movement of such

data is called: *"Data Protection Directive"* (DPD). It is the most important component of privacy and human rights associated in EU.
Our continent is undoubdetly the most developed area of privacy laws. From [1], we retrieve: *"Before the DPD, all the member of the European Union (EU) were signatories of the European Convention on Human Rights (ECHR). Article 8 of the ECHR provides a right to respect for one's "private and family life, his home and his correspondence," subject to certain restrictions. The European Court of Human Rights has given this article a very broad interpretation in its jurisprudence.*

In 1980, in an effort to create a comprehensive data protection system throughout Europe, the Organization for Economic Cooperation and Development (OECD) issued its "Recommendations of the Council Concerning Guidelines Governing the Protection of Privacy and Trans-Border Flows of Personal Data" [17]. The seven principles governing the OECD's recommendations for protection of personal data were:

- *"Notice—data subjects should be given notice when their data is being collected."*
- *"Purpose—data should only be used for the purpose stated and not for any other purposes."*
- *"Consent—data should not be disclosed without the data subject's consent."*
- *"Security—collected data should be kept secure from any potential abuses."*
- *"Disclosure—data subjects should be informed as to who is collecting their data."*
- *"Access—data subjects should be allowed to access their data and make corrections to any inaccurate data."*
- *"Accountability—data subjects should have a method available to them to hold data collectors accountable for following the above principles."*

The OECD Guidelines, however, were nonbinding, and data privacy laws still varied widely across Europe. The European Commission realised that diverging data protection legislation amongst EU member states impeded the free flow of data within the EU and accordingly proposed the Data Protection Directive.

The main lines of this directive are the following:
"Personal data should not be processed at all, except when certain conditions are met. These conditions fall into three categories: transparency, legitimate purpose and proportionality."

Where:

- *"Transparency is the data subject having the right to be informed when his personal data is being processed. The controller must provide his name and address, the purpose of processing, the recipients of the data and all other information required to ensure the processing is fair. (art. 10 and 11)."*
- *"Legitimate purpose is the fact that personal data can only be processed for specified explicit and legitimate purposes and may not be processed further in a way incompatible with those purposes. (art. 6 b)"*
- *"Proportionality is the fact that personal data may be processed only insofar as it is adequate, relevant and not excessive in relation to the purposes for which they are collected and/or further processed."*

As said before in Section 1, the notion of informed consent primes in the EU law about privacy. Indeed, in the first guidelines of the OECD is stated: *"data subjects should be given notice when their data is being collected"* and *"data should not be disclosed without the data subject's consent"* which mean the subject has to be informed and has to give his consent. In the Data Protection Directive, we retrieve: *"Transparency is the data subject having the right to be informed when his personal data is being processed."*, plus, the notion of legitimation which imposes to use data only for the consented purpose.

The OECD guidelines have guided a lot of researchs on privacy management and on data protection. It is why we find important to have a complete look over this directive, which can thus be found in Appendix A in a summarize or yet on the OECD website [35].

2.2 Android System

As the main case study of this thesis is Android applications, we need to introduce this mobile operating system, describe how it works globally and see how user's data is handled in such a system.

2.2.1 Android Global Architecture

"Google's Android system is a comprehensive software framework for mobile devices (Smartphones, tablets, ...). Android includes an operating system, a middleware and a set of applications rounding it." [42]

Android is a multi-process system based on a linux kernel in which each application runs with its own `UID` (user id). Applications are written in Java and compiled in a Dalvik EXectuable which is a custom byte-code particular to Android. Each application is distributed under the form of a package, with ".apk" extension, which is similar to ".jar" files in Java.
Android applications are a set of components. They do not declare a main method where

everything starts. Each single thing the user can do is handled by components named activities, and each one of those can be called by another application if this last application has the right permission to use it or if this activity does not ask for any permission.
Each application by default has no permission granted. Permissions allow applications to perform specific operations on phone's resources and invoke activities from other applications, see Figure 1.

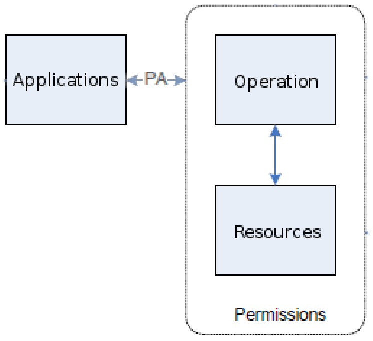
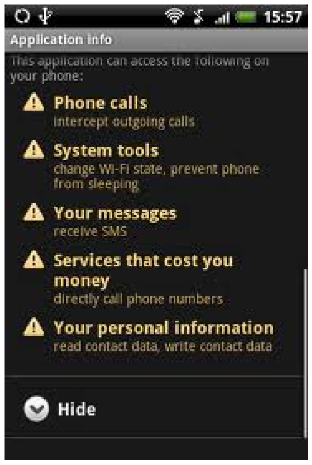


Figure 1: Android Permissions and Resources


15:57

Application info

This application can access the following on your phone:

Phone calls
intercept outgoing calls

System tools
change Wi-Fi state, prevent phone from sleeping

Your messages
receive SMS

Services that cost you money
directly call phone numbers

Your personal information
read contact data, write contact data

Hide

Figure 2: Android Application Installation: Allowing Permissions

The only way for an application to get a particular permission is to include it in the file *"AndroidManifest.xml"*. This file is at the root of every application package and is mandatory. All permissions included in this file are then presented to user when installing the application. Either the user grants all of them or he refuses the installation (which is a

first limitation of Android in terms of access control policies). See Figure 2 for a screenshot of the instalation validation.
There is about hundred permissions provided by Android system but each application can declare its own set of permissions to prevent any other one from using any activity of this application without user's consent. In practice, very few applications declare such permissions. Example of Android permissions are: write_sms, internet, call_phone, which are respectively used to grant access to the sending of SMS, to Internet and to calling features of the mobile device.

The mechanism which forces each Android application to run with a different `UID` is know as sandboxes. A particular `UID` is assigned when the application is installed and stays the same for all the application's life. So, two applications launched are running in two different processes. One application can not touch other applications. The only way for two applications to share the same resources and eventually the same process is to share the user ID thanks to a feature of the system named *sharedUserIDFeature*. This feature can be declared in the manifest and allow two applications (or more) to share the same user id. It could lead to security issues, but we'll be back later on this. As each ".apk" package is signed with developper's private key, all the application packages willing to use shared id feature must also bear the same digital signature. Otherwise, each ".apk" package would be able to use `UID` of another application and thus access files and granted resources of this one.

Along with resources like GPS and Internet connexion, permissions allow to use content providers. Developer's guide describes them as applications *"storing and retrieving data and making it accessible to all applications. They're the only way to share data across applications; there's no common storage area that all Android packages can access."* [19]. There are several content providers on the base system, for pictures, videos, contacts and so on. Each one can only be accessed if the requesting application has the permission to access the resource delivered by the content provider. Example: An application can only use contacts content provider if this application has the right to read the contacts list.

2.2.2 Android Privacy Study

As the *TaintDroid* [15] study revealed, Android suffers from privacy issues through the wide community of developers surrounding the framework. Some developers are trustworthy, some are not and some in-between.
We have identified several significant shortcomings in Android's approach to privacy management that can cause those issues.

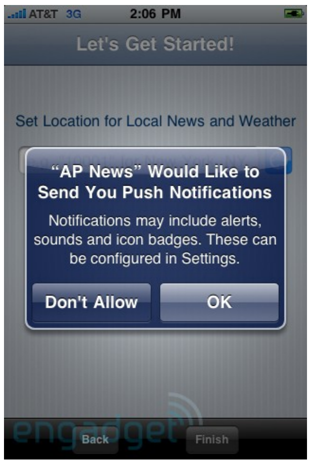
After having installed a particular application, users are never asked again for permission to access resources. This ignores the fact that mobile applications are used in changing contexts and users might want to change the access given to the application after installation.

Plus, a user can not choose, at the installation or after, to allow only a subset of asked permissions to an application or to conditionally allow those. It is either granting all permissions, either refusing the installation.
As said before, applications are asking for permissions to access resources but it is never explicitely explained *why* permissions asked are relevant. It can be clear for computer scientists that a game like chess-game does not need permission to access GPS location but it might not be that clear for anybody else.
As a last but major point, we have addressed the fact that two applications can share the same user ID with the *sharedUserID* feature at developer's disposal. The user does not give any consent to this and it allows those applications to override the permission system. If an application *"a"* has access to the contacts list and an application *"b"*, with same `UID`, has access to SMS sending, then they can, while working together, send SMS to someone in the contacts list. Thus two applications which seem harmless when executed alone, could access to private resources and override permissions without user to be aware of it. This will lead to dependency-aware privacy management later on.

2.2.3 Comparison with iOS

iOS from Apple is the major competitor of Android system. This platform is not as easy to fix as using the Android platform due to non open-source licences. iOS has a complete different policy in terms of applications allowed in the *App Store* (counterpart of the Android Market for iOS) and an opposite global point of view. While Android's spirit is to be user-centric, iOS tends to be developer-centric. Indeed, applications submitted to the App Store are checked by Apple for policy requirements fulfilments and then approved to be delivered by App Store, or refused. Each application installed from this marketplace is supposed trustworthy and there are only two cases where user is asked for permission: notifications and location access. If a particular application wants to send notifications to the user, user is asked to grant this permission at the installation, see Figure 3.
Also, when a particular application try to access geolocation, user is warned and asked for validation. Beyond that, at no time the user has to valid permissions or grant access to other resources. Apple believe in approving all applications that can be used on their phones and do not put faith in consensus or the individual users. There are variant opinions for this approach, some say it is for the good while others say that the Apple Store guidelines are vivid and sometimes very confusingly implemented, which contributes in them shutting their doors on the applications that have once gained the entrance pass for the App store. While this could seems more secure as applications are tested for privacy and security issues, it is not necessarily, as a study by Eric Smith [44] revealed.

In this study, we learn that devices running the iOS feature a software-readable serial number, also called a "Unique Device Identifier", or `UDID`. This kind of identifier was also found on Intel Pentium 3 processors some years ago and was proven to be a privacy violation because of the fact that any application could read this identifier and send it to third-party servers or applications. The article took a number of iPhone apps from the



"AP News" Would Like to Send You Push Notifications

Notifications may include alerts, sounds and icon badges. These can be configured in Settings.

Don't Allow | OK

Figure 3: iOS: Notification Granting

iStore in the "most popular" categorie. For these applications, they collected and analyzed the data being transmitted between installed applications and remote servers using several open source tools. The result is quite blatant, we retrieve here the conclusion of the study: *"68% of these applications were transmitting `UDIDs` to servers under the application vendor's control each time the application is launched. Furthermore, 18% of the applications tested encrypted their communications such that it was not clear what type of data was being shared. 14% of the tested applications appeared to be clean. The study also confirmed that some applications are able to link the `UDID` to a real-world identity."* [44]
As `UDIDs` can be readily linked to personally-identifiable information, the "Big Brother" concerns from the Pentium 3 era should be a concern for today's iPhone users as well. The checkup made by Apple on applications before being validated is thus not sufficient, as everything cannot be tested and as privacy issues also depends of user's interpretation of privacy.

Recentlty, another interesting discovery was made by Alasdair Alan and Pete Warden [6], stating that the iPad, is regularly recording the position of the device into a hidden file. It means that the device has stored a long list of locations and time stamps since user has ran it the first time. It seems to be clearly intentional from Apple. It has serious security and privacy implications, indeed: *"What makes this issue worse is that the file is unencrypted and unprotected, and it's on any machine you've synchronized with your iOS device. It can also be easily accessed on the device itself if it falls into the wrong hands. Anybody with access to this file knows where you've been over the last year, since iOS 4 was released."*

Even if this thesis takes Android as a case study, the conclusions and issues of privacy discussed can be easily applied and found in other mobile operating systems like iOS. This marks the need for a more intense privacy management for mobile applications and mobile environments.

2.3 Rights Expression Languages

The second goal of this thesis is to include rights expression languages in mobile applications. We thus need to give a little background about those languages. We will not be exhaustive here but just explain what such a language is and compare some of the most used REL's nowadays.

"A Rights Expression Language (REL) is a machine-readable language that declares rights and permissions."[5]

Each rights expression language can be based on different patterns, most common are: rules-based rights expression language, ontology-based, OO-based,.... REL's have often an *XML* representation wich is the format (generally) used to store those expressions.
There is a lot of rights expression languages but we will only present here those wich have a privacy aspect included or a privacy perspective.

The first one and the one we will use in our case study is the *Open Digital Rights Language (ODRL)*. This REL was adopted by the Open Mobile Alliance [3] as the REL used in their DRM specifications and new mobile phone handsets. This makes this REL a good way to study rights expression languages on mobile applications. The community behind ODRL is also quite active and a working group is trying to improve the language for further versions. One of the draft of this working group is regarding expression of permissions and privacy.

"ODRL is intended to provide flexible and interoperable mechanisms and they claim a transparent and innovative support for use of digital resources in publishing, distributing and consuming of electronic publications, digital images, audio and movies, learning objects, computer software and other creations in digital form. ODRL has no license requirements and is available in the spirit of open source software." [45]

ODRL is ontology-based and is composed of a formal expression language and a data dictionary.

Another major REL is *XrML* or *eXtensible Rights Markup Language* (see [10]) which has also been standardized as the REL for *MPEG-21* digital rights management [27]. *XrML* is also XML-based and describes rights, fees and conditions together with message integrity and entity authentication information. *XrML* was one of the first of his kind and is more dedicated to digital rights management than privacy or mobile applications.

[3] `http://www.openmobilealliance.org/`

XACML [2] is another rights expression language which is a new standard by OASIS (Organization for the Advancement of Structured Information Standards [4]) and is focusing on encoded data exchanges, with a simple, flexible way to express and enforce access control policies in a variety of environments. The language is mainly targeting, as baseline, supply chains and federated networks to resolve trust issues: *"How does automated enterprise software know whether to trust requests for protected information, merchandise, or credit?"*

If we focus on semantic web (i.e.: methods and tools to formalize World Wide Web), the REL adressing privacy issues the most is *APPEL (A P3P Preference Exchange Language)* [3]. It allows users to express privacy preferences via their browsers and this way compare their privacy policies with the *P3P* policies of visited websites.
But this rights expression language is subject to controverse and poorly used. Indeed, *P3P* is often described as an unnecessary complicated structure with a lack of formal specification (as a lot of RELs though), and missing guidelines for user agents.
Plus, *P3P* does not ensure that the policy declared on a website is complete and compliant to what the website actually do with user data. As a mainline conclusion for *P3P*, we retrieve from [41]:

"There is, at this time, no mechanism by which users can ***easily*** *verify the* ***compliance*** *of a data collection with the collector's poster privacy policy".*

[4] http://www.oasis-open.org/

3 STATE-OF-THE-ART

This Section reviews the literature related to objectives fixed for this thesis. We need first to have a look at the existing in terms of privacy management in mobile applications and especially existing privacy tools in Android. We also have to look at privacy conflicts, how to handle them with existing methods. Finally, we will review all related works regarding rights expression languages.

3.1 Privacy Management in Android

As users cannot explicitely say *why* a particular application asks for a particular permission or what this application will do with their data, and thereby cannot decide what permissions such application should run with, the need for a tool checking if an application really uses resources the way they should and the way users trust it, is growing. One of those tools is *ScanDroid* [18], for automatically reasoning about the trust issues of Android applications. This tool tracks data flows through and across components, while relying on an underlying abstract semantics for Android applications. *ScanDroid*'s analysis is modular to allow incremental checking of applications as they are installed on an Android device. It extracts security specifications from the *"AndroidManifest.xml"* that accompany such applications in the root of each ".apk" package, and checks whether data flows through those applications are consistent with those specifications. To work properly though, *ScanDroid* does need source code embedded in applications packages which is not an assumption we can make for most of the applications on the Android market. Indeed, packages are nearly always just embedding compiled byte-code, resources and librairies.

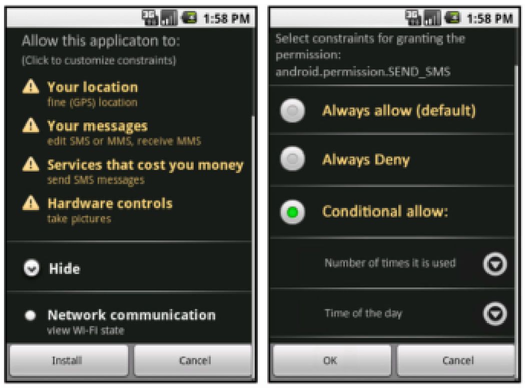
We thus need to look at a tool which can help to detect privacy issues and trust issues without any source code available.
We have already mentioned *TaintDroid* [15] as a monitoring tool studying flows of data between applications. Many smartphone applications are not open source, therefore, static source code analysis is infeasible. Even if source code is available, runtime events and configuration often dictate information use; realtime monitoring accounts for these environment specific dependencies. The key beyond *TaintDroid* is the "taint tracking" principle. Each data is tainted regarding its kind. This tracking is often performed at the instruction level. Finally, the impacted data is identified before it leaves the system at a taint sink. Here the sink is usually the network interface as the study was mainly interested in the number of applications sending data to third-party servers. *TaintDroid* has one big advantage regarding pre-existing work, that is performance. Approaches that rely on instruction-level taint analysis using whole system emulation [23] have a high impact on performance. Instruction-level instrumentation incurs 2-20 times slowdown [23] in addition to the slowdown introduced by emulation, which is not suitable for realtime analysis.
To assign labels, *TaintDroid* considers defined interfaces through which applications access sensitive data, i.e.: resources. For example, all information retrieved from GPS hardware is location-sensitive, and all information retrieved from a contact provider will be tainted

as contact-sensitive.

TaintDroid [15] is not the only study that revealed issues of privacy on this famous mobile system. Indeed, in [12] Davi et al., are studying Android's security robustness and a possible permission escalation attack overriding the security mechanisms.
By this way, they identified a severe security deficiency in Android's application-oriented mandatory access control mechanism (i.e.: the permission mechanism) that allows transitive permission usage, which we will study later with the notion of dependencies between applications. In their attack example, authors were able to *"escalate privileges granted to the application's sandbox and to send a number of text messages (SMS) to a chosen number without corresponding permissions."*

Apex [32] , by Khan, Nauman and Zhang, is a user-centric framework for privacy management in Android applications that extends the basic permission scheme of Android. This framework allows users to specify detailed runtime constraints to restrict the use of sensitive resources by applications. The user-specified constraints are checked by a special version of the Android installer called *Poly* [31]. Extensions are incorporated in the Android framework with minimal changes to the source of existing security architecture. *Apex* uses an XML rights expression format to store constraints on applications. Although the set of possible constraints a user can specify is currently limited, the authors of Apex indicate that there is ongoing work to extend this. All access policies are stored with constraints associated in a policy repository embedded in the framework.

Figure 4, that we retrieve from [32], shows the modified installation of an android application with *Apex*.


1:58 PM

Allow this applicaton to:
(Click to customize constraints)

Your location
fine (GPS) location

Your messages
edit SMS or MMS, receive MMS

Services that cost you money
send SMS messages

Hardware controls
take pictures

Hide

Network communication
view Wi-Fi state

Install | Cancel

1:58 PM

Select constraints for granting the permission:
android.permission.SEND_SMS

Always allow (default)

Always Deny

[✓ Conditional allow:]

Number of times it is used

Time of the day

OK | Cancel

Figure 4: Apex Extension Framework: Installation

This framework is the closest to our idea of user-centric privacy management for Android applications. Nevertheless, we will expose further some limits of this one and how to imagine another framework of the same idea while including a notion of dependencies between applications to override implied privacy issues (as showed the example in [12]).

Privacy is also depending on the context where applied. Context-aware privacy is thus also a challenging part of privacy management. "Context" is an ambiguous word, we thus retrieve the interpretation of Mancini, Keerthi et al. [29]:

"An experience of a particular event in the specific context within which that event took place, or rather within what users subjectively perceive as constituting the context of that experience. Because participants themselves chose the phrase that they associate to an event, the phrase is capable of triggering a connection to the experience to which it was associated and to bring participants back to that context.".

There are several interesting works to look at when speaking of context-aware privacy: In [8], Bai et al. are also studying the existing security mechanism on the Android platform and focus on the great challenges implied by the mobility and openness of mobile computing environment. They then proposes a context-aware usage control mechanism to enhance data protection and resource usage constraints on Android. Their model, ConUCON, is able to take obligations, states and contexts into consideration at usage decisions. Their framework enables the user to grant permissions in a fine-grained manner.
Always in a context-aware idea of privacy, Conti et al. [11] also present a system that is able to enforce fine-grained policies, e.g. that vary while an application is running. *"A context can be defined by the status of some variables (e.g. location, time, temperature, noise, and light), the presence of other devices, a particular interaction between the user and the smartphone, or a combination of these"*. Their framework allows policies to be defined either by the user or by trusted third parties.

All those frameworks have an impact on privacy issues on Android applications and we will see that, adding an idea of dependencies in such a system, will provide a high level fine-grained privacy management for android applications.

3.2 Policy Conflicts

User-centric privacy policies are better addressing user's subjective way of apprehending privacy but have also some disadvantages. One of them and the one we will look at here, in this Section, is conflicts of policies.
Indeed, as users are defining privacy rules themselves thanks to tools at their disposal, they could define rules conflicting with each other or impossible rules. It is why we need to look at literature to see how to handle such conflicts.

In [9], Bandara, Lupu et al. have shown how conflict detection in policies can be achieved using event calculus (see [43]) and abductive reasoning techniques. They focused on distributed systems management scenarios but can be extended to other policy-based applications like mobile applications.

Ontologies are commonly used in the world of privacy as many RELs are ontology-based and as the rights expression language we will use later on is ontology-based, we should focus on howto resolve such conflicting ontologies. Lu and Zhang [46], focus on concept restriction conflicts. Their work is focusing on a detection and elimination approach through ontology merging. They shown that two equivalent concepts in two different sources of ontologies could have different values or different restrictions. They use Tableau algorithm to detect those conflicts plus a formal reasoning to deal with them.
Tableau algorithm [7] is based on satisfiability problem which we will use in our work to detect policy rules conflicts.

3.3 Rights Expression Languages

We have now to get familliar with works in rights expression languages litterature, but we will just focus on works interesting us in the scope of this thesis as there are many subjects and researchs on this particular domain. Ianella and Governatori, in [22], worked on formal contract logic [21]. They expressed privacy issues and scenarios of social networking in a modified formal logic and discussed the problems of privacy conflicts beyond those scenarios by adding a superority relation in their formal expression of social networking scenarios. This superiority relation allows to resolve conflicts between two rules representing privacy policies defined by a user on a social network.
This work is using rights expression language, formal reasoning and user-centric privacy (as rules over privacy on a social network are set by user), and is thus working on social networks privacy, the same manner we want to work on mobile applications.
Although there are many and many RELs appearing with slighlty different purposes, main languages (ODRL, XrML,...), have undoubtedly become complex in using and getting them masterized. This is due to the fact that those RELs are trying to be scenario-exclusive, meaning trying to be able to handle a wide variety of possible scenarios for DRM, access control, privacy, permissions and so on. The result is that it is often difficult to cleanly partition out only the pieces needed for a particular purpose. In [38], Pramod et al. propose a higher layered system, subdivided in cores:

- *"The core REL should only contain the rights model."*
- *"The core REL should be stateless."*
- *"The core REL should be language neutral."*
- *"REL primitives, and DRM services in general, should refer generically to the services they use."*
- *"A DRM service should only know what it absolutely needs to know in order to complete its task."*

This conclusion is interesting as our work needs only the privacy model of the particular REL (*ODRL*) we are using and it is sometimes hard, as we will see by studying the *ODRL* model later on, to choose a component to represent a particular scenario as the possibilities are huge.

4 PRIVACY POLICY MODELLING AND ANALYSIS FOR ANDROID APPLICATIONS

This Section is centered on the first contribution of the thesis by beginning with the highlighting of gaps present in the state-of-the-art. Once gaps are identified, we introduce privacy aware role-based access control from [33] which will be our staple model to reason about permissions in Android system and privacy issues. This will lead to a dependency-aware privacy management where we will be looking at how applications are dependent of each other and what privacy issues it can raise. In such a system, with user-centric management, conflicts could arise and it is thus important to look at how we can deal with such.

4.1 Gaps in the State-of-the-art

We have looked, in the previous sections, at the main concerns of privacy issues in Android applications. We have seen that tools like *Apex, TainDroid, ScanDroid,...* are existing to resolve or find some of those issues, but a tool like *ScanDroid* or *TaintDroid* is monitoring applications and flows of data without solving those issues or without proposing an easy policy management to the user. On the other side, *Apex* does propose a framework for privacy management but some issues are still present while using this framework. First, *Apex* does not respect the fact that privacy is changing along with the context of usage. Some people would like to change their privacy profile from time to time. Indeed, *Apex* proposes only to users to add constraints on a rule or deny a permission to access a resource at the installation. If the user wants to change it beyond that step, he cannot. A second limit of *Apex*, in the scope of our work anyway, is the fact that dependencies between applications are not taken into account. Also, we would like to introduce main rights expression languages as a standard for privacy policies representation in mobile applications, and *Apex* is using its own XML-representation.
Both [11] and [8] frameworks are overriding the issue of static policies declaration by taking into account usage context applications. But, by reading works such as TaintDroid [15] or the work on escalation privileges attack [12], we can see that there is a need to take dependencies between applications into account, to avoid scenarios where some privacy issues can still appear. Plus, as *Apex*, most of the current frameworks are defining policies in their own way, with a custom-created XML-based language, which, for interoperability and privacy changes (or evolution) aspects, are kind of a limit. We will see how rights expression languages, as ODRL [26], can help override those problems.

Before looking at possible solutions and what dependency between applications mean, we need to introduce *P-RBAC* [33] which will be our model to reason about Android applications scenarios and find possible issues and possible solutions, while expressing those in a more abstract way.

4.2 Privacy aware role-based Access Control

P-RBAC [33] is an extension of role-based access control model (*RBAC*) to support privacy policies. The model is using entities used in *RBAC*: Users, Roles, Permissions (which are actions over objects) and Conditions and is extending those entities to support privacy policies. There are two new entities: Obligations and Purposes. *Obligations* are actions that must be performed if a particular permission is granted (e.g., audit logging, deleting old data, etc.). The *Purposes* entity gives the reason for granting a particular permission. The *Objects* entity of standard *RBAC* has become *Data*, meaning any information relating to users, and finally, *Conditions* specify the constraints that must be satisfied in order for a particular permission to be applicable.

As the purpose of *PRBAC* is to extend classical *RBAC* to support privacy, *PRBAC* is defined as a family of conceptual models with different capabilities. *Core P-RBAC*, the base model, is at bottom of the familly. This last model is the one we will fund our reasonning on.
The design of *Core P-RBAC* should have *"sufficient expressive power for representing public privacy policies, privacy statements,..., and policies based on privacy related acts"*[33], such as DPD in Europe or HIPPA (HIPPA act concerns protection of medical records and other health information in US).

The core *P-RBAC* components, as shown in Figure 5, contains all of the following components, which we restate from [33]:

- *A set U of users, a set R of roles, a set D of data, a set Pu of purposes, a set A of actions, a set O of obligations, and a condition language, LC_0.*
- *The set of Data Permissions, $DP = \{(a,d) \mid a \in A, d \in D\}$*
- *The set of Privacy-sensitive Data Permissions, $PDP = \{(dp, pu, c, o) \mid dp \in DP, pu \in Pu$, c is an expression of LC_0 $, o \in P(O)\}$. $P(O)$ is the power set of O.*
- *User-Role Assignment, $UA \subseteq U \times R$, a many-to-many mapping user to role assignment relation.*
- *Privacy-sensitive Data Permission Assignment, $PDPA \subseteq R \times PDP$, a many-to-many mapping privacy-sensitive data permission to role assignment relation.*

In the next Section we propose a number of modifications to the *P-RBAC* model in order to better support this permission model of Android mobile applications, thus allowing permission policies to be checked for inconsistencies and potential privacy violations. We will help ourselves with a simple scenario which will allows us to reason more easier through this Section.

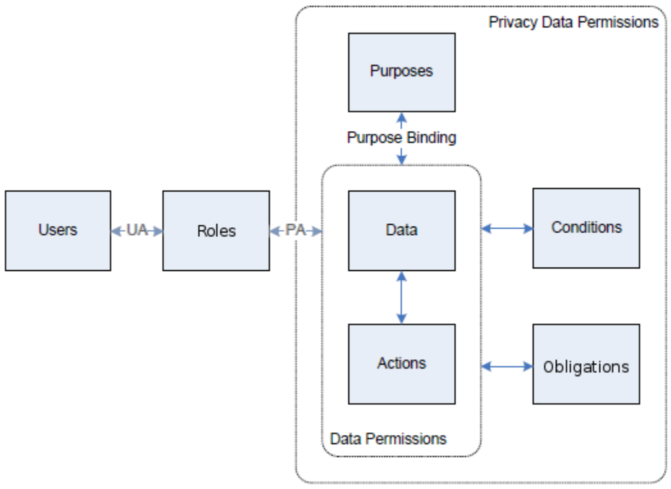


Figure 5: P-RBAC Components Overview ([33])

4.3 Android Scenario

This Section presents a concrete case study to express and deal with privacy issues mentioned in the previous sections.

Alice installs four applications on her Android phone:

- `FBContacts` handles *Facebook* accounts and synchronizes them with her phone's contacts list
- `GPSFriends`, is an application using *Google Maps* to locate friends who are using the same application.
- `FreeGame`: A free game application that Alice installed by clicking an advertising link.
- `GPSWifi`: An application to share free to access Wi-Fi locations with friends.

Each application requests and obtains the following permissions at install time:

- `FBContacts`: access to the phone's contacts list
- `GPSFriends` and `GPSWifi`: access to location data and internet.
- `FreeGame`: access to the internet.

Although Alice is not aware of it, these applications are interrelated in the following way:

- `FreeGame` and `GPSFriends` are sharing the same `UID` (with sharing id feature, as we have mentioned in Section 2.2.1)
- `FBContacts` and `GPSWifi` both use `GPSFriends` to get the user's location.

Alice uses the *Apex* tool (or similar) (see Section 3.1), to specify constraints that limit the conditions under which each permission applies. These constraints are:

- `FBContacts`: can only access location data if time is between 4pm and 8pm.
- `GPSFriends`: can only access location data if location is in the UK and the time is between 8am and 4pm.
- `GPSWifi`: can only access location data if the location is in London.

4.4 Dependency Graphs

We will use dependency graphs to be able to capture dependencies between applications. Those graphs will be the base of each reasonment made about those dependencies and are thus crucial. We will introduce this idea by using our scenario presented in the above Section (see 4.3).
The dependency graph in Figure 6 shows the relationships between permissions (shown as ovals) and applications (shown as boxes) for the `FreeGame` and `GPSFriends` applications. An arrow from an application to a permission indicates those permissions that are specified in the application's manifest file and are thus granted by the user at install time. However, in this example the existence of a path from `FreeGame` to Location indicates that this application also has permission to access location data, despite not explicitly requesting this permission from the user, by using another application. Figure 7 shows the dependencies between the `GPSFriends, FBContacts` and `GPSWifi` applications.
Direct links are thus representing authorized access to resources, granted by users, to applications. Indirect links can thus be seen as applications overriding non-authorized access using the dependencies between applications to access resources they should not normaly access. In other words, indirect links represent privacy breaks in our scenarios.

This idea of dependency graphs was inspired by the work of Ferrante et al. [16] where a new program representation, called the program dependence graph or PDG, has been presented and shown to permit efficient and powerful program transformations. We have to determine how those graphs are created in our Android system and how they evolve along with the applications represented.

4.4.1 Creation of dependency graphs

We mentioned in Section 2.2 that Android permissions are assigned to applications at installation time, see Figure 1. Each application, when installed, gets all the permissions asked when granted by users.
Our dependency graphs are thus created at installation time, when *"AndroidManifest.xml"* file is parsed for permissions list and for other features like if shared `UID` is set or not. If user refuses installation, graphs are thus not created.

Constraints on the branches (like *"GPSWifi can only access location data if the location is in London"*) are also added at the installation but by the mean of *Apex* [32] (or a similar tool, we took *Apex* as example) if user decides to add some constraints on the permissions asked, displayed on the installation screen. We consider them being added by *Apex* here, but a tool like our prototype, see Section 6.7 also allows to add such constraint and not only at installation time but anytime during life of an application. If we consider such a tool, those constraints on branches could be added anytime.

4.4.2 Evolution of dependency graphs

During system's life, applications are installed, removed, updated,...Our graphs cannot thus be static, i.e.: be defined once and then not follow concerned applications' life.

If an application is removed, we need to update dependencies in all graphs where this application plays a role. Elseif, we will have inconsistencies in our graphs and the applied constraints would lead to incoherent rules and a wrong image of dependencies between applications in the current system state.
If an application is updated, two cases are possible. Either new permissions are asked and thus, presented to the user for validation (indeed, when applications are updated through the marketplace of Android, new permissions are shown to the users and he is asked for validation), and thus new graphs are created, or old ones are modified. Either permissions' scheme is not modified and dependency graphs should not be updated.

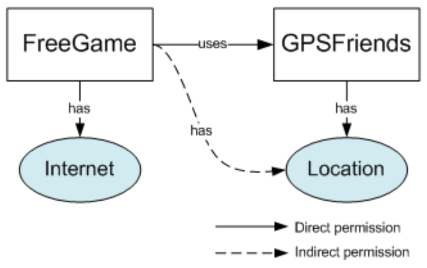


Figure 6: Relationships between Permissions and Applications

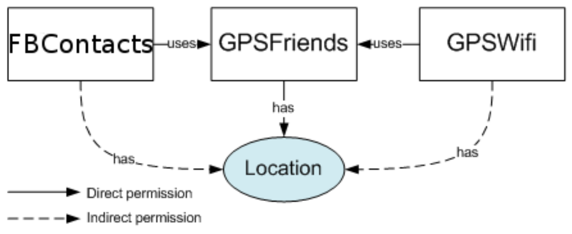


Figure 7: Dependencies between Applications

The above scenario illustrates how information in the *"AndroidManifest.xml"* file is insufficient for a user to grant informed consent for an application to access personal data. This is because the installation process does not expose how dependencies between applications can affect the actual permissions available to the applications.
In order to address this issue, we propose extending the *P-RBAC* specification given above to support automated reasoning about the actual permissions available to an application. Our modifications make it possible to take into account both dependencies and user specified constraints when reasoning about Android permissions.

4.5 P-RBAC for Mobile Applications

The core of P-RBAC can easily be adapted to fulfill mobile permissions requirements, and in our case, Android permissions requirements. This Section builds this adaptation step-by-step.

Because Android devices are typically single user devices, there is only one role to consider and therefore we can remove the concept of *"role"* from the model. This is a valid assumption for each current mobile device. Although with growth of mobile technologies and mobile demand, it may be subject to change in the future.
Additionally, we will not use here the P-RBAC concept of *"purpose"* since the purpose description associated with Android permissions is a simple text label with no defined semantic that can be used to analyse privacy policies. But we do not remove this entity from the model as it could be useful at some point to express purposes of constraints over permissions, or information related to the permission granted. We kept the *"obligation"* entity even though it is not currently supported by the Android permissions system, because it could be useful for logging access to resources made by applications and in order to give users feedback on potential privacy violations.
Indeed, logging can be used in many way's to warn user about privacy issues: discreet ways (notifications icons, logs,...) or more direct (sounds, popups,...).

The modified P-RBAC model thus contains the following components:

- A set A of applications, a set R of resources, a set X of actions, a set O of obligations and a condition language LC_0.

- The set of Resources Permissions, $RP = \{(x, r) \mid x \in X, r \in R\}$
- The set of Privacy-sensitive Resources Permission $PRP = \{(rp, c, o) \mid rp \in RP, o \in O$, c is an expression of LC_0 $\}$.
- Privacy-sensitive Resources Permission Assignment, $PRPA \subseteq A \times PDP$, a many-to-many mapping privacy-sensitive resources permission to application assignment relation.

We can now express simple Android permissions together with constraints using this model. For example, by taking the permissions and constraints defined by Alice in our case study (4.3):

The following permissions on

`FBContacts`: *"`FBContacts` can only access location between 4 p.m and 8 p.m"*,
`GPSFriends`: *"`GPSFriends` can only access location between 8 a.m and 4 p.m"*

can be expressed as follows (`ACCESS_FINE_LOCATION` denotes the Android permission needed to read the GPS sensor location):

permission(FBContacts, ((has, ACCESS_FINE_LOCATION), ($time > 16 \wedge time < 20$), Log_Access())

permission(GPSFriends, ((has, ACCESS_FINE_LOCATION), ($time > 8 \wedge time < 16$), Log_Access())

However, the above permissions expressions do not take into account the dependencies that can arise between applications. To address this limitation we need to be able to represent dependencies between applications and to this end we introduce a notion of **"dependency groups"** to the P-RBAC model. The details of this are discussed in the next Section.

4.6 Dependency-aware Privacy Management

As we explained above (see Section 2.2), Android assigns a unique user identifier (`UID`) to each application on the mobile device but applications can request the use of sharing `UID` feature. This far, we have been associating applications directly with privacy data permissions. But we wish to introduce the notion of dependencies between applications in our modified *P-RBAC* model.
We thus introduce a *"dependency group"* entity in between applications and privacy data permissions. This makes a dependency group analogous to a *"role"* in the original *RBAC* formulation.

This new *"dependency group"* entity is referring to environment of an application in terms of usage dependencies. Each application that invokes operations on other applications (i.e.: activities on Android), and applications sharing the same `UID`, should be assigned to the same dependency group. Permissions and associated constraints are then granted to the group. The user can thus constraints those permissions with conditions like before but those conditions would apply to all applications associated with the concerned dependency group.

Existing tools would assign same permissions on applications that are used together. If two or more applications are defined to be dependent of each other, they belong to the same group.

We can thus redefine our *P-RBAC* model as follows:

- A set A of applications, a set G of dependency groups, a set R of resources, a set X of actions, a set O of obligations and a condition language LC_0.
- The set of Resources Permissions $RP = \{(x, r) | x \in X, r \in R\}$
- The set of Privacy-sensitive Resources Permission $PRP = \{(rp, c, o) | rp \in RP, o \in O,$ c is an expression of $LC_0\}$.
- Application Assignment $AA \subseteq A \times G$, a many-to-many mapping application to dependency group assignment relation.
- Privacy-sensitive Resources Permission Assignment $PRPA \subseteq G \times PRP$, a many-to-many mapping privacy-sensitive resource permission to dependency group assignment relation.

Under this new formulation, Alice can now add conditions on the usage of location on `FreeGame`, since both `FreeGame` and `GPSFriends` would be in the same dependency group, sharing same privacy rules. This makes it easier to prevent unauthorized access to Alice's location information because `FreeGame` would be subject to the same constraint on location as the other application, `GPSFriends`.

A potential complication of using dependency groups is in dealing with situations where large number of applications are in the same group due to complicated dependencies between all of them. Alice's scenario is an extreme case where there is a lot of relations between applications but in a system with hundreds of applications, a lot of them would run as standalone entities or be coupled with, at most, one or two other applications. Therefore, we expect dependency groups to be of a manageable size. But, this dependency group entity can raise several questions when looking at the analogous *"role"* entity in *RBAC*:

- **What about appartenance of an application to several dependency groups?**

The question of multiple groups belonging can be quite challenging. Why? If an application is included in several groups it means that it is dependent of one or more application of each group. But as explained before, this dependency can either be a shared `UID` or a component usage. We have thus three possible scenarios to study.

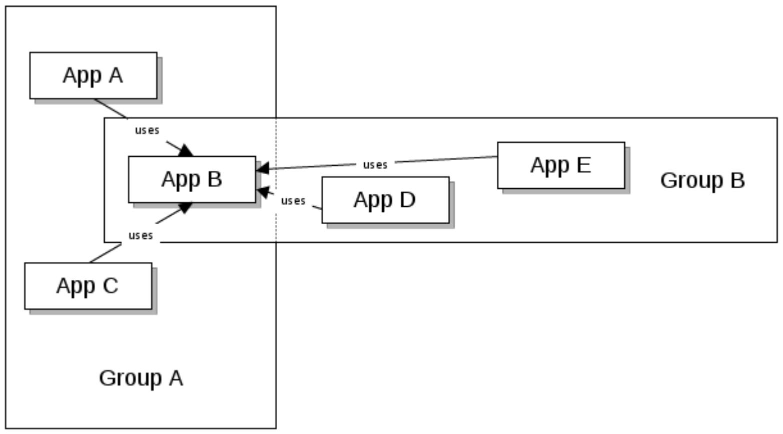


Figure 8: Multi-groups Belonging - Case 1

The first one, is shown in Figure 8. We have two groups of applications with no shared `UID`, just dependencies due to usage of "external" components. This scenario is acceptable, `App B` can belong to both `Group A` and `Group B`. Indeed, all applications of both groups are using `App B` but those are not dependent of each other and can thus be in separated groups.

The second one, is shown in Figure 9. Same scenario as Figure 8, but here, we have a shared-`UID` relationship between `App B`, and one application of `Group B`, `App E` which means that if `App B` belongs to both groups, `App E` can override constraint of its group and use the permissions granted to `Group A` and this way, our dependency group entity becomes useless. It is thus a scenario we must prohibit!

The last one, could seem quite obvious regarding if we prohibit it or not but has to be mentioned. We have the scenario of Figure 10. we have a shared-`UID` relationship between `App B`, and one application of `Group B`, `App E` and also one application of `Group A`, `App A`. Which means `App A` and `App E` can both override constraints of their respective group and use the permissions granted to the other group. This scenario has also to be prohibited!

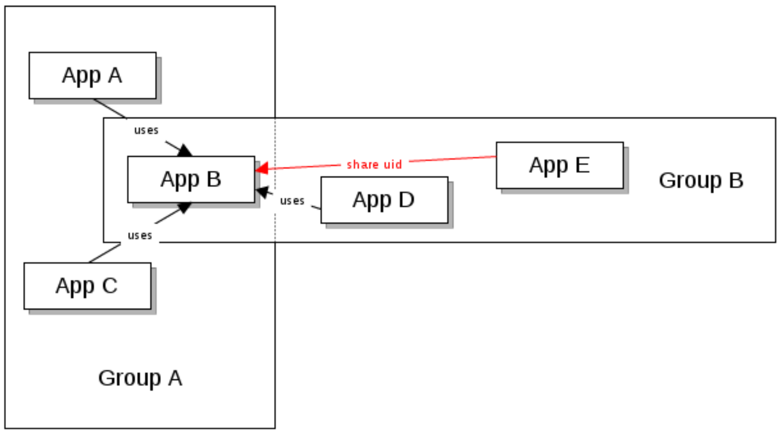


Figure 9: Multi-groups Belonging - Case 2

- **What about groups hierarchy?**

RBAC allows roles hierarchies and *P-RBAC* also defines a hierarchical component which introduces roles hierarchies, data hierarchies and purposes hierarchies (see [33]). Roles hierarchies come from *RBAC* and is often present in its extensions. But in our case, does we really need groups hierarchies? What could be the advantages and disadvantages of such hierarchies?

On Figure 11 we have three groups, A, B and C, with B and C being sons of A, hierachicaly speaking. A is subject to privacy rules A, while B and C are respectively subject to rules $A \wedge B$ and $A \wedge C$. It would mean that B and C are subject to the same permissions and constraints as A but could also have more permissions or more constraints than A, as specialized groups. It would imply a lot of groups to be divided into several subgroups and it would alter the main idea which was, groups of dependent applications under same permissions and same constraints (same privacy rules). Plus, it would growth in complexity for the end-user which would thus need to understand the concept of hierachy and would probably not want to matter with that as he would probably not see the point of such a construction. We could have an opposite discussion if this was supposed to be developer-centric.

Now let's have a look at another interesting scenario which is avoided with our dependency groups entities (see Figure 12):

Imagine that Alice has added a new condition on the permission of `GPSFriends` to access location, constraining access to location information when she is in London:

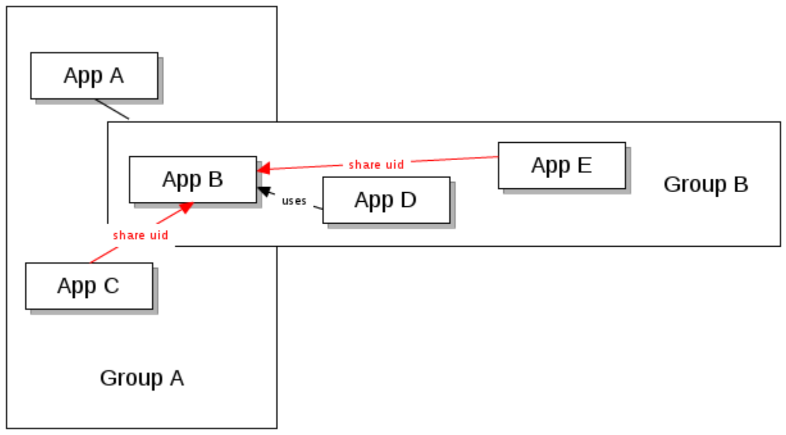


Figure 10: Multi-groups Belonging - Case 3

$Permission_1$: ($Group_1$, ((use, ACCESS_FINE_LOCATION), $Location = London$, Log_Access()))

Alice also constraints `GPSWifi` to be granted access to location information only when her location is in the UK:

$Permission_2$: ($Group_1$, ((use, ACCESS_FINE_LOCATION), $Location = UK$, Log_Access()))

However, the problem that arises is that `GPSWifi` has not been given a direct permission to use the GPS location data. Instead it accesses location data via `GPSFriends`, which now has a constraint that limits its ability to access location information to those times when Alice is in London. This means that any attempt by `GPSWifi` to access location information when Alice is in the UK, but outside of London will fail, because `GPSFriends` does not have the required permission in this situation.

There are two possible solutions to this problem, the first one being to grant `GPSWifi` a direct permission to location data which can then be constrained to be applicable whenever Alice is in the UK. Alternatively, we could define a dependency group and assign `GPSWifi` and `GPSFriends` to it. This would allow Alice to define the conditions she wanted on permissions for the whole group and not for each application separately as if they were used in completely different groups.

In addition to dependency-aware permissions management, we also have to deal with the issue of constraints conflicts that could arise when users are allowed to specify the conditions under which particular permissions arise. Indeed, Alice can now define privacy rules on a *"dependency group"* and, this way, constraints permissions of all applications inside this group. However, if the constraints she defines inside a group conflict with

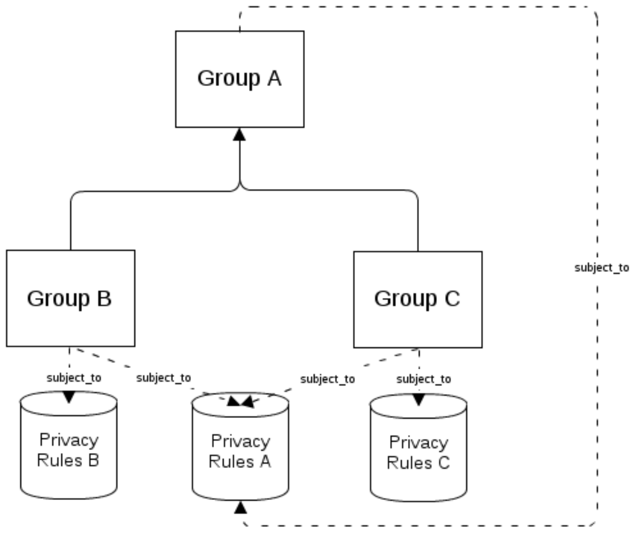


Figure 11: Groups Hierarchy

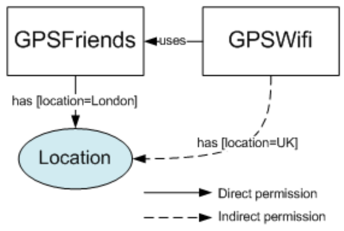


Figure 12: Privacy issue due to usage of another component

each other, all the applications belonging to the group could be prevented from accessing required resources, or would be allowed access them under the wrong circumstances. We deal with the issue of constraints conflicts in the next Section.

4.7 Policies Conflicts

We have now build a model based on *P-RBAC* able to represent mobile applications scenarios, more precisely in the scope of this thesis, Android applications scenarios, and reason about those. We have introduced the notion of groups to be able to resolve some privacy issues presented in Section 2.2.2.
But this model will be managed by users indirectly while choosing their privacy policies and allocating applications to groups. This user-centric way of managing privacy has to be taken into account because of the possible conflicts which can arise while user is defining privacy rules.
In [34], Sadeh et al. are avoiding conflicts in users' privacy rules of social networking by

allowing only rules that grant access to a resource rather than combinations of rules with some granting access and others denying it. *"For example, a person can specify 'Mary can see my location between 9 AM and 5 PM', but cannot specify rules like 'Colleagues cannot see my location on weekends'."*. But in our case, we have prohibitions and conditionnal allows, we can thus not make an assumption of this kind. Indeed, we would like to allow users to deny permissions to applications.

The set of possible conflicts that can arise in our particular model is as follows:

- **Conflicting constraints:** We will talk of conflicting constraints beyond two rules if those cannot possibly coexist.
- **Concurrent constraints:** We will talk of concurrent constraints beyond two rules if those are applied on the same resource and same constraints variables but the intersection of both rules exists and does not generate conflicts.
- **Incomparable constraints:** We will talk of incomparable constraints if they have incomparable rules, i.e. two disjoint value sets in two constraints on the same resource [33].

4.7.1 Scenarios of conflicts

We can use our scenario with Alice to reason about possible conflicts. As `FBContacts` uses `GPSFriends` to get information about location, they will end up in the same dependency group, thus the two constraints added by Alice on each application would become:

$Permission_1$: $(Group_1, ((\text{use, ACCESS\_FINE\_LOCATION}), time > 16 \wedge time < 20, \text{Log\_Access}()))$

$Permission_2$: $(Group_1, ((\text{use, ACCESS\_FINE\_LOCATION}), time > 8 \wedge time < 16, \text{Log\_Access}()))$

This scenario will lead to *conflicting constraints* (i.e. conditions in our P-RBAC model) in our dependency group since it is not possible to enforce both of these permissions together. This becomes apparent when we combine the permissions by taking the conjunction of the conditions specified:

$Permission_1 \wedge Permission_2$: $(Group_1, ((\text{use, ACCESS\_FINE\_LOCATION}), time > 16 \wedge time < 20 \wedge time > 8 \wedge time < 16, \text{Log\_Access}()))$

We have conditions conflicting due to conflicting constraints between both privacy rules. Let's imagine another scenario where Alice defines the following rules on location:

$Permission_1$: $(Group_1, ((\text{use, ACCESS\_FINE\_LOCATION}), time > 10 \wedge time < 20, \text{Log\_Access}()))$

$Permission_2$: ($Group_1$, ((use, ACCESS_FINE_LOCATION), $time > 8 \wedge time < 16$, Log_Access()))

This could be seen at first as conflicting constraints, but it is not, as they can coexist by reducing both to one unique with the strongest constraint of the two rules. It is what we will call *concurrent constraints*. The resulting rule would be as follows:

$Permission_1 \wedge Permission_2$: ($Group_1$, ((use, ACCESS_FINE_LOCATION), $time > 10 \wedge time < 16$, Log_Access()))

A last scenario, would be incomparable constraints. Let's state that Alice defines the following rules:

$Permission_1$: ($Group_1$, ((use, ACCESS_FINE_LOCATION), $country =$ "*UnitedStates*" $\wedge$ $time > 10 \wedge time < 20$, Log_Access()))

$Permission_2$: ($Group_1$, ((use, ACCESS_FINE_LOCATION), $country =$ "*Belgium*" $\wedge time > 8 \wedge time < 16$, Log_Access()))

The second members of both constraints are concurrent constraints but the first member makes them apply to different countries. It means that the first is only valuable in USA and the other one in Belgium, constraints are thus not conflicting, they are just incomparable.

4.7.2 Detecting conflicts

Constraints conflicts could be seen the same problem as concept equivalence detecting and concept compatibility checking in ontologies merging [46]. The same pattern is used in *P-RBAC* [33] as the one used in ontologies merging [46] to detect constraints conflicts in permissions. The pattern is based on the satisfiability problem:

"Satisfiability is the problem of determining if the variables of a given Boolean formula can be assigned in such a way as to make the formula evaluate to true or determining whether no such assignments exist." [4]

We consider that each statement in the constraints that could be defined by Alice is a boolean, which means that there are only two possible outputs: `true` or `false`. This allows us to treat the detection of conflicting constraints as a satisfiability problem.

Satisfiability problem is used in *P-RBAC* to detect conflicting conditions. Algorithm 4.7.2 is the algorithm used in *P-RBAC*, that we can use in our simplified model. The *isConflict()* function in the algorithm is checking if values are conflicting for a particular condition.

Algorithm 1: Algorithm 1 - P-RBAC Conflict Detection ([33])

```
Input: var1,var2: two set of conditions applied in two permission assignments
Output: True if conditions are conflicting, False otherwise
orderedVar1 ← Order conditions of var1 by name;
orderedVar2 ← Order conditions of var2 by name;
$i \leftarrow 0$;
$j \leftarrow 0$;
result ← false;
while $i < var1.size$ do
    while $j < var2.size$ do
        if $orderedVar1[i].name = orderedVar2[j].name$ then
            if isConflict($orderedVar1[i].value, orderedVar2[j].value$) then
                result ← true;
                return result;
            $i++$;
            $j++$;
        if $orderedVar1[i].name < orderedVar2[j].name$ then
            $i++$;
return result;
```

If we apply this algorithm to Alice's scenario, the result would be `true` as she defined two conflicting constraints inside the same dependency group.

5 RIGHTS EXPRESSION LANGUAGES AND ANDROID

This Section focuses on the second contribution of the thesis by introducting rights expression languages into mobile applications, especially here ODRL [22] language. The first need is to explain how they can help in such a context. Once this done, we will have to decompose ODRL to see wich contents are interesting in the scope of this work. We will then see how ODRL can be useful to mobile applications by using our previous scenario with Alice. This will allow us to express common scenarios of mobile applications using a REL but also transform our modified P-RBAC model into a rights expression language.

5.1 Contributions of rights expression language

Before introducing ODRL core model, and expressing our P-RBAC model into such a rights expression language, we have to raise the question of the utility of such a tool in mobile applications. Indeed, rights expression languages are often quite complex and our goal is not to complexify the environment we are working on, but more looking at how we can improve current tools and tools to come using a standard followed by his community and his capacity of expression.
We identify here three major contributions of rights expression languages, more can probably be stated. We also express the counterparts of using such a tool, by using some references which focus on the limits of rights expression languages.

5.1.1 Interoperability

Interoperability is a key feature for rights expression languages. Most of the tools of Android system (like *Apex* [32] or others [8]) are using their own XML-based reprensentation of policies, which interoperability speaking, is quite closed. Such tools can thus not take into account policies defined by another tool or a translation phase would be needed, and thus an overhead for the mobile device. Plus, with different languages used by different tools, a centralized policy database on a device is not possible.
A standard expression language would facilitate such a thing but as we will see, even in the world of RELs, interoperability is not always taken for granted. Indeed, we restate from [37]:

“Interoperability is a key for the real deployment of Digital Rights Management (DRM) systems. A clear example is at the level of Rights Expression Languages (RELs), where two of them are competing to have a place in the market. On the one hand, MPEG-21 REL is an ISO/IEC official standard, and on the other hand ODRL (Open Digital Rights Language) is a public specification that is being used by the Open Mobile Alliance (OMA), a relevant industrial forum in the area of mobile systems.”

Although authors are focusing on digital rights management system, the same things applies with privacy management systems. The "competition" between RELs is quite a challenge for the interoperabilty between systems where different RELs are in use. But, studies on the subject profuse (see by example: [37],[30],[28]) and the interoperability between two RELs, which are defined, studied, and are evolving version by version with a community of users, has way less problems than two XML-based languages which are defined for one framework only and restrained to the functions of this framework. Plus, RELs have often several core components which are studied to be efficient for the domain where they apply, whatever it is permissions, privacy or yet digital rights.
The acceptance of one unique standard REL for a particular domain would narrow down the interoperabilty problems to nearly zero. ODRL [22] is seen by the Open Mobile Alliance as the standard for mobile environment but it is without counting the proprietary languages, software and devices which do prefer having their own language, and this way forcing users to use a predefined set of their own tools.

5.1.2 Privacy Change

The second key feature of rights expression languages is the evolution aspect. Indeed, applications and systems are being updated from time to time and new features can be added, removed, modified,.... This evolution does not grow alone without impact on privacy. Each time an application or a system evolves, it can have an impact on its own privacy level or even privacy of other systems or applications. Without a rights expression language, the evolution of a system in such a way would lead to either a privacy issue, either a useless set of constraints in the application's privacy policy. XML-based languages like the one defined in *Apex* are not studied to support privacy evolution or rapid adaptation to system changes. ODRL and other rights expression languages, on the other side, are studied to be complete or nearly-complete and thus, each scenario is taken into account which allows adaptation of policies to evolution of applications to be faster and easier.

5.1.3 Community

A last key of rights expression languages and mainly ODRL is the community of users beyond it. Indeed, ODRL beneficiates of a wide community of users discussing of future versions of the language and application of this language to new domains (like privacy or mobile applications). Such a community allows the language to evolve and follows new domain of applications while also allowing to put weaknesses frontward. Community wiki and working groups can be found in [5].

5.2 Open Digital Right Language (ODRL)

We have already introduced rights expression languages in our background Section (2.3), but we will now focus on ODRL [22] rights expression languages and see how it can fit in

[5]`http://odrl.net/wiki/tiki-index.php`

our foundation of a model for mobile applications permissions. As a preambule, we need to explain why we have chosen ODRL over other rights expression languages discussed in Section 2.3. We will then make some correspondance between ODRL and Android's permission system to be able to compare concepts of both models and explain how policies are managed in such a system. It will lead us to the exploration of ODRL, into three parts. The first one is the the ODRL version 2.0 core model specification. Then, we will review the common vocabulary which provides basic vocabulary and its semantic for the policy expression language. While reviewing those components, we will basically build a meta-policy XML file for our model. We will only focus here on parts of the language interesting in terms of permissions, privacy and expression of such.

The semantic of the vocabulary for all the specifications is based on a request for comments (RFC)[6] which precognizes, to avoid ambiguities, the usage of key words for such a specification. Those key words are: “MUST”,“MUST NOT”, “REQUIRED”, “SHALL”, “SHALL NOT”, “SHOULD”, “SHOULD NOT”, “RECOMMENDED”, “MAY”. Plus UML Notation for the core model.

5.2.1 Why ODRL?

We have chosen ODRL for three reasons:

- The first one is due to the mobile environment we are working on, indeed ODRL has been officially accepted by the Open Mobile Alliance (OMA) as the standard rights expression language for all mobile content.
- The second reason is the open aspect which has lead to a community of developers sharing their ideas online, and working on new drafts of the language accessible to anyone. It allows to follow ideas, ask questions, discuss scenarios and so on.
- The third reason is the large vocabulary proposed by ODRL, which allows to adapt this language to a new environement like mobile phones.

We could have chosen XACML [2] or XrML [10] for other reasons, but a choice has to be made and the major goal was to study one particular REL to see how it can be adapted to fulfill privacy requirements of mobile applications in a concrete scenario. XACML would also be interesting as a case study in a future work because of the access control aspect and the business-oriented idea beyond it.

[6]`ftp://ftp.rfc-editor.org/in-notes/rfc2119.txt`

5.2.2 Correspondance between Android and ODRL

If we look back at our Figure 1, restating the major idea beyond Android's permission system, we have 4 concepts: *Permissions* which are *Operations* over *Resources*, and *Applications*. By example, a permission would be *Internet Access* and this permission would be assigned to an application. We have discussed the need for the add of constraints (and groups, but we focus on policies here). In this basic system, no constraints can be added on the permissions. One contribution of this work is to allow users to add constraints over permissions granted to applications. So we need to add a brick to this system, and this brick, which is allowing declaration of constraints and more advanded policies, is brought by ODRL.
When we look at major concepts of ODRL, we have: *Permissions/Prohibitions*, *Constraints*, *Asset*,.... We can thus use this expressiveness to complete the permission system of Android applications and allow the add of constraints or the expression of more advanced policies.

We are talking a lot about policies, we should develop more how they are managed and by who.

Policy Management Policies of the basic system are created when an application is installed. Android waits for the confirmation of installation by user, which is by the same way granting all permissions asked by the application he wish to install. A policy file is held by Android and each time this particular application is launched, the information from *"AndroidManifest.xml"* of this application is checked to verify compliance between permissions granted and permissions asked.
When a user removes the application, all information about granted permissions to this application is removed.

With a tool like *Apex* (or our prototype, see Section 6.7), policies are more complex, allowing constraints and more expressiveness, mainly if we use a rights expression language. Thus those policies are stored in a different database than the basic file of the system. If we use *Apex*, those policies are created at the installation where user can add constraints over permissions or deny permissions with a modified installer furnished with *Apex*. After that, policies stay the same for the whole application's life and the only way to modify it, is to re-install the application.
With a tool like our prototype, those complex policies are not created at the installation. Indeed, it is the choice of user to bother with the add of constraints on permissions granted to applications (groups of applications in our case) or not. If he wants to, then he can use the policy manager, to add those constraints or to modify permissions or remove them. The management of policy is thus user-centric in this kind of framework, and we plan to use ODRL in this idea, where policies created by users will be generated in ODRL to allow high expressiveness.

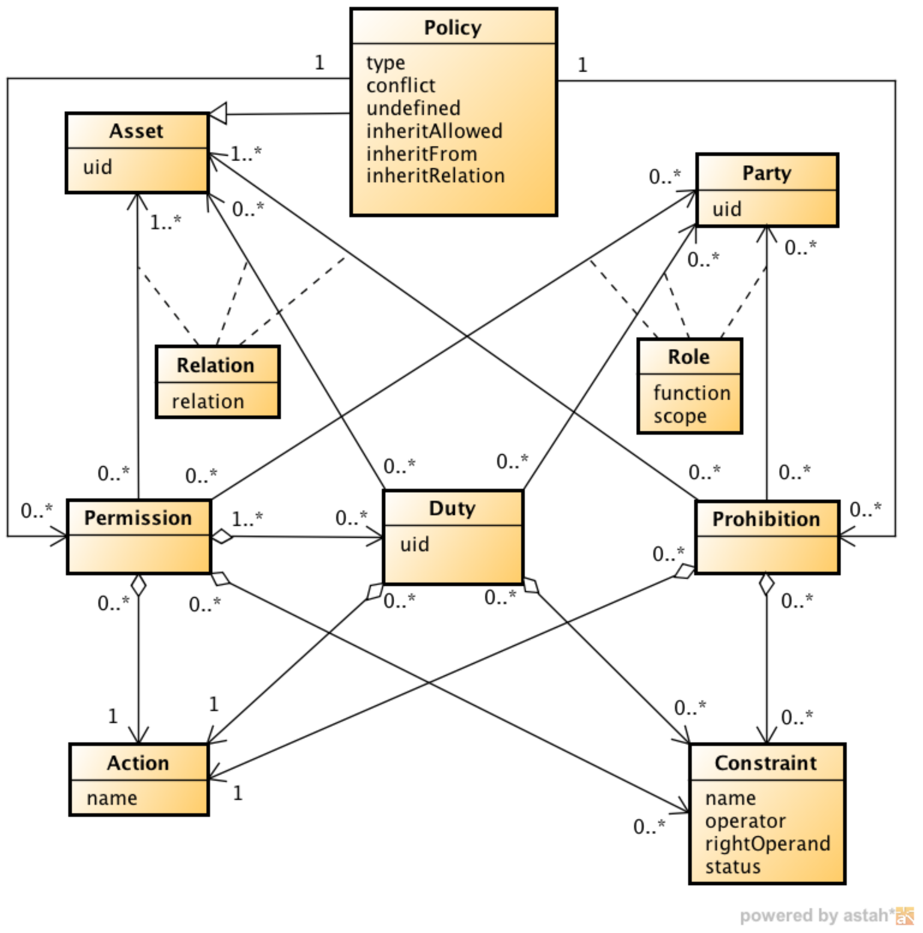


Figure 13: ODRL Core model ([22])

5.2.3 ODRL Core Model 2.0

"The ODRL Core Model is designed to be independent from implementation mechanisms and is focused on the optimal model and semantics to represent policy-based information." [22]

Figure 13 that we restate from [22] is the UML representation of the core model 2.0 of ODRL. We will describe it broadly here while staying oriented to mobile systems.

This diagram shows a **Policy** entity which will be the root of every XML-based ODRL policy file. A policy is inherited from an **Asset** which has a particular unique ID which will thus also be the ID referencing all policies of a particular asset. Each policy has a type (see common vocabulary in Section 5.2.4) which caracterize the domain where the policy is applied which is thus required. Other fields are optionnal and will not be discussed here. A policy contains zero or more **Permissions** and zero or more **Prohibitions**. Each one

of those entities being associated with at least one asset. Indeed, a prohibition by example can only prohibit a particular operation on a particular asset.
The **Duty** entity indicates a requirement that should be fulfilled in return for being entitled to the referring permission entity. In a way, a duty is similar to a permission in that it is an action that can be undertaken. If a permission refers to several duty entities, all of them have to be fulfilled for the permission to become valid. If several permission entities refer to one duty, then the duty only has to be fulfilled once for all the permission entities to become valid. Each duty as its own unique ID which is used to refer a duty to multiple permissions.
An **Action** entity is a composition of permissions and prohibitions and a set of duties. Each action has a name which will also be explained in the common vocabulary Section (see Section 5.2.4). One ore more **Contraints** entities can be set over a policy restraining one or more permissions or one or more prohibition with possible duties to be fullfilled when constraint are not or not respected. Each constraint has 4 attributes: a name, an operator, a right operand and a status. Those will be also explained in Section 5.2.4.
Of course, permissions, prohibitions and duties are all associated with a **Party** entity which is identifying a person, group of people, organization or in our case a particular application. The party MUST identify a (legal) entity that can participate in policy transactions. Each parrty has also a unique identifier, which will be `UID` of applications in Android system.
We have two association class left: **Relation** entity which can be used to link to an asset from either a permission, a duty or a prohibition, indicating how the asset should be utilised in respect to the entity that links to it and a **Role** entity which can bes used to link to a party from either permission, duty or prohibition, indicating which role the party takes with respect to the entity that links to it.

After this short presentation of the ODRL core model, we can outline a first root in our XML file of ODRL policies for mobile applications [7]:

```
<o:policy xmlns:o="http://odrl.net/2.0" uid="POLICY ID">
   <o:permission />
    <o:prohibition />
</o:policy>
```

5.2.4 ODRL Common Vocabulary

The common vocabulary specifies the terms used by the core model (see Section 5.2.3) for policy expression needs. This vocabulary-based model try to fulfill requirements of current state-of-the-art of domains where rights expression languages apply (or could apply).
Those domains are quite varied: access controls management, privacy management, social networking management, publishment, digital rights management and so on. The major

[7] Full XSD Schema is defined here: `http://odrl.net/2.0/schema.xsd`

idea behind the vocabulary model is to define a wide range of terms for expressing types, permissions, prohibitions, constraints, and duties over assets. Each term has this way an identifier. This one is a unique method to refer to the semantics in specific encoding schemes (eg XML, RDF etc) [22].

Policy Types The first set of terms defined by the model is the *type* of policies which allows to caracterize the domain where the policy apply. We will only restate here terms that matches permissions and privacy. We restate those from [22], current draft of common vocabulary.

agreement	Policy expressions that are formal contracts (or licenses) stipulating all the terms of usage and all the parties involved.	Must contain at least the Party entity with Assigner role and a Party with Assignee role. The latter being granted the terms of the Agreement from the former.
offer	Policy expressions that propose terms of usage from an Asset owner.	Must contain a Party entity with Assigner role. The Offer may contain a Party entity with Assignee role, but does not grant any privileges to that Party.
privacy	Policy expressions that stipulate the terms of usage over personal information.	Must contain at least the Party entity with Assigner role and a Party with Assignee role. Must also contain a duty on the assignee related to obligations towards managing the assigner's Asset containing personal information. The Assignee is being granted the terms of the Privacy policy from the Assigner.

Table 1: ODRL Common Vocabulary: Policy Types

We have thus three main interesting types: agreement, offer and privacy.

In our mobile applications domain, the notion of Offer does not seems adequate as the Assignee role is not predominant and all the offer is focused on the Assigner offering terms of usage for a particular resource. In our context of application, the role of Assigner is not important as it generally refers to the user or the system itself (again, for the user) and as user is generally unique. On the other side, Assignees are played by applications (groups

of applications in our modified model), and this role is thus particularly important.
Remains two notions: Agreement and Privacy. Agreement, that is, a term expressing a contract between an Assignee and one Assigner, which grant terms to the first one. Privacy is a term focusing on privacy policies defined by the Assigner for an Assignee, but declares a Duty which is mandatory and is expressing obligations of the Assignee due to the privacy policy.
Those notions are thus quite close but as this is explicit in the definition, we will choose the term of *Privacy* for the type of our ODRL policies. Which, by referring to the core model (see Section 5.2.3) and the present notion of *Privacy* type gives us our XML basic structure of privacy policy for mobile applications:

```
<o:policy xmlns:o="http://odrl.net/2.0" type="o:privacy"
   uid="POLICY ID">
   <o:permission>
  <o:asset id="urn:resource" />
  <o:role function="o:assigner" />
  <o:role function="o:assignee" />
  <o:duty />
    </o:permission>
    <o:prohibition>
  <o:asset id="urn:resource" />
  <o:role function="o:assigner" />
  <o:role function="o:assignee" />
  <o:duty />
    </o:prohibition>
</o:policy>
```

This basic structure contains one permission and one prohibition (of course, it could contains only permissions or only prohibitions or more of both). Each one of them is ruling a particular *asset* which has an `ID`, identifying the asset which is constrained. This asset is identified by a `URN/URI` (Uniform Resource Name/Identifier). Permissions and prohibitions both have two roles: an assignee and an assigner (even if in our case the assigner is not a useful information, as this is system or user). A possible *duty* is also allowed, to enforce a particular action whenever the constraints are respected by the assignee to access the asset.

Actions Actions, that is, the particular operation concerned by the privacy rule. In our model, it can be seen as the set of operations over the resources.
ODRL defines more than 50 terms for actions and thus, sometimes, it is difficult to choose the most appropriate term or to understand the main meaning of a particular term which can lead to ambiguities.

append	The act of appending to the asset	For example, the ability to add record to a database (the asset).
copy	reproduce	The act of making an exact reproduction of the asset.
delete	The act of permanently removing the asset	When used as a Duty means all copies removed.
display present	The act of making a transient visible rendering of the asset	For example: displaying an image on a screen..
execute present	The act of executing the asset	For example: machine executable code or Java such as a game or application.
inform	The act of informing a party of uses of the asset	Typically used as a Duty.
install	The act of loading the asset onto storage device ready for operation	
obtainConsent	The act of requiring explicit consent from a party to perform the action on the asset	Typically used as a Duty for the asset owners to decide on a case-by-case basis.
pay	The act of paying a financial amount to a party for use of the asset	Must link to an Asset which represents the amount of the payment. The payer is the Assignee and the Payee is the Assigner of the policy.
play present	The act of rendering the asset into audio and/or video form	For example; playing a movie file.
preview	The act of providing a short preview of the asset	For example; the first 5 minutes of a movie.
print	The act of rendering the asset onto paper or hard copy form	For example: creating a permanent, fixed (static), and directly perceivable representation of the asset.
read	The act of reading the asset.	For example, the ability to read a record from a database (the asset).
uninstall	The act of unloading the asset from storage device	The asset is not longer accessible.
write	The act of writing to the asset	For example, the ability to write a record to a database (the asset).

Table 2: ODRL Common Vocabulary: Actions

We will here consider only main actions and we will see that our model requires only one of them for Android applications, but the heterogeneity of actions in ODRL allows to add more granularity in another system where the permissions system or the privacy management is defined in terms of different operations applicable to different resources and not only the fact of accessing resources which is nearly always the case with android permissions system.

The table 2 is just a subset of all available defined actions but this subset is sufficient for mobile environments, and even maybe too complex. This *action* entity allows to define precise granular policy rules for a particular resource.
By example, *preview* can be used to allow a particular application to load a preview of 10 seconds of a particular video (which is the resource here). Or yet, *textToSpeech* can be used to allow a particular application to render a particular text into speech. Beside those and some other actions, there is a lot of casual actions: write, read, uninstall, install, sell, pay, play-present, display-present,...

Android applications and our modified model especially are defined to be rules like: "this application can **access** this resource". Those rules do not specify any kind of operations but in another system or in a future version or tool for Android applications, this could be the case. Thus, those fine-grained actions could be useful to declare precise policy rules for actions over resource.

So in Android, the only pertinent action is *execute-present*, because each asset is executed by the system when an application ask for it and has the right to ask it. We can update our current XML-based representation:

```
<o:policy xmlns:o="http://odrl.net/2.0" type="o:privacy"
   uid="POLICY ID">
    <o:permission>
         <o:asset uid="urn:resource" />
         <o:action name="o:present"/>
       <o:role function="o:assigner" />
  <o:role function="o:assignee" />
  <o:duty />
    </o:permission>
    <o:prohibition>
         <o:asset id="urn:resource" />
         <o:action name="o:present"/>
       <o:role function="o:assigner" />
  <o:role function="o:assignee" />
  <o:duty />
    </o:prohibition>
</o:policy>
```

Constraints We have now to define constraints over permissions and prohibitions to "present" one particular asset. This entity has 3 particular attributes: a name, a right operand and an operator. Right operand is constitued of standard simple data types like String, Boolean,...while name refer to the variable being constrained by the operator and the right operand, same way of "where" clause in SQL DDL code.

Most useful names are:

count	The numeric count indicating the number of times the corresponding entity may be exercised	Should be a positive integer.
dateTime	The date (and optional time and timezone) representing a point in time or period	Date and Time value must conform to [ISO-8601] as represented in the XSD schema.
elapsedTime	A period of time in which the policy action can be exercised	The start of the period is when the action is first exercised.
purpose	Specification of a defined purpose applicable to the asset usage	For example, educational use.
spatial	A code representing a geospatial area	The code value and code source must be represented. For example, the ISO3166 Country Codes and the Getty Thesaurus of Geographic Names.
timeInterval	Recurring period of time in which the usage may be exercised	Interval value must conform to ISO8601 as represented in the XSD schema.
system device	An identifiable computing system	For example, identifiable via the CPU or unique hardware address.
virtualLocation	Specification of a digital locale	For example, an Internet domain or IP address range.

Table 3: ODRL Common Vocabulary: Constraints

Most part of those constraints will be added to our prototype, but some constraints are impossible to check due to performances concerns or possibilities of the device. By example, a timer for *"elapsedTime"* would be too lousy in terms of battery usage and is thus not adequate for mobile environment (at least, nowadays).

Those constraints variables are constrained by different type of operators which are: eq, gt, gteq, lt, lteq, neq,...and some more not interesting in our privacy management for mobile applications goal. Those are respectively meaning: equals, greater than, greater than equals, less than, less than equals, not equals.
If we update our XML-based representation of privacy policy now, we have:

```

<o:policy xmlns:o="http://odrl.net/2.0" type="o:privacy"
    uid="POLICY ID">
    <o:permission>
          <o:asset uid="urn:resource" />
          <o:action name="o:present"/>
        <o:role function="o:assigner" />
   <o:role function="o:assignee" />
   <o:constraint name="" rightOperand="" operator="" />
   <o:duty />
     </o:permission>
     <o:prohibition>
          <o:asset id="urn:resource" />
          <o:action name="o:present"/>
        <o:role function="o:assigner" />
   <o:role function="o:assignee" />
   <o:constraint name="" rightOperand="" operator="" />
   <o:duty />
     </o:prohibition>
</o:policy>
```

Party and Role The role entity which is representing targets of the policy rules has two attributes: a function and a scope.
The below table shows terms that may be used as the attribute "function" of the entity. We keep only the two most relevant terms for our domain of application:

In Android, the assigner function will always be the system entity (even by means of user authorizing system to grant permission to an application). But we will let the concept

assigner	The Party is the issuer of the policy statement
assignee	The Party is the recipient of the policy statement

Table 4: ODRL Common Vocabulary: Party

of assigner for other mobile systems.

The scope attribute indicates how to interpret the identified party. For example, if a party is identified as a member of one group of application and is the assignee, it will be represented with “group” scope. Vocabulary terms in the below table may be used as the attribute “scope” of the role entity, or at least the only two interesting for mobile environments. Indeed, either it is an application which is the assignee, either it is a group of applications, nothing else.

individual	The Party is a single individual.
group	The Party represents a defined group with multiple individual members.

Table 5: ODRL Common Vocabulary: Role

Asset and Relation For the asset entity, the relation attribute can be used to specify the relationship of the asset and the policy. But in our case, the only accepted relation is the default one:
target: The asset is the primary object of the policy.

The final XML-based representation of a skeleton of ODRL policy thus looks like:

```
<o:policy xmlns:o="http://odrl.net/2.0" type="o:privacy"
   uid="POLICY ID">
    <o:permission>
        <o:asset uid="urn:resource"/>
        <o:action name="o:present"/>
        <o:constraint name="" operator="" rightOperand=""/>
        <o:role uid="APPLICATION NAME" function="o:assignee
           "/>
        <o:duty />
    </o:permission>
    <o:prohibition>
```

```
        <o:asset uid="urn:resource"/>
        <o:action name="o:present"/>
        <o:constraint name="" operator="" rightOperand=""/>
        <o:role uid="APPLICATION NAME" function="o:assignee
           "/>
        <o:duty />
    </o:prohibition>
</o:policy>
```

The duty entity can also be subdivided but for generic purpose, is not mandatory. It will only be when a particular action with zero or more constraints is needed when a particular permission/prohibition is associated with an asset.

5.2.5 Expressing Android Scenarios Using ODRL

We have decomposed ODRL through a review of the core model and the common vocabulary and have tried to show the XML base representation of our policies for mobile applications. We will now have to apply this to some scenarios, to have a better look at how it can be used in a particular context. We also have then, to meet our dependency-aware model with this ODRL decomposition we made, to finally be able to express privacy policies over groups of applications.

Expressing Alice's scenario using ODRL We will, in this subsection, review the first scenario of Alice and we will see how we can express it using our meta-policy file from last Section.
Let's first remind the constraints added by Alice on the applications she installed. The complete scenario can be found in Section 4.3.

- `FBContacts`: can only access location data if time is between 4pm and 8pm.
- `GPSFriends`: can only access location data if location is in the UK and the time is between 8am and 4pm.
- `GPSWifi`: can only access location data if the location is in London.

Using our XML-based ODRL policy structure, we would thus have three different privacy rules. One over `FBContacts` with a constraint on location restraining geolocation between 4pm and 8pm. One over `GPSFriends` with two contraints on location: restraining geolocation between 8am and 4pm and location inside UK. And one last rule over location for `GPSWifi` restraining geolocation to the area of London.
We have thus to declare three policies, one for each application, which is also an overhead comparing to our groups of application, on which we will apply those policies later on.

`FBContacts` would be ruled by:

```
<o:policy xmlns:o="http://odrl.net/2.0" type="o:privacy"
   uid="POLICY ID">
    <o:permission>
        <o:asset uid="ACCESS_FINE_LOCATION"/>
        <o:action name="o:present"/>
        <o:constraint name="o:dateTime" operator="o:lteq"
           rightOperand="20:00"/>
        <o:constraint name="o:dateTime" operator="o:gteq"
           rightOperand="16:00"/>
        <o:role uid="com.android.FBContacts" function="o:
           assignee"/>
        <o:duty />
    </o:permission>
</o:policy>
```

`GPSFriends` would be ruled by:

```
<o:policy xmlns:o="http://odrl.net/2.0" type="o:privacy"
   uid="POLICY ID">
    <o:permission>
        <o:asset uid="ACCESS_FINE_LOCATION"/>
        <o:action name="o:present"/>
        <o:constraint name="o:dateTime" operator="o:lteq"
           rightOperand="16:00"/>
        <o:constraint name="o:dateTime" operator="o:gteq"
           rightOperand="8:00"/>
  <o:constraint name="o:spatial" operator="o:eq"
     rightOperand="United Kingdom">
        <o:role uid="com.android.GPSFriends" function="o:
           assignee"/>
        <o:duty />
    </o:permission>
</o:policy>
```

Finally, `GPSWiFi` would be ruled by:

```
<o:policy xmlns:o="http://odrl.net/2.0" type="o:privacy"
   uid="POLICY ID">
    <o:permission>
        <o:asset uid="ACCESS_FINE_LOCATION"/>
        <o:action name="o:present"/>
  <o:constraint name="o:spatial" operator="o:eq"
     rightOperand="London">
        <o:role uid="com.android.GPSWifi" function="o:
           assignee"/>
        <o:duty />
    </o:permission>
</o:policy>
```

Those scenarios are representing permissions for applications to access location, with some constraints and eventually some duty. We could also represent prohibitions as we want a user being able to deny a permission to an application. Let's say Alice refuses to let `FBContacts` access her location, under any constraint. It would be represented by a prohibition rule:

```
<o:policy xmlns:o="http://odrl.net/2.0" type="o:privacy"
   uid="POLICY ID">
   <o:prohibition>
        <o:asset uid="ACCESS_FINE_LOCATION"/>
        <o:action name="o:present"/>
        <o:constraint />
        <o:role uid="com.android.FBContacts" function="o:
           assignee"/>
        <o:duty />
    </o:prohibition>
</o:policy>
```

Expressing our dependency-aware model using ODRL *Now that we have presented our dependency-aware model based on P-RBAC and have expressed Alice's mobile applications scenario with an ODRL expression language, we have to take benefits from both sections and try to express our model using ODRL and this way having policy rules* *over groups of applications and no more over applications one by one.*

As said before, another advantage of groups-based policies is that there is only one policy file for each application of one group and not one file for each application separatly. Alice's scenario contains conflicts, so all constraints cannot be grouped without modifying those. We have discussed this in Section 4.7. Our conflicts detection algorithm can detect all of those conflicts but cannot fix all of them. So, it would simply prevent the user that the last constraint he tried to add was refused because of conflicts.
We will thus reason here about somewhat different constraints, to avoid falling back into the same discussion:
Let's thus state that Alice has choosen three constraints to add on the group she just created: *GROUP_INTERNET* containing: `GPSWifi`, `GPSFriends`, `FBContacts`:

- On location: those application cannot access location between 8 a.m and 8 p.m
- On internet: those applications cannot access Internet outside United Kingdom
- On phone calls: those applications cannot access phone calls

```
<o:policy xmlns:o="http://odrl.net/2.0" type="o:privacy"
   uid="POLICY ID">
    <o:permission>
        <o:asset uid="INTERNET"/>
        <o:action name="o:present"/>
  <o:constraint name="o:spatial" operator="o:eq"
     rightOperand="United Kingdom">
        <o:role uid="GROUP_INTERNET" function="o:assignee"
           />
        <o:duty />
    </ o:permission>
    <o:permission>
        <o:asset uid="ACCESS_FINE_LOCATION"/>
        <o:action name="o:present"/>
  <o:constraint name="o:dateTime" operator="o:lteq"
     rightOperand="20:00">
  <o:constraint name="o:dateTime" operator="o:gteq"
     rightOperand="08:00">
        <o:role uid="GROUP_INTERNET" function="o:assignee"
           />
        <o:duty />
    </ o:permission>
     <o:prohibition>
```

```
        <o:asset uid="PHONE_STATE"/>
        <o:action name="o:present"/>
        <o:constraint />
        <o:role uid="GROUP_INTERNET" function="o:assignee"
           />
        <o:duty />
    </o:prohibition>
</o:policy>
```

Each new application on the group would then be under the same rules. We do not have to add a rule for a new application each time one is added. Of course, only if the rule we want to add is already existing for one of the application, otherwise, we have to add it to group's policy rules, which will implies that each application of the group will also be constrained by this new rule.

5.2.6 Limits of ODRL

We have highlighted advantages of applying ODRL to mobile applications and how to apply it to such scenarios. But there are also limitations on the usage of rights expression languages which are common to all systems. It is why we find it important to mention and discuss them in this subsection.

The central problem mentionned in the litterature (in [38] by example) is the complexity of current RELs like ODRL. Plus, those languages do not offer any abstraction mechanism to deal with this complexity. This growing complexity implies a difficulty of reuse. In particular, because the extensions in current RELs are content-based, rather than functionality-based, reuse is difficult. It means that each extensions added to core model or common vocabulary are based on content of a particular domain of application rather than a functionality adaptable to several ones. In [38], authors are justifying this by stating:

"This is because in these languages DRM functions such as trust management, authentication, encryption, or negotiation services are not completely separable from the language, as we believe they should be". The entities provided by ODRL, i.e., the core schema and the common vocabulary, are too coarse and cut across too many areas of functionality. *"Reuse becomes more natural if it is based on smaller units of functionality such as authentication or encryption, rather than the end-to-end functionality described in a large use case".* For instance, not all mobile systems will requires the same functionalities in terms of privacy protection in order to ensure it. This makes it difficult to design a single content-based extension that can satisfy the needs of all systems in this market. On the other hand, this specificity allows an easier integration to potentially different mobile systems into their existing architecture.

This content-oriented aspect in place of a functional-based aspect is due to a design by use cases which are useful to represent functional requirements of a system or a domain of application but they are not able to deal with non-functional requirements such as reusability, portability and interchangeability which are thus not of central concerns in evolutions of rights expression languages like ODRL.

6 VALIDATION

We have divided this work through two axes, in the first one we were studying privacy issues of mobile applications and we have established a model of the access control policies of Android applications based on P-RBAC, while taking dependencies between those same applications into account. In the second one, we have introduced benefits of rights expression languages for expressing privacy in mobile applications and we have tried to express our previous model into ODRL language. The point of the next Section is thus to append both axes into a prototype where the goal is not to have a complete ready-to-be-published application but more to have an insight about how we can integrate those axes into a real system and what they are bringing in terms of functionalities. The system chosen, as the previous case study, is Android system. Our prototype is named PrimAndroid for Privacy Management for Android Applications.

6.1 Prototype Functional Description

The prototype needs several functionalities to fulfill our idea of privacy management for Android applications.

User must be able to:

- Create/delete groups of applications with a particular name consistent for him
- Add/remove installed applications to/from those groups
- Add/Modify/remove constraints over the permissions granted to all applications of one particular group. Constraints are ruling resources such as Internet, geolocation, contacts, storage, messages,...

System must be able to:

- Record constraints added by user under ODRL format
- Add/remove dependent applications to/from the group where a particular application is added/removed, if any
- Check constraints added on groups when an application is running and asks access for a particular resource
  - Deny permission if constraints are violated
  - Warn user about the deny
  - Check possible conflicts between constraints

Note: Applications which are refused the access to a particular permission, throws a `SecurityException` which is sometimes handled by those, sometimes not. It is the role of developers to make sure such things are managed, in a principle of robustness.

6.1.1 Future functional requirement

We will limit ourselves to the implementation of the given needs but one more functionality could be interesting to look at, in a future version: **Policy sharing**. Indeed, we could allow users to exchange policies made over groups of applications. It would allow some users to make predefined groups of casual applications, with coressponding policies and other users would be able to apply this policy directly if they have the same usage habits with the applications in those groups.

6.1.2 Use cases

We restate here the functional description by the meaning of use cases. See Figure 14

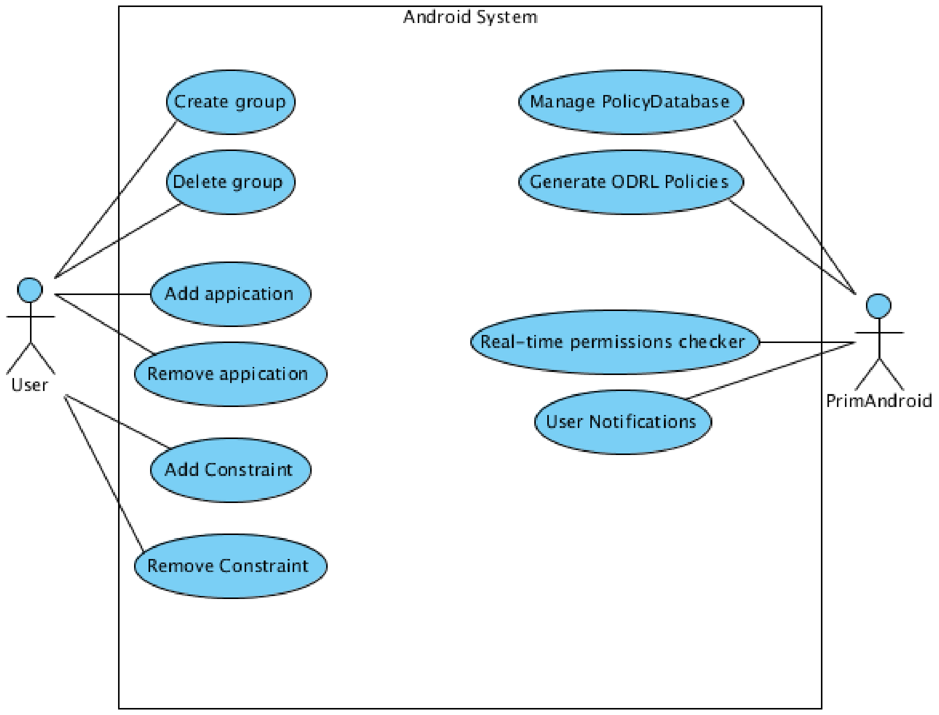


Figure 14: Use Case Diagram of PrimAndroid

6.2 Prototype Non-Functional Description

Beside functional requirements, we have to look at some non-functional ones, as a mobile phone is subject to constraints of security, performance and robustness. And the need for those constraints to be fulfilled is growing with evolution of what we can do nowadays with such a system. Our prototype does need to avoid adding weaknesses to the base system in those terms.

6.2.1 Robustness

As mentionned earlier, see Section 6.1, each time an application is refused the access to a particular resource, system throws a `SecurityException`. Most of the time, applications are catching this exception and are redirecting the flow of events according to where the exception occured. But sometimes, developers are not preventive enough and do not catch this kind of exception when getting access to a resource of the phone. This can result in an application failure and a warning displayed for the user, which can be annoying. Our prototype is working the same way, applications which are refused the access to a resource, get a `SecurityException`, but a notification is send to the user saying that the permission was refused, in case of an application without management of such exception. It is the role of the developer to ensure that applications written for Android system are taking every scenario into account, especially when accessing resources.

6.2.2 Security

Security speaking, we will just mention the fact that we have kept the idea of the base system. Each application can get information about permissions granted to other applications, but cannot modify or write such permissions. In our prototype, the policy database can only be accessed by *Activities*, and such activities can just access reading methods, not writing ones, exception made of the PrimAndroid application.

6.2.3 Performance

In a performance point-of-view, we would like to mention the policy database of our PrimAndroid applications. Indeed, each group is associated with a file containing all policy rules of this same group. There is thus one file for one group, for complexity, and thus performances reasons. Indeed, if all policy rules where in a unique file, checks for one group would imply to parse a file containing all policies of all existing groups. Instead, our system allows to parse only rules of the concerned group. As performance is an important asset in battery-embedded devices, it can be a gain of processing. Plus, if the first check of Android basic permissions fails, we do not check our policy databse, and deny the access immediately. It avoids to unnecessarily read our policy database.

6.3 Implementation Requirements

The changes needed for an implementation of our prototype on Android system implied the need of modifying classes in lower level than the application layer. We thus needed to make our own read-only memory version (ROM), from scratch from the android sources which are free and accessible to anyone.

Here are some characteristics for the development:

- Android sources version: Android Froyo 2.2
- JDK version: 1.5 (mandatory for compliance with Android SDK)
- Development IDE: Eclipse EE + Android SDK
- Main languages used: JAVA,XML

6.4 Global Architecture

We have catched the global requirements of our system in terms of functionalities, we need to describe the architecture and its impact on Android's base architecture. In this goal, we will first give a global view of how we integrated our framework into Android system. Then, we will validate our prototype by confronting it to our model. Screenshots can be found in appendix, see Appendix C.

6.4.1 System Architecture: Android Integration

The Figure 15 is broadly summarizing the modifications made to Android architecture to include our framework. Each Android class used is not mentionned here, we only restate the main components and the main modifications made to the system.

Our framework front-end is an application, installed as an application package in the highest level of Android system: *application layer*. This is the layer where users are installing applications. This way, users who want to manage their privacy can use the application by launching it from the menu of Android, and users who do not care, are not bothered with it.
This application, named *PrimAndroid*, allows thus users to create groups of applications, manage those groups and add privacy rules to constrain applications belonging to those same groups. The application also manages conflicts between added constraints and dependencies (*shared uid type*) between those same applications. If an application is added to a particular group, and this application shares `UID` with another application, then this last one is also added to this group. If an application is removed, dependent applications are also removed from the concerned group.

Constraints created by users are generated in ODRL format using ODRL parsers, see Appendix B. Those privacy policies are then stored in a policy database which is the reserved memory of the PrimAndroid application.

Into the layer below, the *framework layer*, we have added a policy manager: *PPolicyManager*, which has for role to edit and read the policy database of the upper layer, to centralize the access to this database and avoid multiple concurrent accesses.
As said before, in Section 2.2, all components of Android applications having interraction with users or files are *Activities*. The main class, parent of all activites, is in the framework

layer and was thus also modified with head-functions of the *PPolicyManager*. This allows all activities to access this policy database by calling their parent which is using *PPolicyManager*. Of course, only *PrimAndroid* has the right to edit the privacy rules, but each application can read those rules if needed.

Another important class of the Android framework we have modified is *ContextImpl* which is the common implementation of Context API, which provides the base context object for each activity and other application components. Context, that is, an interface to global information about an application environment.
When a particular application access a particular resource, two methods are called in this class:

- public int checkPermission(String permission, int pid, int uid);
- public int checkCallingPermission(String permission);

Both methods are called to check wheter or not a particular application has the right to access a particular resource. Those methods then return an integer with the result of the check. The difference between those two methods does not matter here, but we had to modify both of them to add the checking of our policy database after the checking of the classics Android permissions.
Plus, when a notification is not granted to an application or if an application violates a constraint of the group where it belongs (if in any group), we need to launch a notification to warn user about the issue, and the fact that the permission was not granted. The notification system is managed by *NotificationManager* which was not modified and is called for notifications launching.

The checking of *PrimAndroid* permissions is not made in *ContextImpl* to avoid being messy and unclear. The check is devoted to a class: *PermissionChecker* created from scratch. This class is directly communicating with *PPolicyManager* to be able to check permissions gaven to applications. This class is only called from the two methods mentionned up here, and its only goal is the checking of permissions and the return of a boolean saying wheter or not the permission asked is granted.

6.5 Detailed Modifications

We will now enter the details of the modifications made to the Android sources, and explain each modification made in the architecture as explained in the last subsection. We will begin by describing the application PrimAndroid which was developed from scratch as an android application package. Then, we will develop the modifications we made to the framework layer of Android sources.

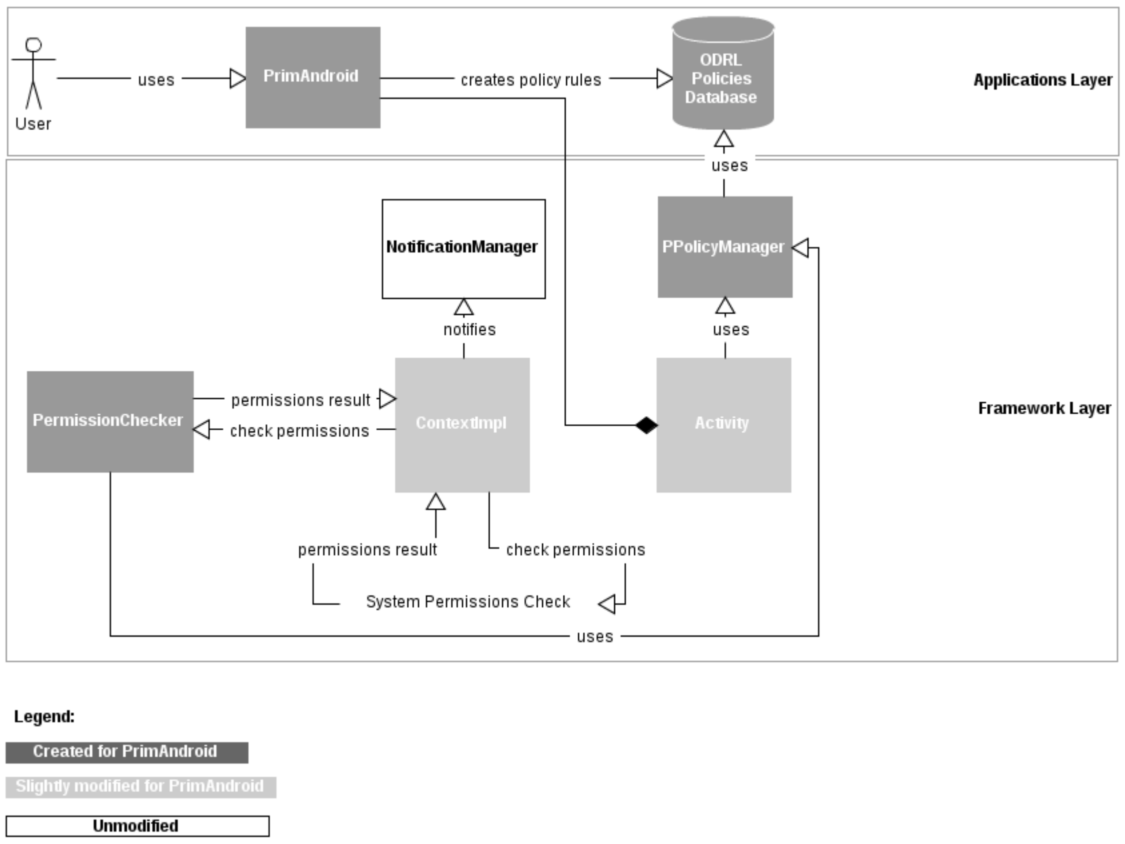


Figure 15: Integration of the framework into Android

6.5.1 Application Layer

As explained in the previous Section, our prototype contains one "classic" android application package. This package is thus an ".apk" and is installed by default in our read-only memory version of Android (ROM), in the package folder: *"path-to-android-sources"/packages/apps/PrimAndroid*. In this package we have several classes and activities. We will here details for each class: an overview of the purpose of this same class and a list of methods with specifications (non-exhaustive). Let's remind here that when we are talking about "activity", we mean a Java class with user interraction (as specified by Android developers guide [8]). All activities extend the super class: *android.app.Activity*, visible in the figure 15.

com.android.PrimAndroid.java *Overview:* Activity displaying all existing groups as a list with menu options for adding/deleting group(s). This is the main activity (The first user will see).

```
/**
 * Fill ListView with all groups names
 */
private void computeGroupsName(){}
```

This method gets all the groups from the policy database and fulfill the group list.

com.android.AddGroupActivity.java *Overview:* Activity allowing the add of a new group of applications.

```
/**
 * @param group: the name of the group to check
 * @return true if group does not exists, false otherwise
 */
private boolean checkUnique(String group){}
```

This method takes the name of the group choosed by user as argument and returns true if and only if this group name does not already exists, false otherwise.

com.android.AddConsActivity.java *overview:* Activity handling the add of constraints inside the selected group and check for potential conflicts. This activity is mainly composed of listeners, except for the conflict detection algorithm.

```
/**
 * @param a: list of actual constraints of the group
```

[8] `http://developer.android.com/guide/index.html`

```
 * @param toAdd: constraint to add
 * @return true iff constraint is conflicting, false
    otherwise
 *
 */
private boolean checkInnerConflicts(ArrayList<Constraint> a
    , Constraint toAdd){}
```

This method checks conflicts, before adding constraints, with existing constraints of the same group. Only conflicts for location and internet constraints could arise, because these are the only resources where we allow the add of a constraint of time or a constraint of date or yet a constraint of localisation. The algorithm used is derived from the conflict detection algorithm in Section 4.7.2. Each constraint added is checked for compliance on time, date or localisation with existing constraints. If at one moment, one association of two constraints is conflicting, the add is canceled and the user is warned. It returns false if and only if no conflicts arise, true otherwise.

com.android.AddAppActivity.java *overview:* Activity allowing the add of an application to the selected group.

```
/**
 * @param appName: name of the application
 * @param appList: list of all applications in all the
    groups
 * @param indexGroup: number of the group
 */
private void checkSharedUIDs(String appName, ArrayList<
    Group> appList, int indexGroup){}
```

This method checks if a particular application is sharing `UID` with another application of another group than selected. If some applications sharing `UID` are found, they are also added in the group as dependent applications, the user is then warned.

```
/**
 * @param group: group selected
 * @param app: app. package name unicity
 * @return true iff app. name does not exists yet in the
    group
 *    false otherwise
 */
public boolean checkUnique(String group, String app){}
```

This method checks that the application package added by user does not already exists in the selected group, return false if and only if this is the case, true otherwise.

```java
/**
 * @param appList: list of all applications inside all
    groups
 * @param targetGroup: group where the application has to
    be added
 * @param appName: name of the application
 * @return false iff application can be added
 */
private boolean isInAnotherGroup(ArrayList<Group> appList,
   String targetGroup, String appName){}
```

This method checks that the application does not already belongs to another group than selected. If true, and no shared `UID` feature is found within this other group and this application, application is added, else, if true and a shared `UID` is found, it is refused. If false, application is added.

com.android.DelAppActivity.java *Overview:* Activity allowing deletion of applications inside a group.

```java
/**
 * @param appName: the application name
 * @param appList: the application list
 * @param indexGroup: the group index in the applist
 */
public void checkSharedUIDs(String appName, ArrayList<Group
   > appList, int indexGroup){}
```

This method checks if an application use the shared `UID` feature to delete dependent applications from the group too.

```java
/*
 * @param group: the group containing the application to
    delete
 * (because application could belong to several group)
 * @param toDel: the application to delete
 */
private void removeFromList(String group,String toDel){}
```

This method removes the selected application from selected group by calling `checkSharedUIDs` method.

com.android.GroupView.java *Overview:* One of the two tabs of GroupActivity gui, showing list of applications inside selected group.

com.android.DelConsActivity.java *Overview:* Activity allowing the deletion of constraints from the selected group.

com.android.DelGroupActivity.java *Overview:* Activity allowing the deletion of a group of applications, even if not empty. In this case, all constraints applied to this group and thus, to those applications, are freed.

com.android.ConstraintsView.java *Overview:* The other tab of GroupActivity gui, showing list of constraints inside the selected group.

com.android.GroupActivity.java *Overview:* Activity displaying two tabs after selecting a group: Applications and Constraints: GroupView and ConstraintsView.

6.5.2 Framework Layer

The classes modified in the layer below the application layer contain important Android system classes, and our modifications are mandatory in order to have PrimAndroid running. It is why we had to make our own version of Android from the sources (ROM) and not just a sample application package. We will restate here those modifications and detail them.

Activity *Overview:* As said before, in the last Section, this class is the parent of all activities of the application layer. *"An activity is a single, focused thing that the user can do. Almost all activities interact with the user, so the Activity class takes care of creating a window for you in which you can place your UI with. To start another activity, all activity classes must have a corresponding declaration in their package's AndroidManifest.xml. The Activity class is an important part of an application's overall lifecycle and the way activities are launched and put together is a fundamental part of the platform's application model"* [9]

Here are the methods added for the purpose of PrimAndroid:

```
/**
 * @param context: context of calling application
 */
public List<String> getGroupsName(Context context){}
```

This method returns a list of all the groups' names from the policy database.

[9]For a detailed perspective on the structure of Android applications and lifecycles, please read `http://developer.android.com/guide/index.html`

```
/**
 * @param group: name of the group
 * @param context: context of calling application
 */
public Group getApplicationsByGroup(String group, Context
    context){}
```

This method returns a Group object from a group name from the policy database.

```
/**
 * @param context: context of calling application
 */
public static List<Group> getAllApplications(Context
    context){}
```

This method returns a list of all the groups from the policy database.

```
/**
 * @param group: name of the group
 * @param context: context of calling application
 */
public static List<Constraint> getConstraints(String group,
     Context context){}
```

This method returns a list of all the constraints of all the groups from the policy database.

```
/**
 * @param list list of all groups
 * @param context: context of calling application
 */
public void setGroupFile(List<Group> list, Context context)
    {}
```

This method sets the groups list objects in the policy database.

```
/**
 * @param list: list of all accesses logs
 */
public void setAccessFile(List<Access> list){}
```

This method sets the access logs list in the access database. One access is added for a group, when this group accesses a particular resource constrained by a maximum number of accesses for all applications of this group.

```
/**
 * @param groupName: name of the group
 * @param list: list of all constraints of all groups
 * @param context: context of calling application
 */
public void setConstraints(String groupName, List<
    Constraint> list, Context context){}
```

This method sets the constraints of all the groups in the policy database.

ContextImpl *Overview: "Common implementation of Context API, which provides the base context object for Activity and other application components"*. It is this class that is called for the checking of permissions when an application asks access to a particular resource. We have thus modified this class to add the check of PrimAndroid permissions, via our policy database.

```
/**
 * @param permission: the resource which is targeted
 * @param pid: the PID of the application targetting this
     resource
 * @param uid: the UID of the application targetting the
     resource
 *
 * @return PackageManager.PERMISSION_GRANTED if permission
     is granted, PackageManager.PERMISSION_DENIED, if
     permission
 * is not granted and PackageManager.CONSTRAINTS_VIOLATED
     if constraints from PrimAndroid's policy database are
 * violated
 */
public int checkPermission(String permission, int pid, int
    uid) {}
public int checkCallingPermission(String permission) {}
```

This method checks whether or not an application has the right to access a particular resource. Two checks are made: One for the permissions allowed by users at the installation, one for the constraints of the policy database of PrimAndroid application.
The second method is basically the same as the first one, exception made that it is only

used for checking if the calling process of an Inter-Process Communication (IPC) [10] we are handling has been granted a particular permission. In the non-modified system, this allows to avoid any application to call any other application to use permissions granted to this last one. Exception made of applications sharing `UID` as explained before and also constraints over permissions. Indeed, if an application A has the same permission has an application B but with different constraints set (with Apex by example) on those permissions, constraints are not taken into account by this mechanism, so this method would end with a permission granted, see figure 12 for example. But, as we added the check of PrimAndroid policy database constraints, this is avoided. Both permissions and constraints are checked over IPC mechanism.

PPolicyManager *Overview:* This class is used as a proxy to access the policy database. Each method of this class is the same as the Activity class above, but in this last class those are just transparent methods to call this PPolicyManager class. We will not restate the methods here, as they have already be explained in the Activity class and as PPolicyManager is used only by system, and at no moment has to be called by an application package. If an application wants to access the policy database, it should use the Activity class.

PermissionChecker *Overview:* This class is used by ContextImpl.class above to check permissions related to the policy database of PrimAndroid and to verify constraints setted on those policies (time,date,localization,accesses,prohibition). To connect with the policy database, this class also uses the PPolicyManager.class from above. This class can be used by any other application to check constraints, permissions or prohibition, as it is a read-only access to this information.

```
/**
 * @param app: the application URN (ex: com.android.browser
    )
 * @parem resource: the resource targetted by the
    application
 */
private void checkPermission(String app, String resource){}
```

This method checks whether or not the application has the permission (PrimAndroid point-of-view) to access this resource. All constraints are thus checked for the group containing the application. Those check are not made by the method but are called in the body: checkProhibitions, checkAccess, checkDay, checkCountry, checkTime. Below find the header of those functions.

[10] `http://developer.android.com/guide/developing/tools/aidl.html#PassingObjects`

```
private void checkProhibitions(ArrayList<Constraint>
   constraints, String app, String resource){}
private void checkAccess(ArrayList<Constraint> constraints,
    String app, String resource){}
private void checkDay(ArrayList<Constraint> constraints,
   String app, String resource){}
private void checkCountry(ArrayList<Constraint> constraints
   , String app, String resource){}
private void checkTime(ArrayList<Constraint> constraints,
   String app, String resource){}
```

6.6 Validation of prototype

6.6.1 P-RBAC Model Confrontation

Now that PrimAndroid has been presented, how can we state that it validate our P-RBAC-based model?

Let's restate our model:

- A set A of applications, a set G of dependency groups, a set R of resources, a set X of actions, a set O of obligations and a condition language LC_0.
- The set of Resources Permissions $RP = \{(x, r)|x \in X, r \in R\}$
- The set of Privacy-sensitive Resources Permission $PRP = \{(rp, c, o)|rp \in RP, o \in O,$ c is an expression of $LC_0\}$.
- Application Assignment $AA \subseteq A \times G$, a many-to-many mapping application to dependency group assignment relation.
- Privacy-sensitive Resources Permission Assignment $PRPA \subseteq G \times PRP$, a many-to-many mapping privacy-sensitive resource permission to dependency group assignment relation.

Our prototype proposes a set of resources permissions retrieved from the system's permissions. On each one of those permissions, user can add one or more constraints of a predefined set proposed by our framework. Those constraints allow users to manage their privacies with more granularity. Thus, we have privacy-sensitive resources permission as a permission constrained by user under some privacy rules. The obligation is here considered as logging. Indeed, each deny or grant of permission is currently logged by the system.
Our framework proposes to users to add applications to groups of similar applications, with automated dependences add. It is the equivalent of the application assignement in our model. Finally, as each privacy-sensitive resource permission added by user, has a

group of applications as target, it means that we also have respected the many-to-many mapping of privacy-sensitive resource permission to dependency group assignement relation.

Our prototype does validate our modified P-RBAC model.

6.6.2 ODRL Model Confrontation

We also have to validate the usage of rights expression languages, and especially ODRL.

Each constraint or prohibition added by user over a group of applications is generated in ODRL, such as discussed in 5.2.5. Each group is associated with a file containing all policy rules of this same group. The file is named "NAME-OF-THE-GROUP".xml. Each group as its own file for complexity, and thus performances reasons. Indeed, if all policy rules where in a file, checks for one group would imply to parse a file containing all policies of all existing groups. Instead, our system allows to parse only rules of the concerned group.

The policies are generated, read, and modified via two parsers: *ODRLWriter.java* and *ODRLReader.java*, both using SAX [11] as API for parsing XML files. Those parsers can be found in appendix, see Appendix B.

Our prototype does validate our ODRL model.

6.7 Limits and Discussion

Although we have shown that our model was implementable on a real system and that all constraints added by users could be expressed using ODRL, there are also some limits to our prototype, as the point was not to be exclusive but to validate our model and ideas presented in this work. We will discuss them here.

- **Overhead for user:** The fact that user has to manage groups (partially, `UID` dependent applications are automaticly added) and add constraints over those groups manually can be an overhead to users. But we started from the following assumption, which seems the best approach in our opinion, but may not be for others:
  A certain part of users do not care about privacy issues. It means that for such user, forcing to add constraints over resources would be an overhead, it is why our framework is based on a top-level application, which allows users to use it if they want to, and to completely avoid it if they don't want to or if they do not understand what is the goal of such application. For users who do care a bit but do not know that they are subject to those issues, our prototype can be a limit. It is why we think that it is important to have both user-centric framework and machine-based framework for

[11]SAX API: `http://download.oracle.com/javase/1.4.2/docs/api/org/xml/sax/package-summary.html`

a highly complete privacy management system. Indeed, this way users have in all cases, a good base that manages there privacy from machine-learned techniques, and for users who are not satisfied with such, they can use our application to add more personnalized rules and manage dependencies between applications.
Thus, as privacy is subjective, varying from one person to another, the only way, always in our opinion, to be complete is to have both user-centric and machine-centric techniques.

- **Dynamic Dependencies:** As said before, our prototype is able to automaticaly add applications to a group which are sharing `UID` with an application just added in this same group. But the other form of dependency, dynamic dependency, cannot be managed using a user-centric framework. Indeed, in Android system, each application using another application without sharing the same `UID`, launches a broadcast with a particular message. Each target applications which has a filter for this broadcast message will respond. This system is thus non-deterministic, as zero, one or more applications can respond. It is what we call dynamic dependencies. The only way to catch those is to add a machine-learning tool which captures those messages, and name of reacting applications. But as many applications can answers, we cannot be sure of which one is really used.

- **Limited set of constraints:** This last limit is more part of a future work. The set of constraints available is quite restricted for now, focusing on the main resources generally causing privacy issues. But with users studies and time, this set could be easily expanded to include more granular constraints.

7 CONCLUSION

In this Section we will present the main findings and conclusions based on research conducted in this thesis. The purpose of this conclusion is to see if we have answered the research questions stated in Section 1.3. Each one of the research questions will be discussed in separate subsections. Finally future research work is discussed.

7.1 What are privacy issues and what are the assets to protect when talking about privacy management in mobile applications?

We have began this work by explaining what are privacy issues and especially how can we talk of privacy issues in a mobile environement and what are the assets and resources to manage to avoid such issues. We have also looked at the evolution of law in terms of privacy protection, as many researchs are based on european guidelines like *OECD* directives.

7.1.1 What are the limitations of current systems in terms of privacy?

We have presented a non-exhaustive state-of-the-art of existing tools (like *Apex*[32], *Taint-Droid*[15],...) and we have analyzed their limits in terms of privacy management but also their contributions to the domain.

7.1.2 How can we use tools are our disposal to override those problems?

Along with the study of existing tools in terms of privacy management, we have studied P-RBAC which is a tool we used to override the problem of the modelisation of privacy issues scenarios in mobile applications. We, then, extended this model to take dependencies of applications into account.
Beside P-RBAC, we have reasonned all along with the current permission system of Android applications.
The second tool we used, is ODRL as a rights expression language.

7.2 How can we use access controls models to reason about privacy issues in mobile applications and how can we adapt those models to our purpose?

As we explained in the last question, we have adapted a model, P-RBAC, designed for taking privacy into account in role-based access control models and we have extended this model to mobile applications, especially in Android system. We have shown how to express scenarios of this system in such a model and how to reason about privacy issues using this same model.

7.3 What can rights expression languages offer to such models in terms of expressiveness, privacy evolution and usage of a standard?

After having introduced rights expression languages in the background Section 2.3, we have discussed their possible contributions in terms of expressiveness, privacy change and usage of a standard. We also have shown limits of such languages. We have finally introduced ODRL as the best-fitted REL for mobile applications.

7.4 How can we apply those adapted models to a specific environment like Google's Android system? How can we incorporate rights expression languages in such a system?

Using ODRL, we have tried to express android scenarios to focus only on a subpart of the language. This has lead to the expression of our modified P-RBAC model using such an expression language.
In the prototype, *PrimAndroid*, we have included a policy database where policies are written in ODRL and are parsed by the system for the completude of the base Android permissions system.

7.5 Future Works

We have exposed our work through this thesis with the goal of improving privacy rights management on mobile applications, especially Android applications and reason about concrete scenarios of privacy issues. We have seen how we could modelize such scenarios and how we could express them using a rights expression language, which was ODRL for us. Although, there is still a lot of interesting future works to do, to go on with research on privacy in mobile applications which, with the constant growth of mobile phones, will be more and more a key asset and a need for users. We identify here some major points which could be the going on of this work, but there are probably many others which could be discussed.

- **Privacy change:** One of the biggest interest of porting rights expression languages to mobile applications was the evolution of privacy through the evolution of application. Indeed, as such languages are quite complete, with a lot of possibilities, the policies could possibly be adapted by the system whenever an application evolves. This is an active topic of research and a lot of teams are working on this privacy change idea. It could allow to end-up with automated updated policies when applications are updated. On Android applications, by example, we could see if permissions of the *"AndroidManifest.xml"* file have been modified, if so we could automate the repercussion on the policy database. But it would also be interesting to generate constraints for the user by taking the works on context-aware privacy into account where constraints could be modified whenever we are in a reunion or in our car by example. The evolution of technology (bluetooth, accelerometer, GPS,...) will allow more precisely to detect such changes of context, but they also will open smartphones to more privacy issues, which is thus a paradox to take into account.

- **User privacy requirements studies:** As we said before, privacy is highly subjective. One particular user may attach importance to one particular asset, another user may not. It means that constraints proposed by a tool like *Apex* or our prototype, or even tools proposed by operating systems, could end-up with unsatisfied users privacy-speaking. User studies would allow to have a comprehensible set of constraints with a lot of granularity and easily adjustable to any user.

- **Machine-based and User-centric techniques:** As we discussed in the validation Section, see Section 6.7, we think that the best framework for privacy management would contains both machine-based (like [18]) and user-centric (like our prototype) techniques to have a maximum granularity.

- **Study of other mobile systems:** Finally, we have focused here on Android applications, but we have seen that other systems such as iOS have also known issues. It would thus be interesting to study the validity of our model and its application to such systems and probably extend this model in function.

References

[1] "the organisation for economic co-operation and development, guidelines on the protection of privacy and transborder flows of personal data", 1999. http://www.oecd.org/.

[2] Xacml: A new standard protects content in enterprise data exchange, 2003. http://java.sun.com/developer/technicalArticles/Security/xacml/xacml.html.

[3] "platform for privacy preferences (p3p) project", 2007. http://www.w3.org/P3P/.

[4] Boolean satisfiability problem. World Wide Web electronic publication, 2010.

[5] "INTERNATIONAL ORGANIZATION FOR STANDARDIZATION ISO/IEC JTC 1/SC 29/WG 11". Coding of moving pictures and audio. 2005. http://mpeg.chiariglione.org/technologies/mpeg-21/mp21-rel/index.htm.

[6] Alasdair Allan and Pete Warden. Got an iphone or 3g ipad? apple is recording your moves, 2011. http://radar.oreilly.com/2011/04/apple-location-tracking.html.

[7] F. Baader and U. Sattler. An overview of tableau algorithms for description logics. *Studia Logica*, 69:5–40, 2001.

[8] Guangdong Bai, Liang Gu, Tao Feng, Yao Guo, and Xiangqun Chen. Context-aware usage control for android. In *Security and Privacy in Communication Networks*, volume 50 of *Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering*, pages 326–343. Springer Berlin Heidelberg, 2010.

[9] Charalambides, Flegkas, Pavlou, Bandara, Lupu, Russo, Dulay, Sloman, and Rubio-Loyola. Policy conflict analysis for quality of service management. IEEE, 2006.

[10] "ContentGuard". "the digital rights language for trusted content and services". http://www.xrml.org/.

[11] Mauro Conti, Vu Nguyen, and Bruno Crispo. Crepe: Context-related policy enforcement for android. In Mike Burmester, Gene Tsudik, Spyros Magliveras, and Ivana Ilic, editors, *Information Security*, volume 6531 of *Lecture Notes in Computer Science*, pages 331–345. Springer Berlin / Heidelberg, 2011.

[12] Lucas Davi, Alexandra Dmitrienko, Ahmad-Reza Sadeghi, and Marcel Winandy. Privilege escalation attacks on android. In Mike Burmester, Gene Tsudik, Spyros Magliveras, and Ivana Ilic, editors, *Information Security*, volume 6531 of *Lecture Notes in Computer Science*, pages 346–360. Springer Berlin / Heidelberg, 2011.

[13] Cécile de Terwangne and Jean-Noël Colin. Défis pour la vie privée et la protection des données posés par la technologie. 2011.

[14] Microsoft Corporation Encarta, World English Dictionary. Encarta encyclopedia, 2009. `http://encarta.msn.com/`.

[15] William Enck, Peter Gilbert, and Byung-Gon Chun. Taintdroid: An information-flow tracking system for realtime privacy monitoring on smartphones. 2010.

[16] Jeanne Ferrante, Karl J. Ottenstein, and Joe D. Warren. The program dependence graph and its use in optimization. *ACM Trans. Program. Lang. Syst.*, 9:319–349, July 1987.

[17] "Organisation for Economic Co-operation and Development (OECD)". "report on the cross-border enforcement of privacy laws", 2006.

[18] Adam P. Fuchs, Avik Chaudhuri, and Jeffrey S. Foster. "scandroid : Automated security certification of android applications".

[19] Google. Content providers. 2010. `http://developer.android.com/guide/topics/providers/conte`

[20] Google. Security and permissions. 2010. `http://developer.android.com/guide/topics/security/`

[21] Governatori and Milosevic. A formal analysis of a business contract language. *International Journal of Cooperative Information Systems, vol. 15, no. 4, pp. 659–685*, 2006.

[22] Guido Governatori and Renato Iannella. Modelling and reasoning languages for social networks policies. *2009 IEEE International Enterprise Distributed Object Computing Conference*, 2009.

[23] Yin H., Song D., Egele M., Kruegel C., kirda, and E. Panorama. Capturing system-wide information flowfor malware detection and analysis. *In Proceedings of ACM Computer and Communications Security*, 2007.

[24] Altman I. *Privacy Regulation: Culturally Universal or Culturally Specific?* 1977.

[25] "Information Commissioner's Office (ICO)". "the guide to data protection", 1998.

[26] ODRL initiative. Open digital rights language, 2010. `http://odrl.net`.

[27] "ORGANISATION INTERNATIONALE DE NORMALISATION ISO/IEC". "coding of moving pictures and audio, mpeg-21 overview v.5", 2002.

[28] Silvia Llorente, Jaime Delgado, Rubén Barrio, and Xavier Maroñas. Translation between xml-based rights expressions using uml and relational models. 2006.

[29] Clara Mancini, Keerthi Thomas, Yvonne Rogers, Blaine A. Price, Lukasz Jedrzejczyk, Arosha K. Bandara, Adam N. Joinson, and Bashar Nuseibeh. From spaces to places: Emerging contexts in mobile privacy. *UbiComp 2009*, September 2009.

[30] Xavier Maroñas, Eva Rodríguez, and Jaime Delgado. An architecture for the interoperability between rights expression languages based on xacm.

[31] Nauman. Android security, a survey. so far so good. World Wide Web electronic publication, 2010.

[32] Nauman, Khan, and Zhang. "apex : Extending android permission model and enforcement with user-defined runtime constraints". April 2010.

[33] Q. Ni, A. Trompette, E. Bertino, and J. Lobo. Privacy-aware role based access control. ACM Press, June 2007.

[34] Sadeh Norman, Hong Jason, Cranor Lorrie, Fette Ian, Kelley Patrick, Prabaker Madhu, and Rao Jinghai. "understanding and capturing people's privacy policies in a mobile social networking application". 2008.

[35] Council of the OECD. "oecd guidelines on the protection of privacy and transborder flows of personal data", 1980. `http://www.oecd.org/document/18/0,3343,en_2649_34255_1815186_1_1_1_1,00.html`.

[36] L. Palen and P. Dourish. Unpacking "privacy" for a networked world. *Proceedings of the conference on Human factors in computing systems - CHI '03*, page 129, 2003.

[37] Jose Prados, Eva Rodr?guez, and Jaime Delgado. Interoperability between different rights expression languages and protection mechanisms. *Automated Production of Cross Media Content for Multi-Channel Distribution, International Conference on*, 0:145–152, 2005.

[38] Jamkhedkar Pramod, Heileman Gregory, and Martínez-Ortiz Ivan. "the problem with rights expression languages". 2006.

[39] Cambridge University Press. Cambridge dictionaries, 2011. `http://dictionary.cambridge.org/dictionary/british/algorithm`.

[40] Oxford University Press. Oxford dictionaries, 2011. `http://www.oxforddictionaries.com`.

[41] Ian Reay, Scott Dick, and James Miller. "a large-scale empirical study of p3p privacy policies". *ACM Transactions on The Web, Vol. , No. 2, Article 6*, 2009.

[42] Asaf Shabtai, Yuval Fledel, Uri Kanonov, Yuval Elovici, and Shlomi Dolev. Google android: A state-of-the-art review of security mechanisms, 2008.

[43] Murray Shanahan. The event calculus explained.

[44] Eric Smith. "iphone applications and privacy issues: An analysis of application transmission of iphone unique device identifiers (udids)". 2010. `www.pskl.us`.

[45] W3C. `http://www.w3.org/TR/odrl/`.

[46] Lu Yao and Zhang Guoyi. A dl-based approach for eliminating concept restriction conflicts in ontology merging. *2008 IFIP International Conference on Network and Parallel Computing*, 2008.

A OECD Guidelines on the Protection of Privacy and Transborder Flows of Personal Data

A.1 PART ONE: GENERAL DEFINITIONS

A.1.1 For the purposes of these Guidelines:

- “data controller” means a party who, according to domestic law, is competent to decide about the contents and use of personal data regardless of whether or not such data are collected, stored, processed or disseminated by that party or by an agent on its behalf;
- “personal data” means any information relating to an identified or identifiable individual (data subject);
- “transborder flows of personal data” means movements of personal data across national borders.

A.1.2 Scope of the Guidelines

These Guidelines apply to personal data, whether in the public or private sectors, which, because of the manner in which they are processed, or because of their nature or the context in which they are used, pose a danger to privacy and individual liberties.

These Guidelines should not be interpreted as preventing:

- the application, to different categories of personal data, of different protective measures depending upon their nature and the context in which they are collected, stored, processed or disseminated;
- the exclusion from the application of the Guidelines of personal data which obviously do not contain any risk to privacy and individual liberties; or
- the application of the Guidelines only to automatic processing of personal data.

Exceptions to the Principles contained in Parts Two and Three of these Guidelines, including those relating to national sovereignty, national security and public policy (“ordre public”), should be:

- as few as possible, and
- made known to the public.

In the particular case of Federal countries the observance of these Guidelines may be affected by the division of powers in the Federation.

These Guidelines should be regarded as minimum standards which are capable of being supplemented by additional measures for the protection of privacy and individual liberties.

A.2 PART TWO: BASIC PRINCIPLES OF NATIONAL APPLICATION

A.2.1 Collection Limitation Principle

There should be limits to the collection of personal data and any such data should be obtained by lawful and fair means and, where appropriate, with the knowledge or consent of the data subject.

A.2.2 Data Quality Principle

Personal data should be relevant to the purposes for which they are to be used, and, to the extent necessary for those purposes, should be accurate, complete and kept up-to-date.

A.2.3 Purpose Specification Principle

The purposes for which personal data are collected should be specified not later than at the time of data collection and the subsequent use limited to the fulfilment of those purposes or such others as are not incompatible with those purposes and as are specified on each occasion of change of purpose.

A.2.4 Use Limitation Principle

Personal data should not be disclosed, made available or otherwise used for purposes other than those specified in accordance with Paragraph A.2.3 except:

- with the consent of the data subject; or
- by the authority of law.

A.2.5 Security Safeguards Principle

Personal data should be protected by reasonable security safeguards against such risks as loss or unauthorised access, destruction, use, modification or disclosure of data.

A.2.6 Openness Principle

There should be a general policy of openness about developments, practices and policies with respect to personal data. Means should be readily available of establishing the existence and nature of personal data, and the main purposes of their use, as well as the identity and usual residence of the data controller.

A.2.7 Individual Participation Principle

An individual should have the right:

- to obtain from a data controller, or otherwise, confirmation of whether or not the data controller has data relating to him;
- to have communicated to him, data relating to him within a reasonable time; at a charge, if any, that is not excessive; in a reasonable manner; and in a form that is readily intelligible to him;
- to be given reasons if a request made under subparagraphs(a) and (b) is denied, and to be able to challenge such denial; and
- to challenge data relating to him and, if the challenge is successful to have the data erased, rectified, completed or amended.

A.2.8 Accountability Principle

A data controller should be accountable for complying with measures which give effect to the principles stated above.

A.3 PART THREE: BASIC PRINCIPLES OF INTERNATIONAL APPLICATION: FREE FLOW AND LEGITIMATE RESTRICTIONS

Member countries should take into consideration the implications for other Member countries of domestic processing and re-export of personal data.

Member countries should take all reasonable and appropriate steps to ensure that transborder flows of personal data, including transit through a Member country, are uninterrupted and secure.

A Member country should refrain from restricting transborder flows of personal data between itself and another Member country except where the latter does not yet substantially observe these Guidelines or where the re-export of such data would circumvent its domestic privacy legislation. A Member country may also impose restrictions in respect of certain categories of personal data for which its domestic privacy legislation includes specific regulations in view of the nature of those data and for which the other Member country provides no equivalent protection.

Member countries should avoid developing laws, policies and practices in the name of the protection of privacy and individual liberties, which would create obstacles to transborder flows of personal data that would exceed requirements for such protection.

A.4 PART FOUR: NATIONAL IMPLEMENTATION

In implementing domestically the principles set forth in Parts Two and Three, Member countries should establish legal, administrative or other procedures or institutions for the protection of privacy and individual liberties in respect of personal data. Member countries should in particular endeavour to:

- adopt appropriate domestic legislation;
- encourage and support self-regulation, whether in the form of codes of conduct or otherwise;
- provide for reasonable means for individuals to exercise their rights;
- provide for adequate sanctions and remedies in case of failures to comply with measures which implement the principles set forth in Parts Two and Three; and
- ensure that there is no unfair discrimination against data subjects.

A.5 PART FIVE: INTERNATIONAL CO-OPERATION

Member countries should, where requested, make known to other Member countries details of the observance of the principles set forth in these Guidelines. Member countries should also ensure that procedures for transborder flows of personal data and for the protection of privacy and individual liberties are simple and compatible with those of other Member countries which comply with these Guidelines.

Member countries should establish procedures to facilitate:

- Information exchange related to these Guidelines, and mutual assistance in the procedural and investigative matters involved.
- Member countries should work towards the development of principles, domestic and international, to govern the applicable law in the case of transborder flows of personal data.

B ODRL Java Parsers

Here, we restate the code of the parser we created for ODRL XML reading, modifying and writing of policies.

B.1 ODRLReader.xml

```java
package android.app.parsers;

import java.io.BufferedReader;
import java.io.FileInputStream;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.StringReader;
import java.util.ArrayList;

import org.xmlpull.v1.XmlPullParser;
import org.xmlpull.v1.XmlPullParserException;
import org.xmlpull.v1.XmlPullParserFactory;

import android.app.objects.Constraint;


import android.util.Log;


public class ODRLReader {

  /**
   *
   * Read Constraints over a group in XML file
   *
   * @param fIn: fileInputStream to read
   * @return Group List
   */
  public ArrayList<Constraint> readXML(FileInputStream fIn)
     {

    //Bind the new file with a FileOutputStream
    InputStreamReader file = null;
    file = new InputStreamReader(fIn);
    BufferedReader f = new BufferedReader(file);
    ArrayList<Constraint> constraints = new ArrayList<
       Constraint>();

    XmlPullParserFactory factory = null;
    try {

      factory = XmlPullParserFactory.newInstance();
      factory.setNamespaceAware(true);
      XmlPullParser parser = null;
      parser = factory.newPullParser();
      parser.setInput((new StringReader(f.readLine())));
      int eventType = parser.getEventType();


      boolean prohibition = false;
      Constraint c = new Constraint();

      while (eventType != XmlPullParser.END_DOCUMENT){

        if(eventType==XmlPullParser.START_TAG){
          //Start group/app Tag
          if(!prohibition){
            if(parser.getName().equals("asset")){
              c = new Constraint();
              c.setResource(parser.getAttributeValue(0));
            } else if (parser.getName().equals("constraint"
               )){
              c.setType(parser.getAttributeValue(0).
                 substring(
                  parser.getAttributeValue(0).indexOf("/")
                     +1,parser.getAttributeValue(0).length
                     ()));
              c.setOperator(parser.getAttributeValue(1).
                 substring(
                  parser.getAttributeValue(1).indexOf("/")
                     +1,parser.getAttributeValue(1).length
                     ()));
              c.setRightOperand(parser.getAttributeValue(2)
                 );
            } else if (parser.getName().equals("prohibition
               ")){
              prohibition = true;
              c = new Constraint();
            }
          }else{
            if(parser.getName().equals("asset")){
              c.setResource(parser.getAttributeValue(0));
              c.setType("prohibition");
              c.setOperator("");
              c.setRightOperand("");
            } else if (parser.getName().equals("constraint"
               )){
              c.setType("prohibition");
              c.setOperator("");
              c.setRightOperand("");
            }
          }
        }
        if(eventType==XmlPullParser.END_TAG){
          //End perm Tag
          if(parser.getName().equals("permission")){
            constraints.add(c);
          }else{
            if(parser.getName().equals("prohibition")){
              constraints.add(c);
              prohibition=false;
            }
          }
        }
        eventType = parser.next();
      }

      file.close();

    } catch (XmlPullParserException e) {
      Log.i("com.android.PrimAndroid.GroupReader", "Null
         Parser");
    } catch (IOException e){
      Log.i("com.android.PrimAndroid.GroupReader", "IO
         Exception");
    }

    return constraints;
  }
}
```

B.2 ODRLWriter.xml

```
package android.app.parsers;

import java.io.FileOutputStream;
import java.io.IOException;
import java.util.ArrayList;

import org.xmlpull.v1.XmlSerializer;

import android.app.objects.Constraint;

import android.util.Log;
import android.util.Xml;


public class ODRLWriter {

  /**
   *
   * Write ODRL permission in XML file
   *
   * @param groupList: list of groups
   * @param file: FileOutputStream to write
   */
  public void writeXML(ArrayList<Constraint> c,
     FileOutputStream file,String groupName) {

    //we create a XmlSerializer in order to write xml data
    XmlSerializer serializer = Xml.newSerializer();
    try {
      serializer.setOutput(file,"UTF-8");
      serializer.startDocument("UTF-8", true);
      serializer.startTag("o", "policy");
      serializer.attribute("xmnls","o","http://odrl.net/2.0
         ");
      serializer.attribute("type","","o:privacy");
      serializer.attribute("uid","","urn:policy:"+groupName
         );

      for(int i=0; i<c.size(); i++){

        if(!c.get(i).getType().equals("prohibition")){

            //Start a permission
            serializer.startTag("o", "permission");
            serializer.startTag("o", "asset");
            serializer.attribute("","uid",c.get(i).
               getResourceName());
            serializer.endTag("o", "asset");
            serializer.startTag("o", "action");
            serializer.attribute("o","ressource", "action/
               present");
            serializer.endTag("o", "action");
            serializer.startTag("o", "constraint");

            if(c.get(i).getType().equals("count")){

              serializer.attribute("","name","o:constraint/
                 count");
              serializer.attribute("", "operator", "o:
                 operator/lteq");
            }else{
              if(c.get(i).getType().equals("timeInterval")){
                //To change
                serializer.attribute("","name","o:constraint/
                   timeInterval");
                serializer.attribute("", "operator", "o:
                   operator/"+c.get(i).getOperator());
              }else{
                if(c.get(i).getType().equals("dateTime")){
                  serializer.attribute("","name","o:
                     constraint/dateTime");
                  serializer.attribute("", "operator", "o:
                     operator/neq");
                }else{
                  //Location
                  if(c.get(i).getType().equals("spatial")){
                    serializer.attribute("","name","o:
                       constraint/spatial");
                    serializer.attribute("", "operator", "o:
                       operator/eq");
                  }
                }
              }
            }
```

```java
          serializer.attribute("", "rightOperand", c.get(i)
             .getRightOperand());
          //End a group
          serializer.endTag("o", "constraint");
          serializer.endTag("o", "permission");
        }else{
          //Start a prohibition
          serializer.startTag("o", "prohibition");
          serializer.startTag("o", "asset");
          serializer.attribute("","uid",c.get(i).
             getResourceName());
          serializer.endTag("o", "asset");
          serializer.startTag("o", "action");
          serializer.attribute("o","ressource", "action/
             present");
          serializer.endTag("o", "action");
          serializer.endTag("o", "prohibition");
        }
      }
      serializer.endTag("o", "policy");
      serializer.endDocument();
      file.close();
    } catch (IllegalArgumentException e) {
      Log.i("com.android.PrimAndroid.GroupWriter","Illegal
         Argument Exception");
    } catch (IllegalStateException e) {
      Log.i("com.android.PrimAndroid.GroupWriter","Illegal
         State Exception");
    } catch (IOException e) {
      Log.i("com.android.PrimAndroid.GroupWriter","IO
         Exception");
    }
  }
}
```

C Screenshots of PrimAndroid

This last appendix just restates some screenshots of our application prototype.

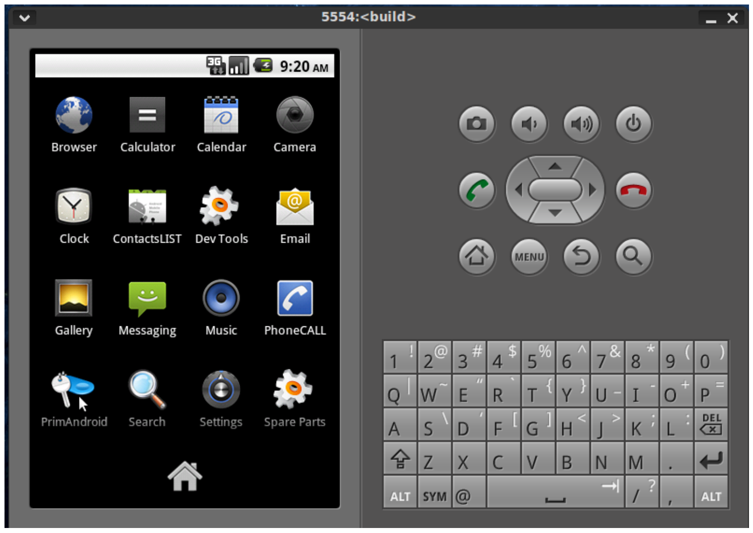
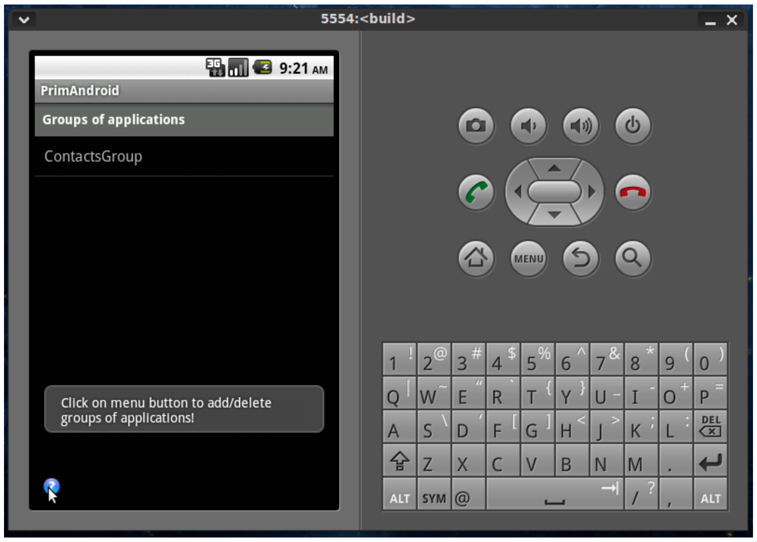


Figure 16: PrimAndroid in the Android menu



Click on menu button to add/delete groups of applications!

Figure 17: List of Groups of Applications

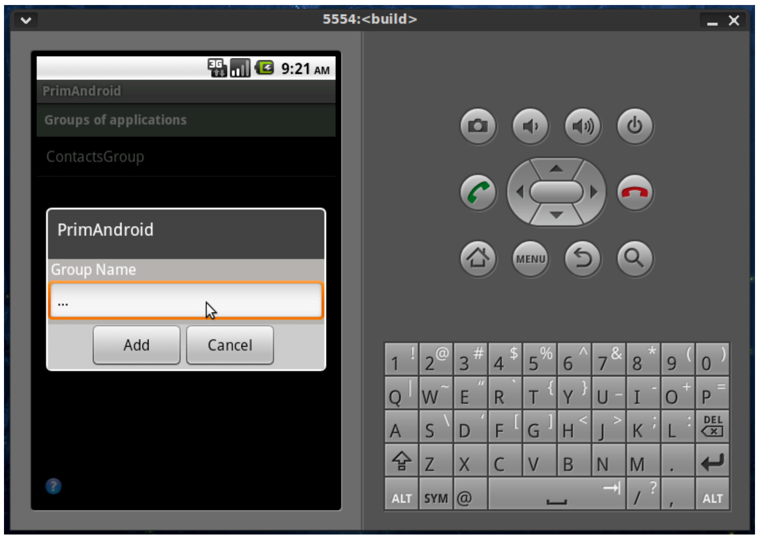
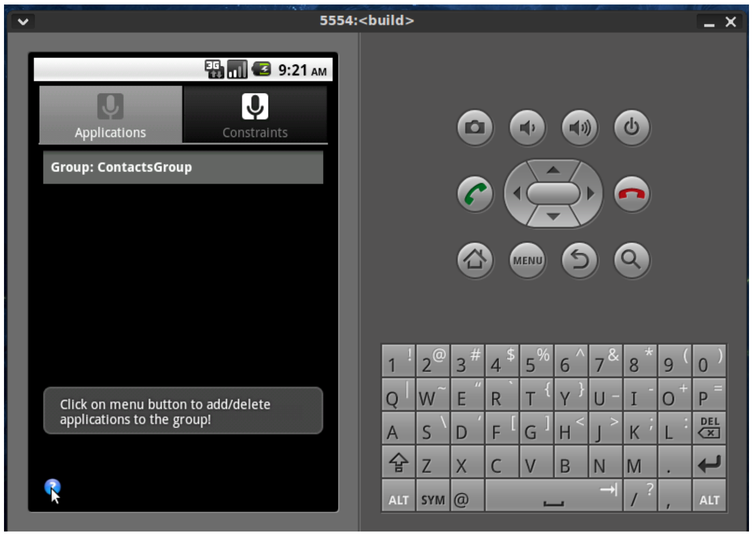
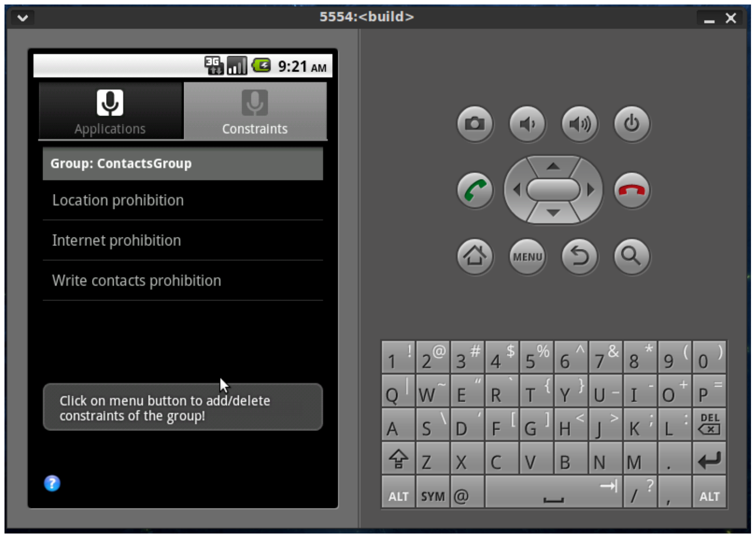


Figure 18: Adding a Group



Click on menu button to add/delete applications to the group!

Figure 19: Applications of a Group



Group: ContactsGroup

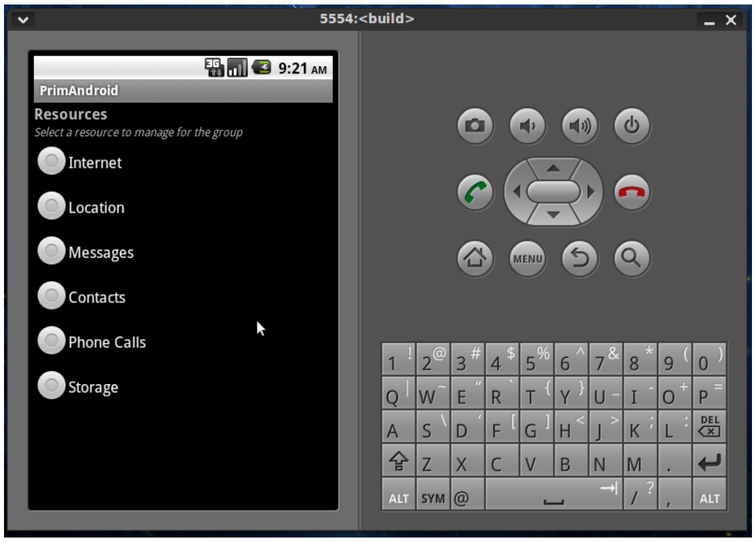
Location prohibition

Internet prohibition

Write contacts prohibition

Click on menu button to add/delete constraints of the group!

Figure 20: Constraints of a Group



Resources

Select a resource to manage for the group

Internet

Location

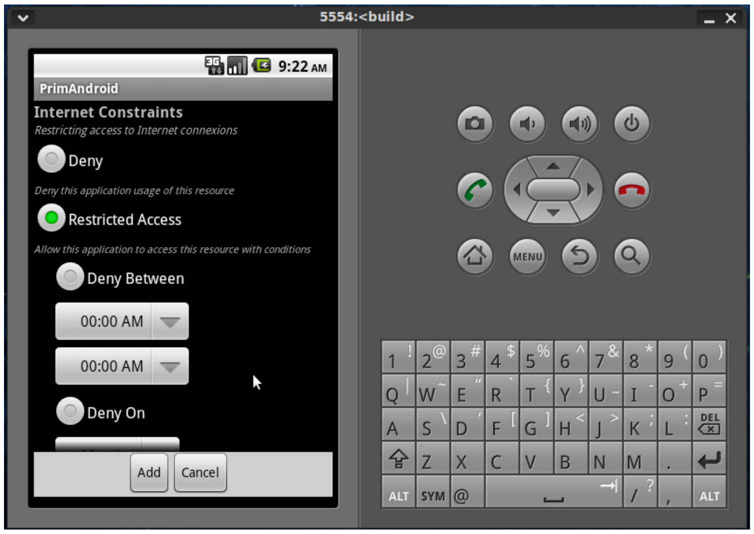
Messages

Contacts

Phone Calls

Storage

Figure 21: Selection of a Resource to Constraint for a Group



Internet Constraints

Restricting access to Internet connexions

Deny

Deny this application usage of this resource

[✓ Restricted Access]

Allow this application to access this resource with conditions

Deny Between

00:00 AM

00:00 AM

Deny On

Figure 22: Adding a Constraint to a Resource