Graph-Based Privacy Preference Expression for the Semantic Web

by

Ryan Randall Wagner

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of
Master of Engineering in Electrical Engineering and Computer Science
at the
MASSACHUSETTS INSTITUTE OF TECHNOLOGY
June 2004

© Ryan Randall Wagner, MMIV. All rights reserved.

The author hereby grants to MIT permission to reproduce and distribute publicly paper and electronic copies of this thesis document in whole or in part.
Graph-Based Privacy Preference Expression for the Semantic Web

by

Ryan Randall Wagner

Submitted to the Department of Electrical Engineering and Computer Science on May 20, 2004, in partial fulfillment of the requirements for the degree of Master of Engineering in Electrical Engineering and Computer Science

Abstract

The Web is changing. Originally a medium for human-readable documents, the next generation Semantic Web is opening up toolkit that provides vast opportunities for sharing information that is encoded in a machine-readable form. The opportunities for sharing data also are opportunities for encroaching on people’s privacy. While some technologies already exist for expressing privacy preferences, they do not integrate closely with the Semantic Web. In addition, previous approaches to privacy expression do not easily expand to data shared over multiple hops, with different privacy preferences at each hop. The Private Information Management Agent is an attempt to mitigate these concerns.

Thesis Supervisor: Daniel J. Weitzner
Title: Principal Research Scientist

Thesis Supervisor: Ronald L. Rivest
Title: Professor
Acknowledgments

This could not have been done without help from a lot of people. I want to thank Danny Weitzner for accepting me as an advisee and for taking valuable time to give me the guidance and advice that I needed on this project. Also, thanks go to Professor Rivest for helping Danny Weitzner and me out with our first M.Eng. thesis. In addition, the people at the W3C have been very helpful in giving me advice and pointing me in the right direction. Academic things aside, this never could have been done without the support of my family (Mom, Dad, and everyone else). Last, a special thanks to Carlos Palacio—he made sure that I did not starve during the final couple of weeks of this project, when I barely had time to eat.
# Contents

1 Introduction

2 The Inadequacy of Current Technologies
   2.1 Digital Rights Management .................................................. 17
   2.2 P3P ........................................................................ 20
   2.3 EPAL ........................................................................ 22
   2.4 Passport ...................................................................... 22
   2.5 Liberty Alliance .............................................................. 23
   2.6 Bridging the Gap ............................................................... 23

3 Tools from the Semantic Web
   3.1 XML .......................................................................... 25
   3.2 RDF and OWL ............................................................... 27
   3.3 Friend of a Friend (FOAF) ................................................. 28

4 PIMA Design Approach
   4.1 PIMA and FOAF .............................................................. 31
   4.2 Quick Introduction to XPath .............................................. 33
   4.3 PIMA and EPAL .............................................................. 34
   4.4 XPath and PIMA .............................................................. 36
   4.5 A Graph-Based Policy Approach ....................................... 38
   4.5.1 Viral Agreements ......................................................... 40
   4.6 The Focus on Use of Data Rather Than Storage of It .......... 41
List of Figures

2-1 An example P3P statement. ........................................ 21
3-1 Badly-formed XML. .................................................. 26
3-2 Well-formed XML. .................................................... 26
3-3 A simple FOAF file. ................................................. 29
4-1 The flow of Alice’s information when booking a flight. ........ 39
5-1 Outline of the PIMA protocol. ..................................... 44
5-2 A simple PIMA request to establish a relationship. ............ 46
5-3 A rule in PIMA. ....................................................... 49
5-4 A privacy policy in PIMA. ......................................... 49
5-5 Bob’s FOAF file with Alice’s data before the request. ......... 51
5-6 Charlie’s FOAF file with Alice’s data from Bob. ................ 52
5-7 Bob’s FOAF file after his agreement with Charlie. .............. 53
5-8 Bob’s denial to Charlie’s request. ................................ 54
5-9 Bob’s acceptance of Charlie’s request. ............................ 54
5-10 A transaction request from Charlie to Bob. .................... 56
5-11 A transaction reply form Bob to Charlie. ....................... 57
5-12 A PIMA privacy information node with transaction histories. . 58
Privacy is becoming more of a concern to people as technology achieves the ability to collect breathtaking amounts of information about individuals. Unfortunately, when it comes to data that businesses collect about customers and affiliates’ customers, there are few substantial United States federal legal protections for privacy [1]. Information collected during a transaction between a business and a customer is not considered the customer’s property, and the customer does not have any specific legal authority to control how that information is disseminated. [2] states “In general, federal laws do not regulate what information businesses can collect about their customers or the sale and distribution of that information. Consumers have the right to ‘opt out’ of list sales. . ., but information can be collected and used without their permission. State laws impose some constraints.”

However, the consumer is not entirely powerless. 15 U.S.C. § 41-58 authorizes the FTC to enforce agreements that companies make with customers, among other things [3]. This means that, if an individual stipulates the terms of sharing his information with a company before a transaction—and the company agrees to comply—the agreement is binding. This law provides an impetus for privacy policies.

The power of consumers to negotiate the dissemination of their personal data forms the basis of privacy preference expression languages. Up to date, these languages have seen only limited use. Research up until now has focused on two kinds of privacy policy expression languages: either between a consumer and an enterprise or completely
within an enterprise. Limited research has been done on languages that might bridge the gap and allow people to share their information with enterprises or other people. In this thesis, I propose a protocol for a Private Information Management Agent (PIMA). This protocol allows people to share their information with other people or enterprises. Furthermore, the privacy preference information is bound to the data in such a way that it follows the data from enterprise to enterprise, person to person. The audit trail left by the PIMA protocol could ultimately allow PIMA users to track and control the distribution and use of their data as it is shared with third parties and beyond.

Another reason for privacy policies is that Americans really do want to be able to control the dissemination of their own private information [4]. In fact, they would like to be quite protective about their information. Unfortunately, the sheer volume of information that is traded on a daily basis makes it next to impossible for a human to personally manage each individual datum about himself on a daily basis. Many Web users simply give up and take a chance:

At the same time that they overwhelmingly express concern about their online privacy, American Internet users do a striking number of intimate and trusting things online. This is another aspect of how the context of privacy discussions is important to understand. More than two-thirds of Internet users who have serious privacy concerns have done at least one of the following online: purchase goods, make travel reservations, get health information, respond to email and instant messages from strangers, make friends and dates with people they have never met face-to-face, join support groups, place their calendars and address books online, and participate in online auctions. All of these online activities require some degree—and often a high degree—of personal disclosure.

In fact, there are a number of other interesting findings by the same report (which, it should be noted, is from 2001):

- "...the majority of Internet users do not know if or how they are being tracked."
• "86% of Internet users say that Internet companies should ask people for permission to use their personal information."

• "...users would appreciate more technological tools that would give them a sense of control, or at least transparency in letting them know what is happening to the pieces of their identity they are divulging as they move through Internet space."

Also, [5] found in a poll of Americans in the spring of 2000 that easily the biggest concern for Internet users is "Business and people you don’t know getting personal information about you and your family," with 84% expressing concern for their privacy. Computer hackers and viruses both ranked significantly lower.

It is clear that "Web users in the United States and around the world need more powerful technical tools to give users greater control over their online privacy relationships" [6].

Stanford Law School professor Lawrence Lessig argues that there are four ways to regulate technology, what he calls the "Four Modalities of Regulation" [7]. They are:

**Laws.** This is self-explanatory.

**Social Norms.** What society does or does not accept.

**Markets.** Markets influence with prices.

**Architecture.** How a system is built can dictate what can be done with it.

There are several notable areas of previous research that have attempted to work with one or more of the above modalities to protect private information. These will be discussed more extensively in Section 2. Some technologies rely solely on the architecture modality to secure data in a way that the architecture actually controls how the protected information is copied and distributed. This can be far more difficult to do than it may initially seem.

Ultimately, a malicious party has many options for stealing sensitive data:
...right now there are basically no controls on information in hardcopy, nor are there any real controls on what is sent out via e-mails or carried out in notebook computers, removable media, and the like.

So one may have the latest firewalls, intrusion detection systems, massive monitoring, and aggressive actions where INFOSEC policy violations occur, but what about the user who prints out a copy of the sensitive information and walks out the door with it? Some corporations or government agencies may have guards that check for sensitive documents going out the door. Nice try, but no way does that work. The guards do not know what is sensitive and what is not. Even if they did, they would have to read the entire document and make judgments based on what they read. A ridiculously impossible and unrealistic task. So, they look for caveats on the document. No problem. As a user, even if the document is caveated coming off the printer, one can tape over it and make a copy of the document or even cut off the caveated borders.

As for removable media such as CDs or floppies, one can just hide them in their pockets. No corporation or even government agency will be doing body searches. [8]

[9] lists examples of internal privacy compromises ranging from disclosure of usernames and passwords to Social Security information to lists of people with AIDS.

Another category of privacy research relies on negotiated agreements about the storage and use of protected information. This category includes PIMA and draws mostly on the legal modality. As stated above, this is an area that is generating much interest and could be crucial to the growth of the Web as a medium for sharing private, personal information. Privacy policies form the basis for my research.

Privacy policies can also draw on the social norms and market modalities. If it is not socially acceptable for a business to request information from a person without providing a policy for the use of that information, people will balk at sharing their information with these enterprises. This would put economic pressure on the enter-
prises to conform to these expectations. As was described earlier there is a demand from the public for control over their information. Unfortunately, as will be described in [10], the main language currently in existence for this purpose has yet to really take off.

Two notable directions within the privacy policy arena are the Platform for Privacy Preferences Project (P3P) [11] and the Enterprise Privacy Authentication Language (EPAL) [12]. P3P is a schema for organizations to express to consumers what the organization does with data it collects from the consumer. It was meant to be generic, and has limited options for expandability. EPAL, on the other hand, was designed to be extremely flexible for use in intra-enterprise applications, but it was not meant to be used across multiple enterprises.

What is needed is something to bridge the gap between EPAL and P3P. An ideal privacy policy scheme would have the flexibility of EPAL but be able to be used in multiple environments, like P3P. It would allow a person to set the circumstances under which he shares his information, and then follow that data as it is shared from enterprise to enterprise to enterprise. It is with these goals in mind that the Private Information Management Agent was designed.
Chapter 2

The Inadequacy of Current Technologies

Many technologies approach the issue of protecting data in various ways. This section discusses a few of the key approaches currently being developed.

2.1 Digital Rights Management

Digital Rights Management (DRM) is one way to contain the exposure of private information. DRM theoretically has the ability to technologically regulate the use and flow of private information by through features such as copy prevention. There are many forms of protecting content from unauthorized copying, which fall under the category of digital rights management. Microsoft defines its own DRM in this way:

DRM is a set of technologies content owners can use to protect their copyrights and stay in closer contact with their customers. In most instances, DRM is a system that encrypts digital media content and limits access to only those people who have acquired a proper license to play the content. That is, DRM is a technology that enables the secure distribution, promotion, and sale of digital media content on the Internet. [13]
That certainly sounds appealing, but DRM technologies have a history of problems. In [14], one copy-protection scheme for CDs was found to be defeatable with either a pen or a piece of tape, and in [15], another system failed to protect its CD content on Microsoft Windows machines if the shift key was held down as the CD was inserted. The latter method failed entirely to protect its digital payload on machines running other operating systems.

Poor attempts at DRM aside, there is another, glaring hole in the DRM idea. At some point, any data of value must be converted to a usable form. If it is an encrypted e-mail, it must be decrypted so a reader can read it. If it is a copy-protected song or movie, it must be in a clear form by the time it goes to the screen or speakers. At this point, the protections are gone, and the data is vulnerable to theft.

It seems as if the only way to avoid this is to implement DRM through closed-source operating systems and secure hardware. This is hardly an open, portable solution.

Even with a perfect DRM system, protection is not necessarily absolute. A person may not be able to copy a movie from a computer screen onto a piece of paper and walk away with it, but one could do exactly that with something as personal as a Social Security number. DRM is not prepared for this possibility.

Maybe if one cannot control the copying, one can at least deter it. Digital watermarking and digital fingerprinting are one approach that some have used to secure large data files such as movies and music. Watermarking and fingerprinting are related to steganography, which is a way of hiding information (such as ownership information) within other information. [20] points out two important problems here. Unfortunately for this approach, it essentially boils down to "security through obscurity", relying on the hope that no one will figure out exactly where the fingerprint or watermark is located in the data. The author pessimistically predicts that for any watermarking-type approach "The mechanisms for watermarking will eventually become public, and when they do, they can be reverse engineered and removed" from the data. The second problem is that watermarking, by itself, does not prevent data from being copied. The author says watermarking and fingerprints are "hardly
enough to be useful because digital property is so easy to alter, and watermarking
doesn’t prevent someone from altering the digital property.”

Watermarking and fingerprinting technologies are so ineffective they do not war-
rant inclusion in PIMA.

One approach to add a degree of limited protection to data would be to use en-
cryption and encrypt each datum with a unique key. Then, keys could be distributed
to various entities for access to the pieces of data that they might need. This ap-
proach has limited use in part because it is difficult to revoke keys to data once they
have been distributed. A new set of keys would need to be created for all parties,
an expensive process. In addition, distributing data to many more people than nec-
essary, even if the data is encrypted, can be detrimental to the long-term security
of the data. As the recent critical security hole (CVE CAN-2003-0971) in the GPG
program illustrates, one should never be too sure, even about encrypted data [16]. In
addition, it would be useful to know each time an entity requests data for which it
has permission, so it would be best to only distribute data as needed, not simply as
authorized.

A more fine-grained approach to managing data rights is through SAML and
XACML. The Security Access Markup Language (SAML) is an XML-based language
that ”provides a standard XML schema for specifying authentication, attribute, and
authorization decision statements, and it additionally specifies a Web services-based
request/reply protocol for exchanging these statements” [17]. In other words, SAML is
a language about authentication and describing decisions made about authorization.
Designed to work with SAML is the eXtensible Access Control Markup Language
(XACML), which ”describes both a policy language and an access control decision
request/response language (both written in XML)” [18]. The description of a generic
access control language like this is useful, but it overlaps with the work done by P3P
(described in section 2.2). While generic access control languages could be used for
privacy protection purposes, they are not specifically tailored for the task of protecting
individual privacy the way P3P is, so P3P would be a preferable language in this case.
However, one could imagine a national security setting, for example, where the generic
qualities of XACML would be more useful than the personal privacy-specific P3P.

2.2 P3P

The widely used privacy expression standard, created by the World Wide Web Consortium, is the Platform for Privacy Preferences Project (P3P) [11]. As the P3P Web site describes it, "P3P is a standardized set of multiple-choice questions, covering all the major aspects of a Web site's privacy policies. ... P3P-enabled Web sites make this information available in a standard, machine-readable format." An important aspect of P3P is the EXTENSION element described in the P3P specification. It allows P3P to be extended to include new syntax and semantics.

In P3P, a Web site publishes one or more policies describing what kind of information is collected and what is done with that information. The consumer then can make choices on whether or not to continue browsing that site, whether or not to accept cookies from that site, and whether or not to enter information into forms on that site. These choices empower the consumer. The policies that the Web sites publish are more than just empty promises. They do carry some of the weight of the law, since the United States Federal Trade Commission is authorized under 15 U.S.C. 41-58 to enforce these promises.

Examples of P3P files can be found in the P3P specification. Figure 2-1 is a fragment of one of the P3P example files.

This statement says that the user can opt-in to have his ID and password saved for easy access to his own information. The only recipients of the information are the original site publishing the policy and agents acting on its behalf (or entities on who's behalf it acts as an agent). The data will only be retained for the stated purpose. If the user decides he does not want his data on the server at a later date, the site must destroy the data. Elsewhere in its policy, it must state a more specific data destruction timetable.

P3P is used in some Internet browsers like Mozilla. In Mozilla, one can set the browser to automatically accept or reject cookies based on whether or not the site
We allow you to set a password so that you can access your own information.

Unfortunately, "There is no user base and no user demand" for P3P [10]. It is a complex protocol, and its deployment has been limited. [19] checked 5,856 Web sites in May 2003 and found P3P policies on only 538 of them. Of these, about one third had technical errors, 7% had critical errors, and 74 sites "violated the P3P specification by posting P3P compact policies without their corresponding full P3P policies." On the bright side, P3P use is increasing, especially for popular U.S. consumer Web sites. P3P is a useful tool that is catching on, but so far has found extremely limited such as in Web browsers.

Two particular problems are not solved by P3P. First, P3P does not easily scale beyond its basic vocabulary. Adding new terms to P3P requires the extension mechanism, which is probably not practical for people and enterprises wishing to have fine-grained control over data. Since P3P was meant for use by typical Web sites that collect normal kinds of data, its standard, basic vocabulary would not be sufficient for other uses.

Second, P3P does not offer the ability to monitor or easily control data once it is
in the hands of third parties. This is an important oversight in a time when data is so easily shared among so many parties.

2.3 EPAL

The Enterprise Privacy Authorization Language (EPAL) was designed at IBM to be a fine-grained alternative to P3P for use in an intra-enterprise environment [12]. EPAL is a very flexible system. It does not rely on any specific vocabulary; instead, it requires users to create their own vocabulary files in order to use it. This works well within a single enterprise, but does not work across enterprise borders, since different enterprises may use different vocabularies. It is not feasible to create one global vocabulary file, either. It would have to be enormous to cover every possible term that might ever be needed.

Despite these limitations, EPAL is a good starting point for PIMA. Its flexibility offers clues as to how to build a privacy language in a way that is not specific to any one vocabulary. The language was designed to be easy not only for computers to read, but also for humans. The structure of the language is relatively straightforward. PIMA springboards from EPAL and adds capabilities that allow it to be used across multiple enterprises. The PIMA approach goes beyond EPAL to allow entities to bind together privacy agreements in a graph, which allows data to easily flow through multiple enterprises while protecting privacy all along the way.

2.4 Passport

Microsoft’s Passport is a single sign-on service that allows a user to log in once and automatically be authenticated to multiple entities [20]. It is not a privacy preference expression language. However, it is related to PIMA because it is a system designed for sharing information across enterprises. Unfortunately, Passport is not a viable option for individuals to use to protect their own data, since it is a propriety system.
2.5 Liberty Alliance

Liberty Alliance Project is a group designed to be an “open” alternative to Microsoft’s Passport project [21]. The Liberty Alliance is primarily a single sign-on system like Passport. In addition, it has a system to allow users to select what organizations can see which data and allows users to associate a privacy policy with their data [22]. However, Liberty Alliance does not define its own privacy preference expression language.

2.6 Bridging the Gap

As we have seen above, there is a need for privacy preference languages. Unfortunately, the two languages that seem to have the most potential, EPAL and P3P, are still rather limited in their capabilities. A person sharing information with a company, which might share information with another company, is left with very little control over his data once it leaves his hands. The Personal Information Management Agent attempts to bridge the gap between the generic, but very universal, P3P and the intra-enterprise EPAL.

The problem that P3P and EPAL do not solve, but which lies between them: A fine-grained privacy policy architecture that can work between both inter- and intra-enterprise.
Chapter 3

Tools from the Semantic Web

The Semantic Web provides many of the tools used in the design of the Private Information Management Agent. The Semantic Web is built on open standards, which are meant to be widely used and almost universally accepted on the Web. These standards have been designed to be very flexible. PIMA just uses just parts of a few of the many Semantic Web tools in existence. One of the great things about the Semantic Web is that it was built in a way that encourages people to build their own tools on top of it. This is exactly how PIMA works.

These technologies are a double-edged sword. On one hand, they facilitate the sharing of knowledge and data in a way so efficient that they could make current privacy fears seem quaint. On the other hand, these same tools can be used to tame the information bonanza to assist people in keeping control of their personal data even as the Information Superhighway upgrades to what seems like the Information Autobahn.

3.1 XML

When the World Wide Web was first created through the HyperText Markup Language (HTML), it was primarily a way of linking together human-readable documents. As the Web exploded, people began to realize that it could do much more than store knowledge for humans; it could also be used to store and share knowledge for com-
puters. There is no single language that is universal to all computers, so the Semantic Web begins with the eXtensible Markup Language (XML) [23]. XML has a few basic properties that are important to understand.

First, all XML files are composed of elements called "tags," which look like `<tagname>`, where `tagname` would be replaced by the name of the tag. If one tag opens a section of an XML document, a corresponding tag closes that same section. It is of the form `</tagname>`. A single tag that represents both the beginning and end of a section looks like `<tagname />`. Tags can be nested within other tags to form a hierarchy. Tags must be opened and closed within their nested structure, so the code in figure 3-1 is not well-formed, while the code in figure 3-2 is a well-formed nesting.

Last, name collisions within the tags of a document are okay as long as one uses namespaces [24]. For example, if Company A uses `x` in a tag to mean one thing, and Company B uses `x` to mean something different, they can still both use their versions of `<x>` within a single document. First, they define the two namespaces at the beginning of the document. In this example, Company A might use the namespace “a” to refer to its terminology, and Company B could use “b.” Within the document, the qualified name `a:x` would refer to Company A’s “x,” and `b:x` would refer to Company B’s “x.”
3.2 RDF and OWL

XML is simply a framework for the Semantic Web, like letters form the framework for a sentence. Resource Description Framework (RDF) and Web Ontology Language (OWL) bring the true “meaning” into the picture [25] [26]. RDF is the grammar for the Semantic Web, providing a way to structure sentences. Each sentence consists of a subject, a predicate, and an object. For example, assume one wants to say that John’s age is 45. The subject of the statement is John. The predicate is “age,” and tells what the statement is trying to describe about the subject. The object is “45,” since it is the defines a value of the predicate “age.” It might be helpful to think of RDF statements in the form “subject has predicate with value object,” or in this case “John has an age with value 45.”

In RDF the “words” are mainly represented by Uniform Resource Identifiers (URIs) [27] and string literals. Uniform Resource Locators (URLs), such as http://web.mit.edu/index.html, are a subset of URIs. In the example about John, John could be represented by a URI (there are better ways of doing this, but this example aims to be simple), as would the word “age.” The URI representing “age” would link to the definition of that word presumably on another Web page. The format for defining RDF terms is RDF Schema [28].

Web Ontology Language (OWL) is another way of expressing meaning in the Semantic Web [26]. It is mainly a way to relate classes to each other. Example terms in OWL include “oneOf,” “disjointWith,” “unionOf,” “equivalentClass,” and “rdfs:subClassOf.” OWL also describes cardinalities of relationships.

RDF and OWL combined offer a rich language to describe the world. They form the backbone of PIMA, allowing it to describe difficult privacy concepts in a manner that computers can understand.
3.3 Friend of a Friend (FOAF)

Social networking sites are quickly gaining popularity. The basic concept is simple: allow people to share information about who their friends are. Then, when people want to meet other people, they can learn about their friends' friends or they can try to determine if the person they want to meet knows anyone who knows them. To operate, one needs to be able to share information about himself (e.g. name, career, e-mail address) as well as about his friends (at the very least, who are they?).

The FOAF project is an attempt to use RDF to define a language about people and their relationships [29]. It operates in a decentralized manner. Instead of build up a social network by storing everyone's data on one server, each user can have his own FOAF file, which can reference friends' FOAF files with a URI. This allows anyone to build his own FOAF search engine and browse through the network of relationships.

Figure 3-3 shows a simple FOAF file about Bob, who has two friends: Alice and Charlie. Note the namespace definitions with xmlns in the first tag of the file. Also, note the nesting of Alice's and Charlie's information within Bob's information.

In the example, Bob is represented by the first foaf:Person node. Within that node is his name, e-mail address, phone number, and more. In the RDF world, one could make sentences like “x has a name with value Bob Smith” and “x has a phone with value 617-555-1234,” where x is the person we call Bob.

As it is, FOAF does not have much privacy capability. To protect FOAF users from spammers, the FOAF spec offers a way to use hashes of e-mail addresses instead of the actual addresses. This has a few disadvantages. First, it obscures the e-mail address from the eyes of legitimate users. Second, the hashes could, in theory, be compiled into a database by a spammer. The database could then be checked against a brute-force cracker to find valid addresses in a manner more efficient than a spammer who tries to find new e-mail addresses by sending out e-mails to randomly generated addresses. Last, the hashes are only used for e-mail addresses in FOAF, leaving the rest of the data vulnerable.

While putting one's FOAF file out on the Internet in its entirety for all to view
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xmlns:foaf="http://xmlns.com/foaf/0.1/">
  <foaf:Person>
    <foaf:name>Bob Smith</foaf:name>
    <foaf:givenname>Bob</foaf:givenname>
    <foaf:family_name>Smith</foaf:family_name>
    <foaf:nick>bob</foaf:nick>
    <foaf:mbox rdf:resource="mailto:bob@example.com"/>
    <foaf:homepage rdf:resource="http://www.example.com"/>
    <foaf:phone rdf:resource="tel:617-555-1234"/>
    <foaf:knows>
      <foaf:Person>
        <foaf:name>Alice</foaf:name>
        <foaf:mbox rdf:resource="mailto:alice@example.com"/>
        <rdfs:seeAlso rdf:resource="http://example.com/alice/foaf.rdf"/>
      </foaf:Person>
      <foaf:knows>
        <foaf:Person>
          <foaf:name>Charlie</foaf:name>
          <foaf:mbox rdf:resource="mailto:charlie@example.com"/>
          <rdfs:seeAlso rdf:resource="http://example.com/charlie/foaf.rdf"/>
        </foaf:Person>
      </foaf:knows>
    </foaf:knows>
  </foaf:Person>
</rdf:RDF>

Figure 3-3: A simple FOAF file.
might facilitate search engines, it could also constitute a huge loss of privacy. Social networking needs to be able to limit the distribution of data to make users more comfortable. One should be able to share his phone number with another person but say, “Please don’t share this with anyone else.” In short, FOAF needs privacy protection capabilities.
Chapter 4

PIMA Design Approach

One of the main goals of the design for the Private Information Management Agent is to rely as much as possible on preexisting open standards. This would allow PIMA to run on as many platforms and in as many environments as possible. Relying on secure hardware or closed-source operating systems disenfranchises the many users of alternative operating systems and hardware. The strength in such a system is based in part on more users using it, not less.

4.1 PIMA and FOAF

When designing a privacy preference expression language, it is helpful to envision a basic environment in which that language is used. In the case of PIMA, the environment allows people to share personal information with each other and with organizations. It must be based on Semantic Web technologies. Furthermore, it should not restrict the sharing of data between entities. In fact, the Friend of a Friend (FOAF) language has a unique way of describing personal information that is particularly conducive to sharing personal data. FOAF was designed to enable social networking, allowing people to connect to other people via mutual contacts, and this aspect of its design lends itself well to PIMA’s graph-based approach to privacy policies.

FOAF is a great testbed for experimenting with privacy policies. It is build solidly on Semantic Web technologies, which form the next-generation Web. It is an open
technology, so it could be adopted by countless organizations. In addition, it is a language for depicting information that could be considered personal or private. It was important for me to work in a realistic environment like this.

In addition, FOAF offers something very special in its construction. Building on the idea of social networking, FOAF not only allows people to link to other people’s information, it also allows people to embed information about others directly in their own files. The difference between a Rolodex and a FOAF file is analogous to the difference between a group of related documents sitting in a pile and the same documents hyperlinked to each other on the Web.

In the example in section 3.3, one can see how Bob linked to Alice’s and Charlie’s separate FOAF files using rdfs:seeAlso with a URI and also how Bob actually included some of Alice’s and Charlie’s information, like their names, in his own FOAF file.

PIMA builds on top of FOAF to include privacy preference information as an attribute of a person or organization in the same way that a name is an attribute of a person in FOAF.

There is one important difference between FOAF and PIMA relationships. In FOAF, the tag knows refers to a relationship from one Person to another Person. knows does not imply friendship, although the specification notes that “social attitudes and conventions on this topic vary greatly between communities....”

For PIMA to work properly, it was necessary to describe relationships not just between Persons, but also between Persons and Organizations. In addition, it is helpful to not worry about the implications about the FOAF knows.

PIMA has its own version of knows. One might differentiate between the two versions in one file by referring to the FOAF knows as foaf:knows and the PIMA knows as pima:knows.

pima:knows describes any relationship between one Person or Organization and one other Person or Organization. This way, one can say that Bob “knows” his bank, grocery store, etc. without having to choose a person from that organization to describe the relationship. The following relationships hold:
With the change to knows, FOAF is a great tool for working with private, personal information.

### 4.2 Quick Introduction to XPath

One technology that has an important use in PIMA is XPath. As its specification says, “XPath is a language for addressing parts of an XML document...” [30]. XPath works by forming a tree structure for XML documents analogous to a file system tree. The root of the document is denoted by “/”. Each node descending deeper into the tree is appended to the end of the path. What follows are some example uses of XPath to address data in figure 3-3.

```
/rdf:RDF/foaf:Person
```

This selects the first foaf:Person node in the document’s tree—the one about Bob.

```
/rdf:RDF/foaf:Person/foaf:knows/foaf:Person/foaf:name
```

This selects the nodes representing the names of the people that Bob knows. It returns a list of those nodes. In this case, the list contains the nodes naming Alice and Charlie.

```
/rdf:RDF/foaf:Person/foaf:knows/foaf:Person[foaf:name="Alice"]/
foaf:mbox/@rdf:resource/text()
```
This XPath first finds the person that Bob knows who’s name is Alice and then locates the text value of the rdf:resource attribute of the e-mail for that person. In this case, the returned node would be the one that corresponds to the following attribute:

```
rdf:resource="mailto:alice@example.com"
```

```
/rdf:RDF/foaf:Person/pima:knows/foaf:Person[foaf:Person/foaf:name="Alice"]
```

This one returns a list of persons who Bob knows that know Alice. It begins by listing all the persons who Bob knows and then filters out the persons who do not know Alice. The remaining persons are the ones who know both Bob and Alice.

The great thing about XPath is how easy it is to refer to rather complicated pieces of data. For example, while is would be easy for most privacy preference languages to refer to “your name,” it would be difficult for many to refer to the concept of “the names of your friends.” With XPath, one can simply follow the above example, which does exactly that or use a relative XPath to address the names of the friends from the context of the person’s main FOAF node (usually at `/rdf:RDF/foaf:Person`):

```
foaf:knows/foaf:Person/foaf:name
```

In addition, this addressing of data allows users to be flexible with defining vocabularies. Unlike EPAL and P3P, special vocabulary files or extensions do not have to be defined. Using a person’s own name as an example, the tag `foaf:name` along with the value of the `foaf` namespace corresponds to a URI where the term is already defined. Users can mix and match vocabularies as they see fit, and do not need to worry about extra overhead.

### 4.3 PIMA and EPAL

EPAL was my first choice for integration with FOAF. It was meant to be a flexible system that could be useful in a variety of intra-enterprise environments, since it did
not rely on any specific vocabulary. Unfortunately, it was not perfectly suited for work with FOAF.

First, FOAF and EPAL have different concepts of hierarchies. In FOAF, OWL hierarchies are used. In this sense, FOAF describes data in terms of classes and attributes. The Person class can have a name attribute. The Person class is a subclass of the Agent class, since a Person is an Agent. This is a similar structure to the one seen in object oriented programming.

On the other hand, EPAL uses hierarchies to group related data into useful sets. As an example, consider a case in which there are datatypes of “Name,” “Age,” “Gender,” and “Birthplace.” Furthermore, assume that these datatypes are grouped under a datatype called “BiographicalInfo.” While it is possible to make the case that one could consider the subtypes to be properties of a “BiographicalInfo” class, it begins to stretch the usefulness of quality class hierarchies.

Another issue with EPAL was the need for vocabulary files. FOAF already has a preexisting definition of its terms in the form of an RDF Schema. To integrate with Semantic Web, it would be helpful for EPAL to not require writers of RDF Schemas to have to write separate EPAL vocabulary files. In addition, EPAL only allows for the use of one vocabulary at a time. This severely restricts its usefulness in inter-enterprise applications. Through the use of XML qualified names rather than ID references to refer to vocabulary terms, one can seamlessly integrate multiple vocabularies in a manner that avoids collisions. This is the path I chose.

One other departure from the EPAL specification that PIMA makes is the use of XPaths. In EPAL, a person might define a term in a vocabulary file called “friendsNames” to refer to the names of one’s friends. This is awkward for dealing with data that travels multiple hops from origin to destination (e.g. from Alice through Bob to Charlie). All the possibilities, such as even “friendsFriendsNames”—or worse—could be needed. XPaths elegantly avoid this problem. As described above, one could refer to “friendsNames” with the relative XPath:

\[ \text{pima:knows/foaf:Person/foaf:name} \]
and “friendsFriendsNames” with:

\[
pima:knows/foaf:Person/pima:knows/foaf:Person/foaf:name
\]

The possibilities are limitless. PIMA uses the basic idea of EPAL, along with its rule-handling procedures, but with changes such as the major ones described in this section.

### 4.4 XPath and PIMA

XPath has already been discussed twice, in sections 4.2 and 4.3. With the basics out of the way, a couple of other points on the use of XPath should be addressed.

First, although it may be apparent to the reader, it is worth noting explicitly that XPaths in FOAF contain some semantic information of their own since the structure of node trees in an XML document reveals information about the relationships of the nodes to each other. For example, a `foaf:name` node one level below a `foaf:Person` node describes the name of that person. It would be invalid (and very confusing) to put the names of one’s friends directly inside one’s own `foaf:Person` node describing himself. Instead, the names of one friends go inside their own `foaf:Person` nodes, which in turn go inside `pima:knows` nodes (or `foaf:knows`), which in turn go inside one’s own `foaf:Person` node. Then the XPath, in plain English says “This person knows another person who has a name.”

To be as generic as possible, PIMA uses relative XPaths when referring to data. So, when talking to Alice, the XPaths that are traded are relative to the `foaf:Person` node that represents Alice’s own FOAF file. When working with an organization, the XPaths are relative to that organization’s `foaf:Organization` node that represents its own FOAF file. In this way, an entity could generically ask each person for his `foaf:name` without having to know the exact structure of each person’s FOAF file.

Another useful capability of XPath lies in its ability to describe descendents arbitrary levels down a tree. For example, if one is working from the context of Alice’s `foaf:Person` node this relative XPath:
XPath first finds the node for the person named Bob who Alice knows. Then, XPath looks to see if there are any paths that exist from the person named Bob to the person named Charlie. The result would be a list of persons named Charlie who are within the Bob’s subtree in the file. So, if Bob knows Eve and Eve knows Charlie, this node about Charlie would be part of the result. Additionally, if Bob knows Tristan and Tristan knows Charlie, this node about Charlie would also be included in the list. Also, if Bob, himself, knows Charlie, this node would be part of the result, too.

These are three separate nodes about Charlie. They may be about the same person and have different information. For example, maybe Tristan thinks Charlie is 25 and Eve thinks he is 26. One of them must be wrong, but it is not up to PIMA to resolve the conflict. PIMA simply can say that different people say different things about the same person. If Alice trusts Eve over Tristan, she can choose to believe what Eve says. Either way, Alice has access to information about Charlie, and she can even see the provenance of that information.

XPath is not without some problems, however. For example, if two people are using different vocabulary files to refer to the same concept of “name,” they need to negotiate the equality of those terms. While PIMA does not need to specifically handle these issues, it could easily use the tools of RDF and OWL to relate terms to each other through mechanisms such as equivalence and sub- or super-classes. The Semantic Web already has the capabilities for dealing with these issues, and it would be natural for PIMA to incorporate those capabilities.

Also, PIMA’s use of XPath does not easily allow various sets of data to be grouped together. The hierarchical structure of FOAF limits the way data can be grouped to nodes on the same XPath. EPAL and P3P, on the other hand, can group datatypes into more random associations (e.g. one’s name together with one’s friend’s age). While this may result in a more verbose privacy language, importantly, it does not limit the capabilities PIMA has to describe data.
Ultimately, the use of XPath to describe data opens up a wide array of possibilities for working with FOAF files and dealing with information that has passed through many hands. This is critical to one of the key features of PIMA: its graph-based policy approach.

### 4.5 A Graph-Based Policy Approach

Graphs are what make the Web and social networking so powerful. They can do a lot for sharing data across the Web, too. On the Web, hyperlinks allow any number of documents to link to any number of other documents. Traversing this links is as easy as clicking a mouse. This allows people to easily “surf” (or in graph terms, “walk”) around groups of interrelated documents. In social networking, for example, a business person looking for a way to meet a prospective client might try to see if they have any mutual friends who could introduce the two. A social networking engine would walk the graph of friendships from the businessman to his friends and then see if it is possible to walk from those friends to the prospective client. If there are any paths that do this, those paths would be returned to the businessman.

Information sharing is ultimately built on some of the same concepts. Consider the case of Alice booking code-shared flights to another country (see figure 4-1). Alice might begin by calling her travel agent. The travel agent takes Alice’s information and finds a two-leg trip to a foreign country. The agent books the trip with one airline, which has a code-share agreement with a second airline to provide part of the travel for a fee. The travel agent shares Alice’s information with the first airline, which, in turn, shares the information with the second airline. Alice’s information quickly goes through several steps (“hops”) between herself and the second airline. It is even possible that the second airline already has some information about Alice from a third airline that Alice did code-share flying with in the past. The paths that Alice’s information follows forms a web of its own.

Previous privacy preference expression languages have focused on information traveling a distance of one hop (from one entity only as far as to a second). P3P does
address the issues of sharing information with third parties through its thirdparty data set; however, this is not easily extended to “fourth” parties or beyond and does not provide a built-in way for tracking data provenance and the privacy agreements that follow the data. While information about third party use of data could be worked into an EPAL vocabulary, it is not native to EPAL and suffers the same provenance and privacy tracking issues as P3P.

Instead, PIMA uses a graph-based approach that integrates it with FOAF. When Alice’s data travels through Bob on its way to Charlie, it accumulates privacy information on the way. The data is governed by the agreements negotiated between Alice and Bob and between Bob and Charlie. The data can then be shared ever farther out, as long as its transfer does not violate any of the agreements negotiated along the way.

In some cases, there might be multiple paths from the potential source of the data to the potential destination for it. In this case, there could be confusion if there is at least one path that would deny the transfer of the data and at least one path that would permit it. In the case, the data transfer can proceed as long as it is based on the privacy terms from one of the paths permitting the transfer. The reason for this is that there are almost always going to be non-permitted paths for transferring data. A trivial example would be from Alice to a spammer to Bob. Alice would almost certainly not allow this sort of transfer. To allow a denial to “veto” a transfer would make transfers next to impossible.

The graph-based approach facilitates complex information sharing with an elegant
system for documenting the sharing. This is one of the main contributions of PIMA.

4.5.1 Viral Agreements

In order for PIMA to make the graph-based approach work, it is important that all the users of PIMA agree that, whenever they pass along data covered by PIMA, they will pass it along only to other entities that also use PIMA. In this way, the PIMA privacy policies associated with data will travel with the data from one entity to another much like the “sticky” privacy policies proposed in [31]. This is also very similar to a viral licensing agreement.

Perhaps the most famous viral license is the GNU Public License (GPL) [32]. The GPL’s viral nature is a result of the following clause:

Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not impose any further restrictions on the recipients’ exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.

The first sentence in the clause ensures that the license follows the program everywhere it is distributed, even if the redistributed version is different from the original version. The second sentence proscribes the recipient of the program from watering down the open source nature of the program. This basically means that the recipient cannot change the license when he passes it on. The last sentence eases nerves.

In order for audit requirements and other privacy requirements to travel with data, PIMA data use must follow a viral license, too. The terms of the license could actually look very similar:

Each time you redistribute the protected data (or any work based on the protected data), the recipient automatically receives a license from you to copy, distribute or use the protected data subject to these terms and
conditions. You may not impose any fewer restrictions on the recipients’ exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.

Notice that rather than the rights of the recipient not being limited in the GPL, the PIMA license actually does the opposite, since it makes sense that, as data is passed along, only more restrictions—not fewer—should apply to the data protected by the agreements.

4.6 The Focus on Use of Data Rather Than Storage of It

The main focus of laws for privacy protection on the Internet has been on the collection of data. The use of that data has often been of secondary importance [33]. Many laws govern how information can be gathered but say little or nothing about what can be done with that information once it has been gathered.

P3P and EPAL do take a step in the right direction by making data usage part of their vocabularies. In P3P, for example, the policy generally helps the user determine whether or not he wants to allow the company to store some information about him for x or y purpose. In EPAL, the convention is similar. Each rule in the policy file has a mandatory purpose field, which describes the reason(s) that the data requester wants the data.

PIMA allows for a new use of privacy policies—making on-the-fly decision about whether or not to share data in real time with an entity multiple hops away. The data can be used in real-time and then immediately discarded. When the data is needed again, it can simply be re-requested. There is no need for data to be stored on remote servers, although it certainly could if that is what the user wants.

This is possible because PIMA has mechanisms specifically designed for data to travel through multiple entities from source to final destination. The privacy policies associated with the data are with it each step of the way. Rather than simply
acknowledging that data will be given to third parties, PIMA offers a way to track data to the third parties and determine who they are. Privacy policies can be set up on each of the involved parties’ servers in preparation for a data transfer, so that when a transaction occurs, it occurs quickly. PIMA’s language, like P3P and EPAL, allows the user to specify restrictions on the use of his information. The fine-grained nature of PIMA that allows the user to specify the terms of how his data can be used, combined with the way PIMA shares data easily across networks, gives PIMA users this ability to keep their personal information stored close to themselves.

Only giving out data when it is needed and then requiring the destruction of the data after its use affords a user new protections. In particular, users’ data would be more difficult to obtain in the execution of search warrants and subpoenas, since the data is not sitting in many locations. Users could also feel more secure about the likelihood that their data could be stolen off of a compromised, remote server belonging to someone they have a relationship with.

The cost of this is having do deal with network partitions. The data would need to be constantly available to be useful to many entities. This would mean that provisions would probably need to be made in many cases for data to be cached for critical applications. However, non-critical applications could simply wait until the data was available again before proceeding.

Ultimately, though, the possibility of transferring data across multiple hops without storing it provides a new level of protection that is difficult, if not impossible, for other privacy expression languages to provide. It allows users the option of going all the way to denying storage rights to others while focusing on how the others use private data.
Chapter 5

The PIMA Protocol

The PIMA protocol is broken into two distinct portions. The first phase is the initial request to establish a relationship. Follow-up requests actually retrieve the data requested. For the purposes of clarity, the term *entity* is used here to describe a foaf:Person or foaf:Organization. The source is the entity where the data is coming from. It could be either the person or organization whom the data is about, or it could be some other party that is holding on to the given data. The destination is the entity that will receive the data during the transmission. It is the party that requests the data.

Figure 5-1 displays an outline of the PIMA protocol. First, the destination requests an agreement with the source. The request contains information about what kind of information is requested, who will use it, what it will be used for, and why it is needed. The source evaluates this request. If the request is acceptable, the source sends a request reply to the destination. The request reply is mainly a set of rules to govern the use and distribution of the data based on the destination’s request and the source’s internal privacy rules. This establishes a relationship. Once established, a destination requests data by sending a transaction request to the source. The transaction request lists the data requested. If the source accepts, it sends back a transaction reply containing the requested data. Then, both the source and destination log the transaction. The log contains the time and date of the transaction as well the kind of data transferred.
5.1 Making Agreements

The first thing that occurs in PIMA is the establishment of a relationship. This begins with the potential destination of data sending a request to the source of the data. This request asks the source for permission to access some of the source's data.

The request follows the general intent of the EPAL specification. It consists of:

- A non-empty set of user-categories
- A non-empty set of data-categories
- A non-empty set of purposes
- A non-empty set of actions
- A container provider

5.1.1 The user-category, purpose, and action Elements

The user-categories are URIs (probably in the form of qualified names) that describe the kinds of entities that will be using the requested data. For example, assume that
some basic terms are defined in an RDFS file stored at a third-party location. In this example, “salesURI” refers to the URI describing what a sales department is. In PIMA, a user-category is listed as:

<pima:user-category rdf:resource="salesURI" />

where the pima namespace is equivalent to http://purl.org/pima/. Multiple user categories are simply listed one after the other. The attribute rdf:resource is a mandatory attribute with a URI to an RDFS definition of the user-category being described.

The purposes and actions work similarly, with tags named pima:purpose and pima:action. Purposes describe why the data is needed. Order-processing, shipping, or IRS auditing could all be valid reasons for requesting data. Actions describe what will be done with the data, such as storing it, selling it, or performing market analysis. These terms are also defined in RDFS files.

5.1.2 The data-category Element

Data categories are slightly trickier. Since they are not URIs, but XPaths, they are not described through the use of an rdf:resource attribute. Instead, the relative XPath is the value of the mandatory pima:datapath attribute. The XPath is relative to the primary foaf:Person or foaf:Organization node describing the source. The element corresponding to the names of an entity’s friends would look like:

<pima:data-category pima:datapath="pima:knows/foaf:Person/foaf:name" />

A request for this data would return any information at or beneath the nodes that are returned during the evaluation of the XPath. If one requested

<pima:data-category pima:datapath="pima:knows/foaf:Person" />

they are requesting to know everything the source knows about all of the people he knows. Most people would probably be uncomfortable sharing this much information.
<rdf:RDF
 xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
 xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
 xmlns:foaf="http://xmlns.com/foaf/0.1/"
 xmlns:pima="http://purl.org/pima/">
 <!-- PIMA Request -->
 <pima:Request pima:source="aliceURI" pima:dest="exampleComURI">
   <pima:user-category rdf:resource="salesURI" />
   <pima:data-category pima:datapath="pima:knows/foaf:Person/foaf:name" />
   <pima:purpose rdf:resource="order-processingURI" />
   <pima:action rdf:resource="storeURI" />
 </pima:Request>
</rdf:RDF>

Figure 5-2: A simple PIMA request to establish a relationship.

5.1.3 EPAL Conditions and Containers

While conditions and containers would be important to a real-world implementation of PIMA, they were not addressed in detail in this project. They are not critical to demonstrate the main concepts behind PIMA, and do add a layer of complexity. However, it is important to understand what they represent.

A condition is a way of saying “allow this only if x is true.” One possibility would be a query to see if the destination attests that its privacy policy is certified by some third party. During the request stage, the destination would have to provide this information. It would be contained in a data structure EPAL refers to as a “container.” The information in the container would be evaluated against the source’s conditions during the source’s rule evaluation to see if the conditions are satisfied by the data in the container.

5.1.4 The Request Element

A simple request, with only one user-category, data-category, purpose, and action, might look like the file in figure 5-2.

In a real request, the attribute values that are shown here ending with “URI”
would be replaced by real URIs referring to the RDFS definitions of those terms.

A request must have at least one user-category, one data-category, one purpose, and one action.

The Request element has a mandatory source attribute and dest attribute. The source refers to the entity that the data is coming from, and the destination refers to the entity that the data is going to. If data goes from A to B to C (i.e. A to C via B), then there will be one request with A as source and B as destination and one request with B as source and C as destination.

### 5.1.5 The Privacy Policy

When the request to establish a relationship is received by the source, it must be evaluated against the source’s privacy policy. The PIMA privacy policy is also based in large part on EPAL policies.

**The pima-info Element**

Any entity with a privacy policy places his privacy information within its FOAF node describing itself. A PIMA privacy information node describes a unidirectional relationship. The node has optional “source” and “destination” attributes. One of the attributes must be present for the node to be well-formed. Each tag can have at most one source and one destination attribute. The tag associated with the node looks like:

```
<pima:pima-info pima:source="AliceURI" pima:dest="BobURI">  
```

This would contain metadata about the data shared by Alice with Bob. The source and destination attributes are URIs.

For destinations, when the source attribute is missing, the PIMA information node could describe the default behavior of how Bob (in this case) reacts as a destination for information with any source. However, once a relationship is established, the information on that relationship should be stored in a PIMA information node that explicitly states the source and destination.
For sources, when the destination attribute is missing, the PIMA information node could describe the default behavior of how Alice (in this case) reacts as a source for information given to any destination. As before, whenever possible, metadata should be put in a node explicitly stating the source and destination.

The PrivacyPolicy Element

Within the PIMA information node is the privacy policy, itself. The privacy policy tag looks like:

<pima:PrivacyPolicy default-ruling="allow">

where the mandatory default-ruling attribute can be either "allow" or "deny." The default-ruling tells the rule evaluator how it should rule if no rules are applicable to the request. "deny" is probably the only safe, reasonable choice for this value.

The Rule Element

Within the privacy policy are rules. Each rule begins with something like:

<pima:Rule id="ruleID" pima:ruling="allow">

The mandatory id attribute is for compatibility with EPAL. The mandatory ruling attribute’s value is, again, either "allow" or "deny." It represents what the ruling should be if the rule fires (e.g. “allow if…”).

Within the rule are the user-category, data-category, purpose, and action tags just like in the request. There must be at least one of each of these categories.

An example of a rule is shown in figure 5-3.

A Simple Privacy Policy

A series of rules listed one after the other forms the basis of a privacy policy, as shown in figure 5-4.
5.1.6 Request Evaluation

Rule evaluation proceeds similarly to EPAL rule evaluation. Perhaps the biggest difference is how PIMA and EPAL deal with data hierarchies ("scope" in EPAL), which were discuss in section 4.3. EPAL looks in a vocabulary file to see if one datatype is a "child" of another. PIMA uses OWL to see if one term is a subclass of another. In the case of data-categories, PIMA simply checks to see if one node is a descendent of the other.

Like EPAL, PIMA gives precedence to rules based on the order in which they appear. Earlier rules have higher precedence. For each rule, until the first rule fires, the algorithm first ensures that elements of the request are in the scope of the rule; then it makes sure that the conditions are satisfied. At least, this is how things operate for "simple requests" in which there is only one user-category, data-category, purpose, and action.

Otherwise, "compound request" evaluation proceeds. For more details on com-
pound request evaluation, see Appendix A.

In some cases, data may pass through multiple hands. In this case, the result of processing the request against the policies for each of those "hands" must result in an allow. Consider the case of Bob sharing information he has about Alice with Charlie. Charlie's request to Bob for the information about Alice is evaluated against not only Bob's personal preferences but also the preferences Bob agreed to with Alice. Only if both of these result in "allow" rulings can Bob share Alice's information with Charlie.

At this point, it might be necessary to get input from the user of PIMA. If someone is considering sharing his own data, he should be notified about a request for information. The person would be given a recommendation of what to do based on the evaluation of the request. Ultimately, the user would decide whether or not to accept sharing his information. The user should be given this option in order to allow him to override the PIMA system and also to make sure he is notified about the distribution of his data. It would be a scary prospect to have a program running in the background that could give away very personal information without one's full awareness.

5.1.7 Request Response

After deciding on the request, the source needs to reply to the potential destination. If the request was denied, the reply would contain little more than this fact. If the request was approved, the reply would contain a condensed privacy policy. The privacy policy would be a rule based on the request. It would have the user-categories, data-categories, purposes, and actions from the request, as well as any conditions and obligations found to be relevant during the policy evaluation phase.

If the data is traveling through multiple hands, then a separate rule is returned for each privacy policy that the request was evaluated against. The structure of the data is reassembled in the destination file in a way similar to how it was found in the source file.

If Bob had information about Alice that he was sharing with Charlie, Bob's FOAF file would look like figure 5-5.
When Charlie receives the response from Bob, he builds his FOAF file like that in figure 5-6.

Alice’s information, including her agreement with Bob, are nested inside Bob’s information, which, along with his agreement with Charlie, is nested inside Charlie’s information. In the original file, Bob only has generic privacy policies listed for himself (as the source or as the destination), but once the agreement is made between Bob and Charlie, his FOAF file changes to include the new agreement as is shown in figure 5-7.

Since the privacy information covers the release of Bob’s information, it goes directly inside Bob’s personal FOAF node. It would be incorrect to add Charlie as a new person who Bob knows and then put that privacy information inside the node about Charlie, since the Charlie’s information is not subject to release by this agreement. Also, the new privacy information node is placed before the other, more generic privacy information nodes since it is more specific and the nodes are checked in order to find a match. If the new node was placed after the already existing privacy information nodes, the node with Bob as the source and no destination would be
<foaf:Person>
    <pima:pima-information source="CharlieURI">
        ...
    </pima:pima-information>
    <pima:pima-information dest="CharlieURI">
        ...
    </pima:pima-information>
    <foaf:name>Charlie</foaf:name>
    <pima:knows>
        <foaf:Person>
            <pima:pima-information source="BobURI" dest="CharlieURI">
                ...
            </pima:pima-information>
            <foaf:name>Bob</foaf:name>
            <pima:knows>
                <foaf:Person>
                    <pima:pima-information source="AliceURI" dest="BobURI">
                        ...
                    </pima:pima-information>
                    <foaf:name>Alice</foaf:name>
                </foaf:Person>
            </pima:knows>
        </foaf:Person>
    </pima:knows>
</foaf:Person>

Figure 5-6: Charlie’s FOAF file with Alice’s data from Bob.
Bob modifies his FOAF file when he has finished evaluating Charlie’s request, and only if Bob agrees to the request. Charlie modifies his FOAF file after receiving an accepting reply from Bob.

The RequestReply Element

In the case of a denial, Bob’s response would look like figure 5-8.

The absence of any rules allowing Charlie access to Bob’s information indicates a denial. The RequestReply element has mandatory source and destination attributes, which are URIs.

On the other hand, an acceptance would look similar to figure 5-9.

The reply contains the result of Bob evaluating the request against both his personal privacy preferences and against his agreement with Alice. The reply could indicate either full or partial acceptance of Charlie’s request. Charlie includes the
<rdf:RDF
   xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
   xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
   xmlns:foaf="http://xmlns.com/foaf/0.1/"
   xmlns:pima="http://purl.org/pima/">
   <pima:RequestReply pima:source="BobURI" pima:dest="CharlieURI">
   </pima:RequestReply>
</rdf:RDF>

Figure 5-8: Bob's denial to Charlie's request.

<rdf:RDF
   xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
   xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
   xmlns:foaf="http://xmlns.com/foaf/0.1/"
   xmlns:pima="http://purl.org/pima/">
   <pima:RequestReply pima:source="BobURI" pima:dest="CharlieURI">
   <pima:pima-information source="BobURI" dest="CharlieURI">
   ...
   </pima:pima-information>
   <pima:pima-information source="AliceURI" dest="BobURI">
   ...
   </pima:pima-information>
   </pima:RequestReply>
</rdf:RDF>

Figure 5-9: Bob's acceptance of Charlie's request.
metadata in the request reply in his FOAF file, as was shown in figure 5-6.

Sharing Data Without Storing It

It is an important caveat to note that not all data requested is stored in FOAF files. Sometimes data may be requested for a single use and then immediately discarded. This prevents information from getting far away from its creator’s control. In this case, privacy agreements would be put into the FOAF files, but not the requested data, itself. If Charlie asks Bob for information about Alice, in this case, Bob would see that he has access to the information, but does not store it, so he would have to request the information from Alice through his existing privacy agreement with her. Once he receives the data from her, he would transfer in on to Charlie and then immediately discard the data he requested on Charlie’s behalf.

5.1.8 Data Exchange

Once the agreement is in place, the information transfer can begin. The request for access and the actual exchange are broken into two separate phases because an entity may wish to have access to information even though it does not have an immediate need for it. Also, an entity may wish to check for updates to information it already has without going through the overhead of renegotiating an agreement each time.

The TransReq Element

A transaction request from Charlie to Bob might look like figure 5-10.

This transaction request is for Bob’s name and the e-mail address of Bob’s friend Alice. When Bob receives the request, he first double-checks to make sure that it is still valid against all the relevant agreements he has on file with third parties (in this case, Alice) and against an agreement explicitly between himself and Charlie. If no, explicit agreement still exists between Bob and Charlie, Bob sends back an empty set of data. If the request is within the agreements, then Bob sends back the information he has in the format shown in figure 5-11.

55
Figure 5-10: A transaction request from Charlie to Bob.

The transaction request has mandatory source and destination attributes, with URI values. It can contain only data-category elements within itself.

The TransReply Element

The reply to a transaction contains mandatory source and destination attributes, represented by URIs. The TransReply element contains a list of at least one data-category element. The data-category element contains between its opening and closing tags the value of the data being referred to by the datapath attribute. This is the only time in PIMA that data-category elements have separate opening and closing tags.

If Charlie is going to store the returned values locally, Charlie must place them into their proper FOAF file locations as indicated by the corresponding datapaths.

5.1.9 Logging Transaction Histories

After a transfer, each party must log the transaction directly inside the corresponding PIMA privacy information node.

The transaction Element

A transaction log looks like:
Figure 5-11: A transaction reply from Bob to Charlie.

so a PIMA privacy information node would ultimately look like that shown in figure 5-12.

The transaction element contains a mandatory datapath attribute with an XPath value referring to the shared data. It also contains a dateTime element showing the time the transaction took place. It does not contain the values of the shared data.

5.1.10 Auditing through Log Analysis

The audit trails made by transaction nodes create a dual-entry bookkeeping style similar to those used by accountants. Every transaction results in an entry being made in two different “ledgers,” the source’s record and the destination’s record. During an audit, these records can be compared to see if they are consistent. Inconsistent records are indications of either faulty record keeping or deception. [34] gives a good analysis of the rational for double-entry bookkeeping:

This system had two main purposes. The two books would be kept by two different clerks, reducing the possibility of fraud. But more importantly,
the two books would be routinely balanced against each other.... This balancing process was an audit: If one clerk tried to commit fraud—or simply made a mistake—it would be caught in the balancing process, because someone other than the clerk would be checking the work....

These are not preventive security measures (although they may dissuade attacks); audit is designed to aid forensics. Audit is there so that you can detect a successful attack, figure out what happened after the fact, and then prove it in court.

An audit could be instigated by a trusted third party, such as a government agency or a privacy rights organization. In addition, it would be possible to extend the PIMA protocol to allow two entities to compare their records they have of transfers between themselves, and—more importantly—to allow the people to monitor the transfer and use of their data multiple steps away.

This is perhaps one of the other potential contributions of PIMA. Transparent use of PIMA would allow Alice, for example, to follow her own records to see who she shared data directly with. She could then query those people for information about the use and transfer of her data. They could check their transaction logs, and
return to Alice a list of entities they shared her data with. In the end, Alice could theoretically find out everywhere her data has gone by following transaction logs.

Auditing logs does not find every problem. If two logs disagree, it could be difficult to tell which one is correct. In addition, if Bob does not tell Alice he shared her data with Charlie, Alice would not notice a problem until she asked Charlie if he has any of her data. Even then, if Bob and Charlie collude, Alice would not know about her data being transferred between the two of them. However, logs make it more difficult to cheat, and if a regulatory agency wanted to do an investigation, it might notice a problem if it see large amounts of personal data kept with suspiciously little or no transaction logging.

5.2 Revoking Agreements

If an entity wants to revoke a set of privacy preferences, it could simply delete that agreement from its FOAF file. For example, Alice could decide she does not want her data shared with third parties, so she would change her privacy policy to reflect that. However, if Bob already has Alice’s data under an agreement that allows him to share her data with third parties, he is entitled to share that data. Not allowing him to do so could constitute a breach of contract. If Alice wants to limit the possibilities that she could end up in a situation like this, she could have her privacy policy reflect the fact that she does not want anyone to store her data. That way, any time a person wants her data, the request has to come all the way to her. Any changes to her privacy preferences would have an instantaneous effect.
Chapter 6

The PIMA Implementation

To make sure that the PIMA design was on the right track, I implemented a portion of it. I chose to implement PIMA using the Python programming language [35]. Python is an easy-to-use language with native support for object oriented programming. It is useful as a scripting language, also, and can be run line by line through an interpreter. Python has libraries that support XML parsing. I used the PyXML package of libraries [36] for working with Document Object Model (DOM) and XPath [37].

6.1 General Architecture

In general, I divided the architecture into two main parts: parsing and processing. The parsers handle reading in XML and are responsible for making sure the XML is well-formed. The processing is the back end that manipulates the objects formed as a result of the parsing.

To reduce memory, parsing a file does not automatically structure the file’s data into a large collection of objects. Instead, the parsers only generate objects as needed by the processing part of the code. This is accomplished by having parsers return other parsers. For example, if the main parser for a document is called mp, then the parser for the main entity (a person or organization) is found at mp.mainEntity(). This does not build an object reflecting the whole document. Instead, it simply readies a different parser to handle parsing one level farther down in the document.
The next step might be to find the PIMA information node. The parser for this node would be at `mp.mainEntity().pimaInfo(i)`, where `i` determines which PIMA information node is being referred to. Again, only parser objects are built, but not processor objects representing irrelevant portions of the document.

### 6.2 The Parsing Interface

The parsing interface tries to be consistent from one parser to the next. Each parser has a document node attribute, representing the DOM document node for the XML file, and a current node attribute, representing the node in the DOM representation for which the parser instance is responsible. During the instantiation of an object, the `__init__` method checks to make sure the node is well-formed.

Methods meant for public use inside the parser generally return either other parsers or processor objects. For example, the PrivacyPolicyParser is responsible for PIMA privacy policy nodes. It can return either RuleParser objects or a Policy object. The RuleParser is a parser for rules, and a Policy is a representation of a privacy policy used by the processing section of the code.

### 6.3 The Rule Processor

The rule processing section of the code follows the EPAL specification for processing rules. It was coded to not be specific to PIMA.

Its major components are a Rule class, Request class, Ruling class, CompoundRuling class, and a Policy class.

The Rule class represents an EPAL rule. It has attributes representing and ID, a ruling (of "allow" or "deny"), user-categories, actions, data-categories, and purposes. Mainly it acts as a container of these attributes.

The Request class represents an EPAL request. Its attributes include user-categories, data-categories, purposes, and actions. Like a Rule, it is primarily a container.

When rules are evaluated in EPAL, the result includes a ruling of "allow," "deny,"
or “not-applicable,” along with a set of obligations (which are not implemented here) and a list of the rules involved in the decision. To keep the basic structure similar to EPAL, a Ruling class contains this same data.

A CompoundRuling is similar to a Ruling class, but it stores data for multiple Rulings encountered during evaluation of a compound request.

The Policy class is where most of the processing occurs. A Policy contains a list of Rules that make up the privacy policy. It is responsible for evaluating requests. It begins by breaking up requests into smaller pieces. A multiple user, compound request is broken into a series of single user, compound requests. These are, in turn, broken into simple requests, which are finally evaluated against the rules. The results of each simple request evaluation is a Ruling, which is collected into a CompoundRuling, the final result of evaluation.

During the evaluation of simple requests, PIMA should, in theory, check the “scopes” of the elements of the request. The current implementation of PIMA does not check scopes; it simply looks for equality. As a proof of concept, this is not necessary. However, checking scopes of elements is straightforward. In the case of user-categories, purposes, and actions, OWL would be used to verify if one thing is a subclass of another. In the case of data-categories, the XPaths would be compared to see if one node is a descendant of the other node.

6.4 The PIMA Class

The PIMA class is what integrates the parsers and the processors. It has as an attribute the main FOAF file for which it is responsible. When input comes into the PIMA class, it first determines what the input is trying to accomplish (e.g. is it a request?). Depending on the input, it is passed to a handler which calls the necessary parsers to parse through the input. The resulting object is then passed through the corresponding processor code by the input’s handler method.
Chapter 7

Results

The implementation of PIMA was built to do the first step of negotiating an agreement. It was not meant to be fast and did not have to fall within any specific resource-usage constraints. It was simply to demonstrate the basic concept behind PIMA and to aid in the development of the PIMA protocol.

The ideas contained in the PIMA protocol were the main focus of this research. The PIMA protocol has several useful features:

- Native logging of transactions.
- Graph-based privacy preference evaluation.
- Ability to shift focus completely from regulating the storage of data to regulating the use of it.
- Based on open standards.

These features make PIMA a very unique protocol for expressing privacy preferences. Users can easily and seamlessly share data across multiple entities. They have the ability to monitor the use of their data, while completely denying its storage. These new features should be a part of future privacy preference expression research.

When comparing PIMA to P3P and EPAL, one can best see the contributions of PIMA to the dialogue on privacy policy expression languages:
Vocabulary  
• P3P uses a base data schema for most of its data. Extensions to the basic vocabulary can be made through an extension mechanism.

• EPAL does not have a base data schema. Instead, it relies on its users to agree to a vocabulary and reference it from the privacy policy.

• PIMA also does not have a base data schema. It uses RDF and RDFS to allow users to use multiple vocabularies from within the same privacy policy. Data is referred to by XPaths, which allow the user to reference data that is nested.

Logging  
• P3P does not utilize logging.

• EPAL does not utilize logging.

• PIMA has a built-in logging system that records the time and type of transactions.

Sharing Data Across Multiple Entities  
• P3P could be used to share data across multiple entities, in theory. Its data schema does refer third parties. However, it would be difficult to trace the provenance of data shared through P3P, and the P3P vocabulary does not have terms that represent concepts like “the name of your friend.”

• EPAL could also be used across multiple entities, in theory. However, it was designed for use in single-enterprise environments. EPAL also requires users to agree on one vocabulary. It would be difficult to get many users to agree to just one vocabulary. EPAL also would need to separately define terms like “friend’s name” and “friend’s friend’s name,” which could become cumbersome.

• PIMA was designed for exactly this task. PIMA nests data about people in an intuitive way that reflects the complex relationships that people have. Because PIMA uses XPath rather than a special vocabulary for referring to various pieces of data, it does not need any special extensions or terms to refer to “friend’s friend’s name” and similar concepts.
Chapter 8

Future Work

There is much more that can be done to advance PIMA and related research.

First, PIMA should be extended to handle EPAL conditions and obligations. It is essential for people to be able to say, “You can only use this information if....” As part of handling conditions, PIMA needs to be able to use EPAL containers, the elements that contain data to be evaluated against the conditions.

RDF and OWL usage should be expanded much more. RDF and OWL can be used to relate different terms in different vocabularies. They can be used to “translate” between vocabularies. In addition, OWL needs to be introduced into the scope-checking described in section 6.3.

It would also be helpful to incorporate ways of dealing with old privacy policies. Ideas include adding version numbers, nonces, or timestamps. This would allow an entity to change privacy policies while keeping the old agreements based on the previous version of the privacy policy.

Perhaps most importantly, digital signatures are an absolutely necessary piece of privacy agreements. It defeats the purpose of the auditing to have a malicious user change the terms of the agreement he made with someone else by simply rewriting that section of his FOAF file. It would be difficult to tell if the malicious user or the other person supposedly involved in the agreement was the culprit. Digital signatures would mitigate this problem. They should be part of the record inside each privacy agreement.
Last, an auditing protocol would be a very useful addition to PIMA. In its current form, PIMA logs information about transactions. It would be helpful to allow entities with legitimate purposes to see transaction histories. This would make data sharing completely transparent to all those involved. It would be a giant step forward from the current status quo of giving out data and not being able to see what happens to it.
Chapter 9

Conclusion

The Private Information Management Agent represents several steps forward for the expression of privacy preferences. PIMA incorporates the power of social networking and the Web along with Semantic Web technologies to offer a highly flexible, powerful protocol for sharing and distributing privacy preferences. The control over data that PIMA gives back to users helps to level the playing field at a time when the awesome power of the Web and computer databases threatens to take people’s control over their own privacy away from them. Now users have a way to not only dictate the terms under which they share information about themselves (and others), they can follow the spread of their information and audit its use. PIMA has the potential to be an important new tool in privacy protection.
Appendix A

Compound Request Evaluation

The following is from the EPAL specification:

Processing a Single user category: We now define how to process a compound request for one user category and multiple other elements. The compound query for a particular user-category is decomposed into simple requests as follows:

1. The algorithm creates two empty sets of obligation/rule-id-set pairs that will be used to collect all distinct obligations and the set of rule-id’s that mandated it. We call these sets denyObligations and allowObligations. In addition, it creates two sets denyRules and allowRules of rule-id’s that will be used to collect the rule-ids that lead to a ‘deny’ or ‘allow’ decision, respectively.

2. The algorithm performs one simple request for the given user-category and any combination of the other elements in the compound request. It uses the same attribute/container instance for all these simple requests.

If the ruling is ”allow”, the returned rule-id is added to the set allowRules. If obligations are returned, the (rule-id, obligation)-pairs are added to the set allowObligations. If the ruling is ”deny”, the
rule-id is added to the set denyRules while (rule-id, obligation)-pairs are added to the denyObligations set.

After processing the simple requests for all combinations of elements in the compound request, we can determine the ruling for this user category as follows:

- If all rulings are 'not-applicable', the ruling is 'not-applicable' with an empty rule-id-set.
- Else, if at least one ruling is 'allow' and no ruling is 'deny', the algorithm returns "allow", the allowRules rule-id-set, and the allowObligations set.
- Else, (if there is one or more 'deny' rulings), the algorithm returns 'deny', the denyRules rule-id-set, and the denyObligations.

Processing Multiple user categories: Multiple user categories are processed in the order that they are defined inside the vocabulary of the policy:

1. If the compound processing for any user category is 'allow', select the first one of such user categories and return the 'allow' ruling with the rule-id’s (allowRules) and obligation/rule-id-set pairs (allowObligations) that are returned for this user category.
2. If the compound processing for any user category is 'deny', select the first one of such user categories and return the 'deny' ruling with the rule-id’s (denyRules) and obligation/rule-id-set pairs (denyObligations) that are returned for this user category.
3. Else, return the 'not-applicable' ruling with an empty rule-id-set.
Appendix B

PIMA Code Sample

```python
from xml.dom.ext import PrettyPrint
from xml.dom.ext.reader import Sax2
from xml.xpath import Evaluate, Context
import os

from Ft.Xml.Xslt import Processor
from Ft.Xml import InputSource

#Globals:

nsBindings = {
    'foaf': 'http://xmlns.com/foaf/0.1/',
    'rdf': 'http://www.w3.org/1999/02/22-rdf-syntax-ns#',
    'rdfs': 'http://www.w3.org/2000/01/rdf-schema#',
    'owl': 'http://www.w3.org/2002/07/owl#',
    'vs': 'http://www.w3.org/2003/06/sw-vocab-status/ns#',
    'wot': 'http://xmlns.com/wot/0.1/',
    'dc': 'http://purl.org/dc/elements/1.1/',
    'pima': 'http://purl.org/pima/'
}
```
# I chose to keep parsing separate from the processing
# to enhance modularity

# Parsing

class NodeParseError(Exception):
    "Node is not parseable"
    pass

# Abstract class

class Parser:
    "Top level, abstract Parser class"
    def __init__(self,docNode,currentNode):
        self.doc = docNode # doc node
        self.current = currentNode # current node in context

    # The context should be shored up to accurately
    # reflect the namespaces the document uses
    def findPath(self,pathName):
        "Uses XPath to return an array of nodes at pathName"
        context = Context.Context(self.current,1,1,processorNss=nsBindings)
        results = Evaluate(pathName,self.doc,context)
        return results

class PrivacyCategoryParser(Parser):
    "Abstract class for parsing Rules and Requests"
def __init__(self, docNode, currentNode):
    Parser.__init__(self, docNode, currentNode)

def userCategory(self, i):
    "Returns the node representing the ith user category"
    uc = self.findPath("pima:user-category[%i]/
                      @rdf:resource" % i)
    if len(uc) == 0:
        return None
    else:
        # check data here
        return uc[0]

def userCategoryURI(self, i):
    "Returns the string URI representing the ith user category"
    uc = self.userCategory(i)
    if uc == None:
        return None
    else:
        return uc.nodeValue

def userList(self):
    "Returns a list of string URIs representing all of the
    user categories in the subdocument"
    i = 1
    ucs = []
    while 1:
        ucUri = self.userCategoryURI(i)
        if ucUri == None:
            break
ucs.append(ucUri)
i = i + 1
return ucs

def dataCategory(self,i):
    # Note: relative paths only?
dc = self.findPath("pima:data-category[%i]/
@pima:datapath" % i)
if len(dc) == 0:
    return None
else:
    # check data here
    return dc[0]

def dataCategoryPath(self,i):
    # Note: relative paths only?
dc = self.dataCategory(i)
if dc == None:
    return None
else:
    return dc.nodeValue

def dataList(self):
    i = 1
dcs = []
while 1:
dcPath = self.dataCategoryPath(i)
if dcPath == None:
    break
    dcs.append(dcPath)
i = i + 1
return dcs

def purpose(self,i):
p = self.findPath("pima:purpose[%i]/@rdf:resource" % i)
if len(p) == 0:
    return None
else:
    #check data here
    return p[0]

def purposeURI(self,i):
p = self.purpose(i)
if p == None:
    return None
else:
    return p.nodeValue

def purposeList(self):
i = 1
ps = []
while 1:
pUri = self.purposeURI(i)
if pUri == None:
    break
ps.append(pUri)
i = i + 1
return ps

def action(self,i):
a = self.findPath("pima:action[%i]/@rdf:resource" % i)
if len(a) == 0:
    return None
else:
    #check data here
    return a[0]

def actionURI(self,i):
a = self.action(i)
if a == None:
    return None
else:
    return a.nodeValue

def actionList(self):
i = 1
as = []
while 1:
aUri = self.actionURI(i)
if aUri == None:
    break
as.append(aUri)
i = i + 1
return as

class FileParseError(Exception):
    pass

# MainParser is the parser for the document node
# It figures out what kind of file it is dealing with
# and delegates the parsing accordingly

class MainParser(Parser):
    
def __init__(self, fileName):
        reader = Sax2.Reader()
        docNode = reader.fromUri("file:" + fileName)
        Parser.__init__(self, docNode, docNode)

    def fileType(self):
        "Check for the file type (e.g. FOAF, Request)"
        people = self.findPath("/rdf:RDF/foaf:Person")
        if len(people) > 0:
            return "foaf"
        organizations = self.findPath("/rdf:RDF/foaf:Organization")
        if len(organizations) > 0:
            return "foaf"
        # Check for a request tag
        req = self.findPath("/rdf:RDF/pima:Request")
        if len(req) > 0:
            return "request"

    def mainEntity(self):
        "Returns the main entity parser for the FOAF file -
either a foaf:Person or a foaf:Organization"
        # check to see a main entity exists
        if self.fileType() != "foaf":
            raise FileParseError, "Cannot find a mainEntity in
            a non-FOAF file"
        entities = self.findPath("/rdf:RDF/foaf:Person")
        entities.extend(self.findPath("/rdf:RDF/foaf:Organization"))
if len(entities) > 1:
    raise NodeParseError, "Multiple main foaf:Persons
    or foaf:Organizations"

return EntityParser(self.doc,entities[0])

def request(self):
    """Returns the request parser for the file""

    # check file type
    if self.fileType() != "request":
        raise FileParseError, "Cannot find a request in a
        non-request file"

    requestList = self.findPath("/rdf:RDF/pima:Request")

    if len(requestList) > 1:
        raise NodeParseError, "Multiple pima:Requests
        in one file"

    return RequestParser(self.doc,requestList[0])

# This describes a foaf:Person or foaf:Organization

class EntityParser(Parser):
    def __init__(self,docNode/entityNode):
        # Run a sanity check on the entity
        if (entityNode.nodeName != 'foaf:Person'
            and entityNode.nodeName != 'foaf:Organization'):
            raise NodeParseError, "Node is not foaf:Person
            or foaf:Organization"

        Parser.__init__(self,docNode/entityNode)

    def pimaInfo(self,i):
        pI = self.findPath("pima:pima-info[%i]" % i)
if len(pI) == 0:
    return None
else:
    # check data here
    return PimaInfoParser(self.doc,pI[0])

def pimaKnows(self,i):
    pK = self.findPath("pima:knows[%i]" % i)
    if len(pK) == 0:
        return None
    else:
        # check data here
        return EntityParser(self.doc,pK[0])

class RequestParser(PrivacyCategoryParser):
    def __init__(self,docNode,requestNode):
        if (requestNode.nodeName != 'pima:Request'):
            raise NodeParseError, "Node is not pima:Request"
        PrivacyCategoryParser.__init__(self, docNode,requestNode)
        # check to make sure data is ok
        sourceList = self.findPath('@pima:source')
        if len(sourceList) > 0:
            self.source = sourceList[0].nodeValue
        else:
            self.source = None
        destList = self.findPath('@pima:dest')
        if len(destList) > 0:
            self.dest = destList[0].nodeValue
        else:
            self.dest = None
def request(self):
    return Request(self.userList(), self.dataList(),
                   self.purposeList(), self.actionList(), [])

class PimaInfoParser(Parser):
    def __init__(self, docNode, piNode):
        if (piNode.nodeName != 'pima:pima-info'):
            raise NodeParseError, "Node is not pima:pima-info"
        Parser.__init__(self, docNode, piNode)
        # check to make sure data is ok
        sourceList = self.findPath('@pima:source')
        if len(sourceList) > 0:
            self.source = sourceList[0].nodeValue
        else:
            self.source = None
        destList = self.findPath('@pima:dest')
        if len(destList) > 0:
            self.dest = destList[0].nodeValue
        else:
            self.dest = None

    def privacyPolicy(self):
        pp = self.findPath('pima:PrivacyPolicy')
        if len(pp) > 1:
            raise NodeParseError, "Multiple privacy policies
                         in a pima-info node"
        if len(pp) == 0:
            return None
def transaction(self, i):
    trans = self.findPath("pima:transaction[%i]") % i
    if len(trans) == 0:
        return None
    else:
        # check data here
        return TransactionParser(self.doc, trans[0])

class TransactionParser(Parser):
    def __init__(self, docNode, tNode):
        if tNode.nodeName != 'pima:transaction':
            raise NodeParseError, "Node is not pima:transaction"
        Parser.__init__(self, docNode, tNode)
        # check to make sure this data is ok
        timeList = self.findPath("@pima:time")
        if len(timeList) == 0:
            raise NodeParseError, "Transaction does not have a time"
        self.time = timeList[0].nodeValue

def transactionData(self, i):
    transD = self.findPath("pima:transaction-data[%i]") % i
    if len(transD) == 0:
        return None
    else:
        # check data here
        return TransactionDataParser(self.doc, transD[0])
class TransactionDataParser(Parser):
    def __init__(self, docNode, tdNode):
        if (tdNode.nodeName != 'pima:transaction-data'):
            raise NodeParseError, "Node is not pima:transaction-data"
        Parser.__init__(self, docNode, tdNode)
        # check to make sure this data is ok
        dxp = self.findPath("@pima:datapath")
        if len(dxp) == 0:
            raise NodeParseError, "Transaction does not have a
datapath attribute"
        else:
            self.dataXPath = dxp[0]

class PrivacyPolicyParser(Parser):
    def __init__(self, docNode, ppNode):
        if (ppNode.nodeName != 'pima:PrivacyPolicy'):
            raise NodeParseError, "Node is not pima:PrivacyPolicy"
        Parser.__init__(self, docNode, ppNode)
        # check to make sure this data is ok
        drs = self.findPath("@default-ruling")
        if len(drs) == 0:
            raise NodeParseError, "No default ruling provided
in privacy policy"
        else:
            defaultRuling = drs[0].nodeValue
            if (defaultRuling != "allow"
                and defaultRuling != "deny"):
raise NodeParseError, "PrivacyPolicy's default-ruling is not well-formed"

self.defaultRuling = defaultRuling

def rule(self, i):
    rs = self.findPath("pima:Rule[%i]" % i)
    if len(rs) == 0:
        return None
    else:
        r = rs[0]
    return RuleParser(self.doc, r)

def policy(self):
    p = Policy(None, self.defaultRuling)
    i = 1
    while 1:
        r = self.rule(i)
        if r == None:
            break
        p.rules.append(r.rule())
        i = i + 1
    return p

class RuleParser(PrivacyCategoryParser):
    def __init__(self, docNode, rNode):
        if (rNode.nodeName != "pima:Rule"):
            raise NodeParseError, "Node is not pima:Rule"
        PrivacyCategoryParser.__init__(self, docNode, rNode)

    def rule(self):
idL = self.findPath("@id")
if len(idL) == 0:
    raise NodeParseError, "Rule does not have an id attribute"

id = idL[0].nodeValue
rulingL = self.findPath("@pima:ruuling")
if len(rulingL) == 0:
    raise NodeParseError, "Rule does not have a ruling attribute"

ruling = rulingL[0].nodeValue
if ruling != "allow" and ruling != "deny":
    raise NodeParseError, "Rule attribute ruling is not allow or deny"

return Rule(id,ruling,self.userList(),self.actionList(),
            self.dataList(),self.purposeList())

#########################################
# PIMA Classes
#########################################

class Transaction:
    pass

#########################################
# EPAL Classes and EPAL Rule Handling
#########################################
# Not needed

class Vocabulary:
    pass

class DataIntegrityError(Exception):
    """Base class for exceptions that deal with input data integrity"""
    pass

class RulingIntegrityError(DataIntegrityError):
    """Ruling is not "allow" or "deny" """
    pass

class Container:
    def __init__(self):
        self.attributes = {}  
    def addAttribute(self, attributeName, attributeValue):
        self.attributes[attributeName] = attributeValue
    def __repr__(self):
        s = "CONTAINER\n" + str(self.attributes)
        return s

class Rule:
    def __init__(self, id, ruling, userCategorySet, actionSet, 
                 dataCategorySet, purposeSet, conditionSet=None, 
                 obligationSet=None):
        # integrity check data 
        if (ruling != "allow" and ruling != "deny"):
            raise RulingIntegrityError, ruling
#build data structure
self.id = id
self.ruling = ruling
self.userCategories = userCategorySet
self.actions = actionSet
self.dataCategories = dataCategorySet
self.purposes = purposeSet
self.conditions = conditionSet
self.obligations = obligationSet

def __repr__(self):
    s = "RULE\n"
    s = s + self.id + "\n"
    s = s + self.ruling
    s = s + "\nUsers: " + str(self.userCategories)
    s = s + "\nActions: " + str(self.actions)
    s = s + "\nData: " + str(self.dataCategories)
    s = s + "\nPurposes: " + str(self.purposes)
    s = s + "\nConditions: "
    if self.conditions != None:
        for condition in self.conditions:
            s = s + "\n" + str(condition)
    s = s + "\nObligations: "
    if self.obligations != None:
        for obligation in self.obligations:
            s = s + "\n" + str(obligation)
    return s

class Request:
    def __init__(self,userCategorySet,dataCategorySet,purposeSet,
actionSet, container):
    self.userCategories = userCategorySet
    self.dataCategories = dataCategorySet
    self.purposes = purposeSet
    self.actions = actionSet
    self.container = container

def __repr__(self):
    s = "REQUEST"
    s = s + "\nUsers: " + str(self.userCategories)
    s = s + "\nActions: " + str(self.actions)
    s = s + "\nData: " + str(self.dataCategories)
    s = s + "\nPurposes: " + str(self.purposes)
    s = s + "\nContainer: "
    for element in self.container:
        s = s + "\n" + str(element)
    return s

class RulingError(Exception):
    pass

class Ruling:
    """The Ruling is either "allow", "deny", or "not-applicable".
    If the Ruling is "allow", then obligations are also returned."""
    def __init__(self, ruling, ruleID=None, obligations=None):
        # check data
        if (ruling != "allow" and ruling != "deny" and
            ruling != "not-applicable"):
            raise RulingError, "Ruling’s ruling attribute
            is not valid"
        if (obligations != None and not isinstance(obligations,
        

list()):
    raise RulingError, "Ruling obligations is not either
    a list or None"

if (ruling == "not-applicable" and (ruleID != None
    or obligations != None)):
    raise RulingError, "Ruling is \"not-applicable\",
    but does not have correct
    attributes"

#initialize data
self.ruleID = ruleID
self.ruling = ruling
self.obligations = obligations

def __repr__(self):
    s = "RULING"
    s = s + "\nRuling: " + self.ruling
    s = s + "\nRule IDs: " + str(self.ruleID)
    s = s + "\nObligations: " + str(self.obligations)
    return s

class CompoundRuling:
    def __init__(self,ruling=None):
        self.allowed = False
        self.denied = False
        self.allowObligations = []
        self.denyObligations = []
        self.allowRules = []
        self.denyRules = []
        if ruling != None:
            self.addRuling(ruling)
    def addRuling(self,self,ruling):

90
if ruling.ruling == "deny":
    # check for prior existence
    for r in self.denyRules:
        if ruling.ruleID == r:
            return
    # No allowRules and no allowObligations
    if self.allowed == True:
        self.allowed = False
    if self.allowObligations != []:
        self.allowObligations = []
    if self.allowRules != []:
        self.allowRules = []
    if self.denied != True:
        self.denied = True
    self.denyObligations.append(ruling.obligations)
    self.denyRules.append(ruling.ruleID)
elif (ruling.ruling == "allow" and self.denied == False):
    # check for prior existence
    for r in self.allowRules:
        if ruling.ruleID == r:
            return
    if self.allowed != True:
        self.allowed = True
    self.allowObligations.append(ruling.obligations)
    self.allowRules.append(ruling.ruleID)
def __repr__(self):
    s = "COMPOUND RULING\n"
    if self.allowed:
        s = s + "allow"
    s = s + "\nRules: " + str(self.allowRules)
s = s + "\Obligations: " + str(self.allowObligations)

elif self.denied:
    s = s + "deny"

    s = s + "\Rules: " + str(self.denyRules)

    s = s + "\Obligations: " + str(self.denyObligations)

return s

class RuleProcessingError(Exception):
    pass

class InvalidUserCategoryIndexError(RuleProcessingError):
    pass

class PolicyError(Exception):
    pass

class Policy:
    def __init__(self,vocabulary,defaultRuling):
        self.rules = []
        self.vocabulary = vocabulary
        if (defaultRuling != "allow" and defaultRuling != "deny"):
            raise PolicyError, "Default ruling is not either \"allow\" or \"deny\"

        self.defaultRuling = defaultRuling

def inScope(self,x,y,hierarchy):
    "Returns True iff x is in the scope of y in the hierarchy"
    subset = False
    for element in y:
        if x == element:
subset = True
break
if (subset):
    debug(str(x) + " subset of " + str(y) + " in "
    + hierarchy)
else:
    debug(str(x) + " NOT subset of " + str(y) +
    " in " + hierarchy)
return subset

def userCategoryInScope(self,ruleUCs,requestUC):
    return self.inScope(requestUC,ruleUCs,"userCategory")

def dataCategoryInScope(self,ruleDCs,requestDC):
    return self.inScope(requestDC,ruleDCs,"dataCategory")

def purposeInScope(self,rulePurposes,requestPurpose):
    return self.inScope(requestPurpose,rulePurposes,"purpose")

def actionInScope(self,ruleActions,requestAction):
    return self.inScope(requestAction,ruleActions,"action")

def requestInScope(self,rule,reqUserCategory,reqDataCategory,
    reqPurpose,reqAction):
    return (self.userCategoryInScope(rule.userCategories,
        reqUserCategory)
    and self.dataCategoryInScope(rule.dataCategories,
        reqDataCategory)
    and self.purposeInScope(rule.purposes,reqPurpose)
    and self.actionInScope(rule.actions,reqAction))
def conditionsMet(self, condition, container):
    return True

def evaluateRule(self, rule, reqUserCategory, reqDataCategory, reqPurpose, reqAction, reqContainer):
    """Processes a simple request against one rule and outputs a Ruling""
    if (self.requestInScope(rule, reqUserCategory, reqDataCategory, reqPurpose, reqAction) and self.conditionsMet(rule.conditions, reqContainer)):
        if rule.ruling == "allow":
            return Ruling("allow", rule.id, rule.obligations)
        elif rule.ruling == "deny":
            return Ruling("deny", rule.id, rule.obligations)
    else:
        return Ruling("not-applicable")

def evaluateSimpleRequest(self, userCategory, dataCategory, purpose, action, container):
    """Processes a simple request against a policy of rules Outputs a Ruling of "accept", "deny", or "not-applicable" ""
    for rule in self.rules:
        ruling = self.evaluateRule(rule, userCategory, dataCategory, purpose, action, container)
        debug("Partial ruling: " + str(ruling))
        if (ruling.ruling == "allow" or ruling.ruling == "deny"):
            return ruling
        # otherwise, it must have been "not-applicable"
def evaluateCompoundSingleUserRequest(self, userCategory, dataCategories, purposes, actions, container):
    """Processes a compound request
    Outputs a CompoundRuling, which is either an accept or a deny""
    cr = CompoundRuling()
    # For the user to ultimately get an "accept", all possibilities
    # of dataCategories, purposes, and actions must return
    # an accept or not-applicable, where the default is accept
    for dataCategory in dataCategories:
        for purpose in purposes:
            for action in actions:
                ruling = self.evaluateSimpleRequest(
                    userCategory, dataCategory, purpose,
                    action, container)
                # Check for implicit denials
                if (ruling.ruling == "not-applicable"
                    and self.defaultRuling == "deny"):
                    # This is not an explicit denial
                    # Instead, it is fair to try other users,
                    # so return a not-applicable CompoundRuling
                    return CompoundRuling(
                        Ruling("not-applicable"))
                # If there is a deny, continue and collect
                # all denies
                if ruling.ruling != "not-applicable":
                    cr.addRuling(ruling)
    return cr
def evaluate(self, request):
    # is this is simple request or a compound request?
    if (len(request.userCategories) == 1
        and len(request.dataCategories) == 1
        and len(request.purposes) == 1
        and len(request.actions) == 1):
        # Simple request
        ruling = self.evaluateSimpleRequest(
            request.userCategories[0],
            request.dataCategories[0],
            request.purposes[0], request.actions[0],
            request.container)
        if ruling.ruling == "not-applicable":  # no rules fired
            return CompoundRuling(Ruling(self.defaultRuling))
        else:
            return CompoundRuling(ruling)
    else:
        # Compound Request
        for user in request.userCategories:
            cr = self.evaluateCompoundSingleUserRequest(
                user, request.dataCategories,
                request.purposes, request.actions,
                request.container)
            if (cr.allowed or cr.denied):
                return cr
        # No users allowed or denied
        cr = CompoundRuling()
        if self.defaultRuling == "allow":
            cr.allowed = True

96
elif self.defaultRuling == "deny":
    cr.denied = True

    return cr

def __repr__(self):
    s = "POLICY"
    s = s + "\nDefault Ruling: " + self.defaultRuling
    s = s + "\nRULES"
    for rule in self.rules:
        s = s + "\n" + str(rule)
    return s

#########################################################################
# PIMA
#########################################################################

class FoafFileError(Exception):
    "Foaf file has an error"
    pass

class NoRelevantPolicyError(Exception):
    "The PIMA/FOAF file does not have a relevant privacy policy"
    pass

#This is the top-level code that runs this program
class PIMA:
    def __init__(self,foafFileName):
        self.main = MainParser(foafFileName)
# check file integrity here
if self.main.fileType() != "foaf":
    raise FoafFileError, "The data repository
    is not a FOAF file"

# This method dispatches to the proper parsers
def process(self,fileName):
    main = MainParser(fileName)
    # check file integrity?
    if main.fileType() == "request":
        self.handleRequest(main)

def handleRequest(self,requestMain):
    print "Handling request"
    request = requestMain.request().request()
    # Find the corresponding pimaInfo, if any
    # The first match applies
    # deny if there are no pimaInfos
    reqP = requestMain.request()
    i = 1
    while 1:
        pimaInfo = self.main.mainEntity().pimaInfo(i)
        if pimaInfo == None: #no more pimaInfos left
            raise NoRelevantPolicyError, "No policies
            acceptable for this entity"
        if (pimaInfo.dest == None
            or reqP.dest == pimaInfo.dest):
            if reqP.source == pimaInfo.source:
                break
        i = i + 1
pp = pimaInfo.privacyPolicy()
eval = pp.policy().evaluate(reqP.request())
print eval
Bibliography


