A Formal Approach to P3P Privacy Policies Evaluation

Vincent Cridlig, Olivier Festor, Jacques Guyard, Pierre-Etienne Moreau

A Formal Approach to P3P Privacy Policies Evaluation

Vincent Cridlig, Olivier Festor, Jacques Guyard and Pierre-Etienne Moreau

Abstract— **Writing and evaluating policies is a recurrent problem over the Internet. These policies, which deal with fields like privacy (P3P), security (P3P) and provisioning(COPS-PR) are often made of many ordered rules. In this paper, we present our work related to P3P rules formalization and validation in an evaluation environnement based on rewriting process. We also develop an environnement which returns the right behavior regarding Internet ressources, P3P policies and APPEL user's preferences.**

Keywords— **policy, rewriting rules, P3P, APPEL, ELAN.**

I. Introduction

OVER the past ten years, privacy within the Internet has become more and more important for users. Many international organizations both public and private are developing many efforts to enable the user's trust to increase through the establishment of new legal barriers as well as the design of software based techniques that are aligned with the legal obligations [1][2]. One of the organization that is developping technical recommendations in this area is the W3C (World Wide Web Consortium). It has defined a new environment called the P3P (Platform for Privacy Preference [3]) which is based on XML dedicated to pricavy preference processing in the context of Web interactions.

When a user connects to a web server, he naturally distributes some data about himself and his behavior (e.g. visited pages, average time between two visits, ...). This information is sometimes merged with the one of other sites in order to profile visitors. That is the reason why users expect increasing transparency in the field of web pratices. Most of the users would trust web sites if they could control the diffusion of their personal information. For instance, it would be interesting to know which information a site accepts to provide, to whom, how long this information will be stored, for which purpose it is collected... However, web users can not inform by themself about the practices of each site they visit. Obviously, it would be too complex and time-consuming. Moreover, this problem is increased because of hidden transactions. Indeed, an HTML ressource may contain several other ressources provided by many other sites. So more than one policy concerned this single ressource.

P3P, which is a w3c recommandation, is a partial answer to this privacy request by allowing web sites to inform users about their privacy policies. P3P defines a standardized language which makes it possible to describe these policies.

P3P also provides users a means of localizing these policies within web sites and comparing them to their own preferences. If user requests are met, he should access the ressources. Otherwise, he is warned thanks to a popup window and he can continue his navigation. This is the "notice and choice" approach.

This paper presents the use of a rewriting language (ELAN [4]) through different approaches.
- evaluation engine of policies written in specific languages
- evaluation engine of policies written directly in ELAN language
- syntactic validation

The first and the third approach are illustrated with the P3P evaluator. The second one is illustrated with the ELAN firewall. Another interesting approach would be to process semantic validation, which means checking rules coherence.

We show that ELAN[4], a rule-based language, is particularly well adapted for evaluating user defined preferences against site policies.

The paper is organized as follows: section 2 introduces the basic concepts of P3P, APPEL [5] and ELAN. Section 3 describes the ELAN rules formalization. In section 4, details about the choices made for the prototype developement are given. Section 5 adds other aspects of what is possible to do with ELAN in the networking field beyond privacy and P3P. A conclusion is drawn in section 6.

II. Concepts

A. The platform for Privacy preferences (P3P)

P3P is a w3c recommendation which aims to allow web site to describe their privacy policies in XML (eXtensible Markup Language).

First, for each collected data type, a web site can describe the way it will use this private data. The main information about this use is:
- the goal of this collect (administration, maintaining, customization, ...)
- data recipients (the site itself, its trading partners, ...)
- the retention of this data
- the data type

Web sites deploying P3P also announce their own data (address, ...), the access they let to users who

LORIA - INRIA Lorraine. 615, rue du jardin botanique. 54602 Villers-les-Nancy, France. Email: {cridligv,festor,guyard,moreau}@loria.fr

want to read the data they provide, the means for users in case of dispute. All this information is gathered in one or more XML policy files.

Then, users must be able to know which policy applies to a given ressource. So web sites create another XML file which links ressources to policies. This file is called policy reference file.

Users also need to know where is this policy reference file. w3c gives several means to answer this question:
- copying this file to a well-known location. This means to the URI www.example.com/w3c/p3p.xml
- adding an HTTP header containing this URI in all the responses from the server
- adding a special tag (LINK) containing this URI in all HTML files

The figure 1 shows how user agents and web servers interact. First, the agent requests a ressource: index.html. Then, the server replies by sending the ressource and the policy reference file URI in a HTTP header. Next, the user agent requests this file. The server send this file back. If the right policy is included in the policy reference file, then the user agent needs not to request the file containing the policy associated with the index.html ressource. Otherwise the user agent sends a new request to obtain the right policy file.

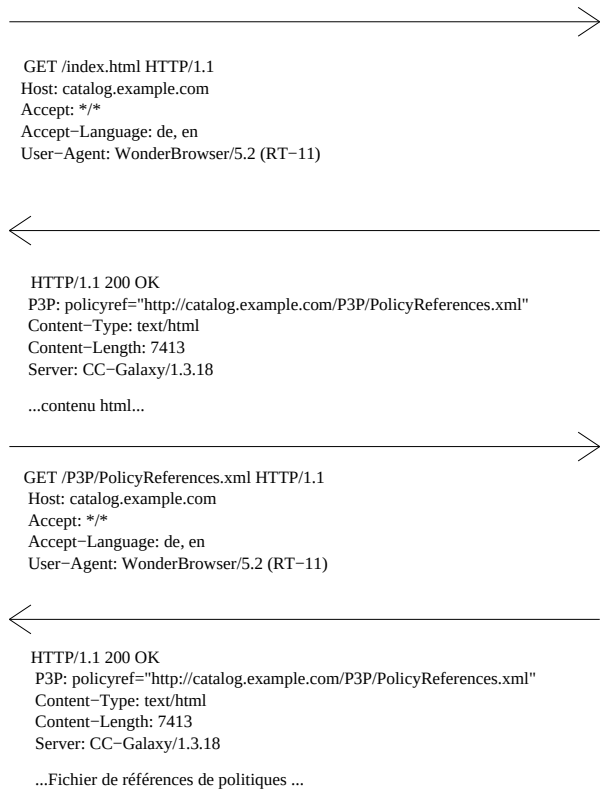

Fig. 1. User Agent / Web Server Interaction

Note that, according to the P3P recommendation, the evaluation is asynchronous: this means that during the critic period when the policy has not been fetched yet, the user agent considers the site as if it has not deployed P3P. Then, the warning process is enabled. However, to limit abuses, w3c recommands to place P3P files in safe zones which do not use cookies and do not collect personnal data.

B. APPEL presentation

APPEL (A P3P Preference Exchange Language) is a P3P-based language defined by w3c. It aims at providing a means for web users to describe their privacy preferences. These preferences are written in XML. Obviously, we can not request basic users to write their own APPEL file which contains logic connectors, regular expressions, well defined tags and attributes... That is why it is possible to import default preferences adapted to most users. But users are allowed to create their own APPEL ruleset if they feel they can write it.

An APPEL preferences file contains a set of rules gathered in the RULESET tag. Each rule is associated with a behavior if it matches the web site policy. Three behaviors are allowed:
- request: the site's policy is acceptable
- limited: the access to the ressource is limited
- block: the access to the ressource should not be given

Below is an extract of an APPEL file which states that cookies posted for tailoring purpose will be accepted if the recipient is only the web site itself. The category `State` means that the concerned cookie is authorized to be used to manage states in a stateless protocol (HTTP):

```
<appel:RULE behavior="request">
  <p3p:POLICY>
    <p3p:STATEMENT>
      <p3p:RECIPIENT appel:connective="and">
<p3p:ours/>
      </p3p:RECIPIENT>
      <p3p:PURPOSE appel:connective="non-and">
<p3p:tailoring/>
      </p3p:PURPOSE>
      <p3p:DATA-GROUP>
       <p3p:DATA ref="#dynamic.cookies">
<p3p:CATEGORIES appel:connective="or">
          <state/>
</p3p:CATEGORIES>
       </p3p:DATA>
      </p3p:DATA-GROUP>
    </p3p:STATEMENT>
  </p3p:POLICY>
</appel:RULE>
```

This example shows that APPEL language uses the same tags as the P3P language. The user preferences file structure is very close to the P3P one. This helps the comparison of the two files.

C. ELAN

ELAN is a specification formalism which makes it possible to describe some multi-sorted signatures, some conditional rewriting rules and some control strategies. One of the ELAN particularities is that it allows the use of out-of-context grammar in order to describe signatures.

This makes it possible to describe and use infixed operators which makes ELAN a smart language to specify complexe data structures like the one used in automated provers, constraints solvers or networks modeling tools.

The first part of an ELAN specification consist on the sorts definition, the imported modules list and a set of out of context grammar rules to declare the functions symbols. Consider, for instance, the signature of a module allowing to model a firewall policy. Usually, such a firewall policy is represented as a table telling which IP datagrams must be filtered and which others must be routed. Here is an firewall example.

ptcl	IPsrc	PortSrc	IPdst	PortDst	action
tcp	1.2.3.4	any	*.*.*.*	80	deny
tcp	1.2.3.*	any	*.*.*.*	80	accept
tcp	*.*.*.*	any	1.2.3.4	80	accept

TABLE I
FIREWALL RULES EXAMPLE

The first table line means that a datagram whose protocol is tcp, whose IP address is 140.192.37.20, coming from any source port and going to any ip destination and destination port is 80, will be filtered.

Describing in ELAN the rules of a firewall policy is equivalent to formalising the information to handle by giving a notation. Saying that `tcp` and `udp` are protocols, and saying that a IP address is made of four bytes separated by dots is written as follows in ELAN :

```
tcp     :Ptcl;
udp     :Ptcl;
@       :(int) Byte;
@.@.@.@ :(Byte Byte Byte Byte) IPAdr;
```

Similarly, we call Header the set of information which is the left member of a firewall rule. A Header is made of a protocol, a IP source address, a source port, a IP destination address and a destination port. If we separate each of these elements with a comma, we translate it as follows:

```
@        :(int) Port;
@,@,@,@,@:(Prtcl IPAdr Port IPAdr Port)Header;
```

The ELAN syntax definition formalism is powerful enough to describe each language expressible with out-of-context grammar rules. This syntax makes it possible to describe and build the data algebric structure represented by terms.

The second part of an ELAN specification consist of the definition of operations handling the previous data structures. The elementary evaluation mecanism rely on the *rewriting*: the rewriting rules are term pairs. (l, r) denoted `l => r` and are used to define a relation between two closed terms.

In this way, we can define a set of rewrinting rules which makes a firewall policy runnable. The translation of policies into rules is straight-forward: we just need to replace what was representing any value (*any* or `*`) with a ELAN variable: any, x1,...,x4,y1,...,y4. In this manner, the translation of the three rules in the table I gives:

```
rules for string
any                          :int;
x1,x2,x3,x4,y1,y2,y3,y4      :Byte;
global
[] eval(tcp,140.192.37.20,any,y1.y2.y3.y4,80)
   => "deny"   end
[] eval(tcp,140.192.37.x4,any,y1.y2.y3.y4,80)
   => "accept" end
[] eval(tcp,x1.x2.x3.x4,any,161.120.33.40,80)
   => "accept" end
...
end
```

In this example, we introduce a new operator: `eval(@):(Header) string`. It allows us to build a term to evaluate. In practice, for each arrival datagram, we just need to call the `eval` function in order to know if the datagram must be accepted or refused.

To compile a set of rules, ELAN uses filtering techniques based on discrimination trees. The complexity of the filtering process depends only on the size of the term to evaluate but not on the system rule number. In this way, the ELAN compiler makes it possible to apply several million rules per second which is acceptable in practice.

III. RULES FORMALIZATION IN ELAN

A. Sorts definitions

To formalize in ELAN the rules of a user's preferences APPEL file, we started with the pseudo-grammar of the APPEL specification.

First, the set of tags defined in P3P and APPEL must be recognized. Thus, in ELAN, we write (line 1 up to 3 of figure 2) that current, admin, develop, ... are tag names (element_name). In the same way, all the existing attributes are recognized as atrbute_names thanks to definitions on line 5 up to 7. Then, we are able to recognize larger entities like attribute_exp (line 9). For instance, behavior="request" type is attribute_exp. An attributes list from the same tag is gathered in the sort attribute_exps: line 10 means that an attribute is a set of attribute which contains only one element. Line 11 gathers two attributes sets in order to create a single set which contains all their attributes.

Our software component is able to parse element names and attribute names. We now can define empty elements like `<current/>` or `<p3p:DATA ref="#user.*"/>`. An empty-expression is made of a name and zero or more attributes. Here is the formalism used in the APPEL specification: `empty-exp="<"elem-name *att-exp"/>"` To translate this in ELAN, we write lines 13 and 14 whose formalism is close to the one of the APPEL grammar. Obviously, we reuse the previous definitions of sorts `element_name` and `attribute_exps`. In the same way, containing tags are defined on line 15.

To write these type definitions, we were inspired by the work related in [6] which deals with the developpement of an XSLT processor in ELAN. Recognizing the components of a P3P or APPEL file allows us to parse it and to evaluate it efficiently.

```
current :ele_name;
admin  :ele_name;
develop :ele_name;
...
ref    :att_name;
crtdby :att_name;
behavior:att_name;
...
@=@    :(att_name string) att_exp;
@      :(att_exp) att_exps;
@ @    :(att_exps att_exps) att_exps assocRight;
...
<@ @/> :(ele_name att_exps) empty_exp;
<@/>  :(ele_name) empty_exp;
<@>@</@>:(ele_name ctd_exps ele_name) ctg_exp;
...
```

Fig. 2. Sorts definition

B. Signatures definition

We defined five main functions which make it possible to process these XML documents. The function called in the initial request is `eval` which takes two parameters: the set of APPEL rules and the web site policy. `eval` is responsible for calling the APPEL rules in the right order. Moreover, it calls the comparison function `cmp` which also takes two parameters: the policy contained in the APPEL rule and the web site policy. It aims to compare these two policies. `cmp` returns a boolean which allows `eval` to take the right decision. If the APPEL rule matches the policy, the process stops and the final result is the associated behavior.

C. Rewriting rules

In this paragraph, we will describe some rules which provide a good idea of the evaluation process.

```
  /****genreal cases******/
eval(<RULE behavior=b at>p</RULE>hs,e)
=> b if cmp(p,e) end
eval(<RULE behavior=b at>p</RULE>hs,e)
=> eval(hs,e) if not(cmp(p,e)) end

  /****terminal cases*****/
eval(<RULE behavior=b at>p</RULE>,e)
=> b if cmp(p,e) end
eval(<RULE behavior=b at>p</RULE>,e)
=> "" if not(cmp(p,e)) end
```

As we saw before, `eval` takes two parameters: the set of APPEL rules and the web site P3P policy. In the general case, there are several rules in the APPEL file. The evaluation result is the behavior of the first matching APPEL rule. "hs" represents the other APPEL rules which follow the first one. `cmp` tests the compatibility between an APPEL rule and a policy "e". In the terminal case, there is only one possible matching rule. If it is matching, the result will be the behavior associated to it. Otherwise, the result is an empty string: this case is considered as an error case. Normally, at least one rule must apply for a given policy because of the special tag OTHERWISE.

We saw that to evaluate an APPEL rule, we test `cmp(p,e)` where p is the policy included in this rule and e is the web site policy (evidence). `cmp` can compare each kind of tags. As we saw in the section "sorts definitions", there are two main cases: empty tags and containing tags.

In the first case which consists on comparing two empty tags, we write the following rule: `cmp(<a/>,<a/>) => true end`

In the second case, the following rules call `cmpEns` which allows to compare the contained list of tags. There are different sub-case depending on the connective attribute which can be omitted. The default value is `and`.

```
cmp(<a appel:connective=co>d</a>,<a>h</a>)
   => cmpEns(d,h,co)  end
cmp(<a appel:connective=co>d</a>,<a at>h</a>)
   => cmpEns(d,h,co)  end
cmp(<a>d</a>,<a>h</a>)
   => cmpEns(d,h,"and") end
cmp(<a>d</a>,<a at>h</a>)
   => cmpEns(d,h,"and") end
```

There is also a third intermediate case which deals with DATA tag. If the value of the ref attribute is a variable category like #dynamic.cookies, we have to compare this category:

```
cmp(<DATA ref=r>d</DATA>,<DATA ref=s>h</DATA>)
=> cmpEns(d,h,"and") and match(r,s) end
cmp(<DATA ref=r/>,<DATA ref=s/>)
=> match(r,s)      end
cmp(<DATA ref=r/>,<DATA>d</DATA>)
=> cmpEns(<CATEGORIES>categ(r)</CATEGORIES>,d,
   "and")                          end
```

Comparing containing elements like in figure 3 is equivalent to compare their content which may be made of several elements. So we need to use a new function: `cmpEns`. `cmpEns` takes three parameters:

- a set R of APPEL elements
- a set E of P3P elements
- a connective which gives the way to compare these sets

`cmpEns` creates recursively the logical expression we need to evaluate. For instance, if the request is `cmpEns(R,E,"and")`, we have to check that the set R is included in E, which is equivalent to:

$$\forall r_i \in R, r_i \in E$$

To do that, we build the expression recursively by rewriting `cmpEns` rules. The previous expression is

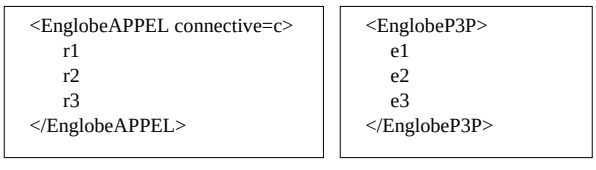

Fig. 3. Elements set comparison

equivalent to: [cmp(r1,e1) or cmp(r1,e2) or cmp(r1,e3) or ...] and [cmp(r2,e1) or cmp(r2,e2) or cmp(r2,e3) or ...] and ...

Another aspect of this comparison is the DATA tag and more precisely its ref attribute. DATA tag has a particular role because it contains the collected data type. This data is structured and separated by dots like "user.name". In order to compare two data, we formalize their structure using ELAN and we define a `match` function. Each element (like user, name or *) is called a token and a tokens combination separated by dots is also a token:

```
*       :token;
user       :token;
name       :token;
bdate       :token;
...
@.@       :(token token) token assocRight;
match(@,@) :(token token) bool;
```

We compare the left tokens first. That is why we use the ELAN keyword assocRight. If one fragment identifier is a prefix of the other, the two ref attributes match. With ELAN language:

```
[] match(x.y,x.z) => match(y,z) end
[] match(x.y,x) => true end
[] match(x,x.z) => true end
[] match(x,x) => true end
```

For two given parameters, the number of applied rules equals the tokens' number of the smallest expression in the worst case. For instance, `match("#dynamic.*", "#dynamic.http.useragent")` needs two rewriting.

D. Rewriting example

Let's develop a rewriting example of `cmp` function:

```
[] cmp(<a appel:connective=co>d</a>,<a>h</a>)
=> cmpEns(d,h,co) end

cmp(
<RECIPIENT connective="or"><ours/></RECIPIENT>
,<RECIPIENT><ours/><same/></RECIPIENT>)
=> cmpEns(<ours/>,<ours/><same/>,"or")
=> or(<ours/>,<ours/><same/>)
=> cmp(<ours/>,<ours/>) or or(<ours/>,<same/>)
=> cmp(<ours/>,<ours/>) or cmp(<ours/>,<same/>)
=> true or false
=> true
```

The comparison of two RECIPIENT elements leads to the comparison of their content. We have to compare the two sets using "or" connective. So we check that {ours} is included in {ours,same} which means: (ours==ours) or (ours==same). The first condition is true so the result is true. If p3p:RECIPIENT had had some same level elements, the process would have been the same for each of these elements Fi and the result would have been: F1 and F2 and ... and true and ... and Fk where "true" is the result of the comparison we just made and "and" is the connective provided as `cmpEns` third parameter.

E. Rules coherence

Using a rule-based language often leads to coherence difficulties. When we launch an ELAN request, the best should be that only one rule matches. This is not the case with our P3P evaluator where rules order implicitely enters into account. It simplifies rules editing and ensures a determinist behavior. In order to avoid rules recovering, we should write many additionnal rules.

This problem is the same when someone wants to write firewall rules. It is difficult to be sure that rules are coherent. However, tools like [7] make it possible to valid firewall rules coherence. ELAN does not allow the administrator to check the rules coherence.

F. Comparison with policy expression dedicated languages

The main difference between ELAN and other languages like CIM or Ponder is that ELAN was not created specifically to write network policies. ELAN is a true programming language so that each policy can theoretically be described.

ELAN is conceptually near to CIM (Common Information Model): if event then action. Event (respct action) is equivalent to the left (respct right) part of ELAN rules.

Authorisation and delegation policies as well as domains defined in Ponder language can be easily translated into ELAN. In order to modelize domains, we just need to define the managers and to tell that each manager is an employee (`Alice : Manager; @ :(Manager)Employee;`). Then, we can write some rules to define access rights: `fileAccess(m,f) => true`, which means that each manager m can access each file f.

So our approach based on ELAN could be applied to network policies.

IV. Evaluator implementation

A. Evaluator supported elements

We initially made restrictive implementation choices for the P3P policy evaluator. In particular, the following elements and attributes are not supported:

- DISPUTES-GROUP element
- the comparison of DATA which need to define the categories of each DATA ref possible values (for instance, user.name predefined category is demographic). We showed that it is possible but only few of them are supported.

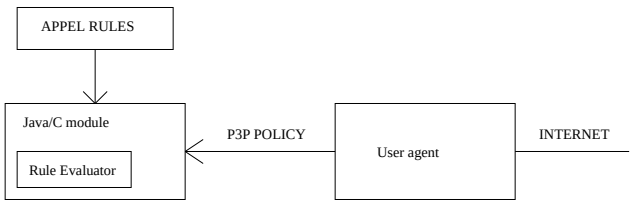

Fig. 4. Evaluation module

Apart from these restrictions and some minor details, our prototype supports the RULESET, RULE, POLICY, ACCESS, STATEMENT, PURPOSE, RETENTION, RECIPIENT, DATA-GROUP, DATA and CATEGORIES tags.

B. Integration within a browser

We are currently integrating the evaluator within an open-source web browser (i.e. Mozilla). It is be a C++ plug-in which will fetch web sites policies and policy reference files. The policy files will be transmitted to the evaluator (figure 4) and compared to the user APPEL file. The following request: `eval(<RULESET> ... </RULESET> , <POLICY> ... <POLICY>)` will be the first step of the evaluation process.

The policy evaluator is considered as an external program and is called each time a new HTTP request is send through a domain whose policy has not been fetched yet or is not yet valid.

C. Choices

Using a rule-based tool is interesting to write policies or to write rules which evaluates policies.

Indeed, in our prototype implementation, we chose to compare APPEL rules to P3P policies but we could also have translated APPEL rules into ELAN rules in order to simplify its implementation. The advantage of our choice is that this algorithm can evaluate each APPEL file whereas this other way would lead to create one ELAN file for each user. But the ELAN capabilities would have been used better. In a way, the rule evaluation is processed with other rules.

This way is the one we chose for the firewall policies. The ELAN program contains the specific policy rules and not some rules which evaluates others.

V. Conclusion

In this paper we addressed the use of a formal approach to specify and generate an evaluation environment for P3P-based privacy policy specifications using the ELAN framework. P3P evaluator and firewall provided as an introduction to ELAN language gave us a first overview of ELAN capabilities to formalize and evaluate privacy and security policies. We also found the P3P boundaries. For instance, P3P provides no way to be sure that an announced policy is really applied. That is why users can not be sure that their personal data will not be used in other goals than the described ones. However P3P is a great step for protecting privacy within the Internet. P3P should be combined with certification organizations which could control sites implementing P3P. In this way, user's trust would increase.

We implemented the environment in a software component which returns the appropriate behavior from a remote ressource, a P3P policy and user APPEL preferences. This environment is being integrated in the Mozilla browser.

In the short-term , we plan to make performance and scalability tests. Concerning our P3P evaluator, a continuous ressources control must not slow down the visit of users. However, the evaluation time (evaluation is always on the client-side) is far shorter than the time needed to download ressources so performance tests are not necessary. Concerning the firewall experimental environment, the requests in terms of performances are higher so we will target it to more performance oriented tests.

Our future work will focus on the study of policy models like P3P as a way to specify security rules in the management level.

References

[1] H. Hochheiser, "The platform for privacy preference as a social protocol: An examination within the u.s. policy context," *ACM Transactions on Internet Technology (TOIT)*, vol. 2, no. 4, pp. 276–306, 2002.

[2] D. M. Kristol, "Http cookies: Standards, privacy, and politics," *ACM Transactions on Internet Technology (TOIT)*, vol. 1, no. 2, pp. 151–198, 2001.

[3] M. M. M. P.-M. J. R. Lorrie Cranor, Marc Langheinrich, "The platform for privacy preferences 1.0 (p3p1.0) specification," W3C," Specification, 2002.

[4] H. D. C. K.-H. K. P.-E. M. Q.-H. N. C. R. M. V. Peter Borovansky, Horatiu Cirstea, "Elan user manual," INRIA," manuel utilisateur, 2002.

[5] M. M. Lorrie Cranor, Marc Langheinrich, "A p3p preference exchange language 1.0," W3C," Draft, 2002.

[6] C. Kirchner, Z. Qian, P. K. Singh, and J. Stuber, "Xemantis : a rewriting calculus-based semantics of xslt," INRIA, Tech. Rep., 2002.

[7] H. H. E. S. Al-Shaer, "Firewall policy advisor for anomaly discovery and rule editing," *IM 2003*, pp. 17–30, 2003.

[8] S. Duflos, G. Diaz, V. Gay, and E. Horlait, "A comparative study of policy specification languages for secure distributed applications," *DSOM 2002,LNCS 2506*, vol. 1, pp. 157–168, 2002.