Policy Merger System for P3P
In a
Cloud Aggregation Platform

Olumuyiwa Olurin

Thesis submitted to the
School of Electrical Engineering and Computer Science
In Partial Fulfillment of
The requirements for the degree of

Master of Applied Science,
Electrical and Computer Engineering

Under the auspices of the
Institute for Electrical and Computer Engineering (OCIECE)
Faculty of Engineering

uOttawa

University of Ottawa
Ottawa, Ontario, Canada

January 2013

Abstract

The need for aggregating privacy policies is present in a variety of application areas today. In traditional client/server models, websites host services along with their policies in different private domains. However, in a cloud-computing platform where aggregators can merge multiple services, users often face complex decisions in terms of choosing the right services from service providers. In this computing paradigm, the ability to aggregate policies as well as services will be useful and more effective for users that are privacy conscious regarding their sensitive or personal information.

This thesis studies the problems associated with the Platform for Privacy Preference (P3P) language, and the present issues with communicating and understanding the P3P language. Furthermore, it discusses some efficient strategies and algorithms for the matching and the merging processes, and then elaborates on some privacy policy conflicts that may occur after merging policies. Lastly, the thesis presents a tool for matching and merging P3P policies. If successful, the merge produces an aggregate policy that is consistent with the policies of all participating service providers.

Acknowledgements

A very special and well-deserved thank you to God and to the following:

1) I am indebted to my supervisors Dr. Carlisle Adams and Dr. Logrippo Luigi for their guidance, patience, insight, and encouragement during the last two years. They have always been very kind and friendly advisors. Their brilliance, steady perseverance and the pursuit of excellence in thinking and writing helped me to successfully coordinate my ideas and writing skills. These skills will continue to inspire me all the way.

2) The Computer Security and Privacy group including Ali Noman, Ruj Sushmita. Our monthly meetings, our formal and informal discussions about security, privacy and cryptography were of great help.

3) I would like to thank my dear friends Mary, Ivan, and Karen Beauchamp for providing their support and help all the way and for making my graduate life very fulfilling and enriching.

4) Lastly, I would like to thank my family and most importantly my parents for their support throughout my graduate studies. Without their care, patience, and encouragement, I would not have been able to complete this important step of my life's journey.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

The need for aggregate privacy policies is present in a variety of application areas today. With the growing success and advantage of cloud computing[1], many organizations are considering moving their data or applications into the cloud. Among many advantages of cloud computing, users have the opportunity to create, share, and manage their online data stored on the cloud. While creating exciting opportunities, cloud services bring new challenges. The user faces complex issues for choosing the right services, and the right service providers, or synchronizing and integrating service applications using the cloud infrastructure. As shown in Figure 1, managing and searching for data on the cloud is a major challenge to most cloud users; an average online user in countries such as the USA or Canada can have around 20 different online accounts. For example, some users can have accounts for storing and sharing documents (e.g. Dropbox, SugarSync), photos (e.g. Facebook, Picasa), and e-mails (e.g. Gmail, Yahoo, Hotmail).

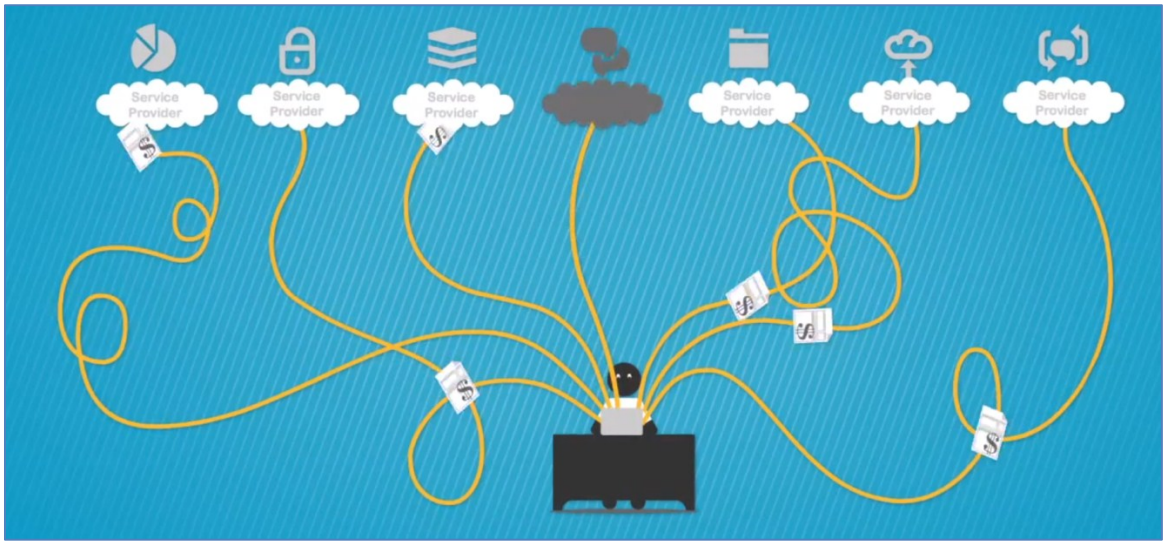


Figure 1: Cloud User is searching and managing multiple online accounts

[1] The International Data Corporation (IDC) indicates that the market for cloud computing services was 16.2 billion dollars in 2008 and will rise to 42.3 billion dollars per year by 2012. http://www.idc.com.

Service providers need a way to reach their entire community of users to support them on integrating services and maintaining their services. The cloud aggregation platform is the right tool to solve these challenges. Cloud aggregation platform is a hub for service providers that come together to find solutions to users' integrating and synchronizing problems. The platform can integrate services deployed in the cloud at the request of users. However, privacy policies from those merged services will remain in separate files, because the concept of merging privacy policies is not present in the aggregation platform for cloud computing. For example, when users receive an aggregate service along with separate multiple privacy policies, they may get confused regarding which privacy policy they should acknowledge. Moreover, a policy user agent that translates an encoded policy into a readable format also needs a single policy file to communicate a consistent summary of the policy.

Cloud computing describes both aggregation platforms and application services. As a platform it supplies, and configures servers. On the other hand, applications on the cloud are accessible via the Internet, and for this purpose large data centers and powerful servers host the services and applications [16] . Cloud computing combines the newest techniques to deliver services that have the following characteristics: rapid scalability, pay per use services, service delivery via broadband networks or the Internet, and sharing resources. However, information collected from people, organizations, agencies or service providers will be at the mercy of the cloud service providers [29]. Some organizations contemplating cloud computing adoption face a number of challenges on policy or security standards [12]. These concerns in return may discourage those organizations from adopting cloud-computing services.

Data privacy has become increasingly important for both users and service providers. Users that now benefit from cloud computing resources can easily release Personally Identifiable Information (PII) about themselves or their personal businesses to their service providers either consciously or subconsciously [5,7]. Some research analysts at reputable privacy and

security laboratories, such as Cylab Usable Privacy and Security Lab[2], found that the trail of information left can rapidly grow, and the potential for the misuse of this information is well-recognized [47].

Privacy issues are a major concern that prevents users from fully enjoying the flexibility, convenience, and variety offered by service providers in a cloud computing environment [47,25,61]. Users that are aware of possible consequences that may arise because of privacy violation on their PII are reluctant to adopt or use some cloud services. Furthermore, those users may quickly reject services that may request sensitive PII, such as credit card information, Social Insurance Number (SIN) or other sensitive personal data. Privacy violations can range from privacy accidents to illegal actions [25,5]. Those violations have generated fears and doubts in the mind of many experienced web users, prompting different questions such as

- What will the service providers do with their PII information after it serves the primary purpose?
- How long will the retention period affect their PII information?
- With whom will they share the PII information collected?

Bournez et al. [17] has defined some policy requirements and other aspects of privacy policies. In [17], they refer to privacy policy as a "set of rules stating how the provider will treat a piece of sensitive data collected. The data handling policy specifies for what purposes the data is collected, with which third parties will they share the data, the obligation to make use of the data as specified in the policy, and for how long they can store the data." Bournez et al. [17] further states that the main legal requirement imposed by institutions (such as Governments, European Union, etc.) is that privacy languages should be used for privacy policies and protection.

[2] Carnegie Mellon University, The Cylab Usable Privacy and Security Laboratory (CUPS) http:www.cups.cs.cmu.edu

1.1 Thesis Motivation

The concerns about the privacy of user data stored in the cloud have grown along with the cloud's popularity [32,68]. It is very important for a user to stay aware of the potential privacy hazards and have better control over disclosing private information. Privacy regulations in the USA, Canada, Europe, and elsewhere include a provision that regulates companies to give notice of their data practices through privacy policies. Privacy policies serve to increase transparency about data practices and support the "awareness" fair information practice principle [19].

Privacy policy languages were developed as one of the responses to the concerns about privacy issues. There are various open source privacy policy languages available on the Internet. A privacy policy language uses standardized syntax and semantics to create and compare user's privacy preferences with service providers' policies, and to make privacy policies more machine-readable and enforceable. However, as reviewed in the recent research [39,43,30], few researchers widely use or talk about standardized service privacy policy languages such as the Platform for Privacy Preferences (P3P) [24] and the Enterprise Privacy Authorization Language (EPAL) [71].

The World Wide Web Consortium (W3C[3]) proposed the P3P framework for automating the privacy policy verification process [24]. "The P3P project enables service providers to express their privacy practices in a standard XML format, which are automatically retrieved and interpreted by user agents. P3P user agents allow users to understand the provider's data handling practices (in-both machine readable and human readable formats) and automate decision-making [24]." The P3P framework is useful and more effective for situations where a user is interacting with a single privacy policy, because the framework was not originally designed to accommodate multiple privacy policies for a single HTTP request. To the best of our knowledge, none of the privacy mechanisms available can merge privacy policies on an aggregation platform. Due to this absence, users cannot derive a fair aggregate policy from

[3] W3C is the international organization responsible for developing protocols and guidelines for the World Wide Web (www). The W3C founded in 1994 to coordinate and support international, open development of technologies related to the web. http:/www.W3.org

multiple privacy policies when the need arises. The problem gets more important as companies adopt cloud computing technologies, because they move their customers' and company's private data, privacy policies, and business applications into the cloud.

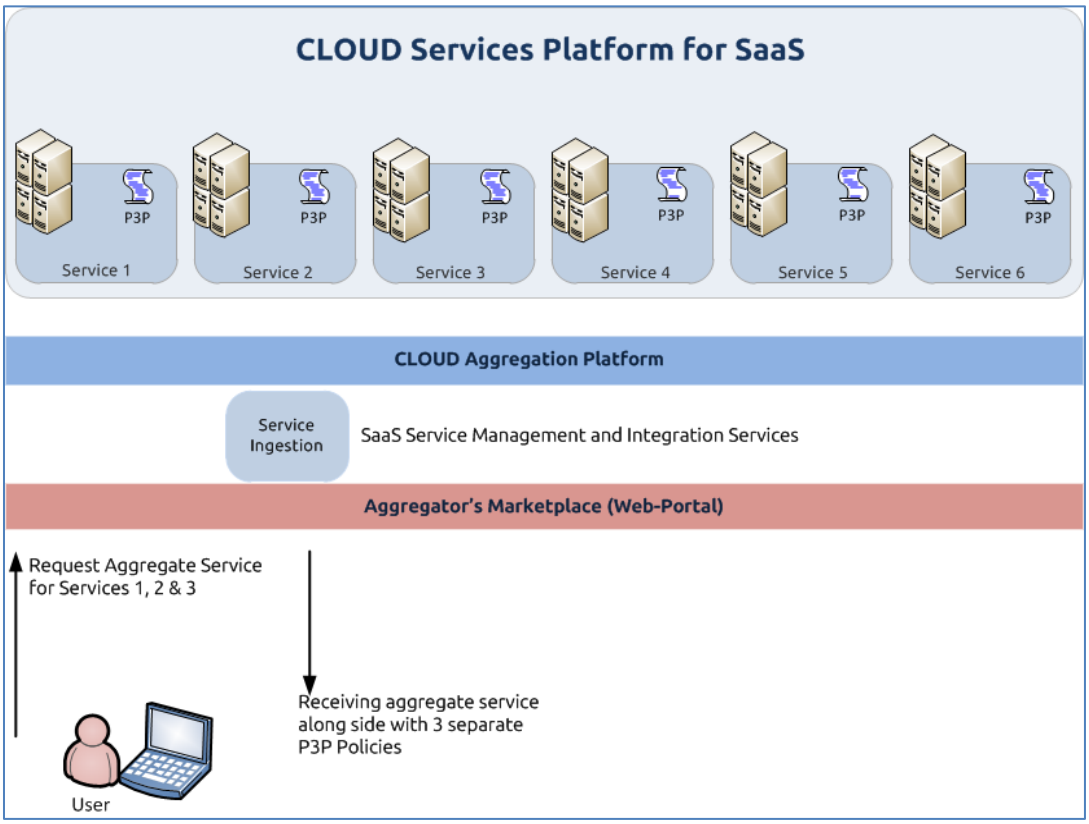


Figure 2: Aggregate service along with multiple P3P policies.

Figure 2 illustrates a sample aggregate platform that provides the user with an aggregate service from three different service providers. It should be noted, however, that this aggregate service will be formed at the request of the user. For example, Cloud aggregators such as Primadesk[4], Rackspace[5], Otixo[6], and Kitedesk[7] offer service integration tools that help aggregate and manage services from different providers (e.g. Gmail, Yahoo, Hotmail, Instagram, Facebook, Dropbox, Google Doc, My box, etc.).

[4] Primadesk https://www.primadesk.com
[5] Rackspace http://www.rackspace.com
[6] Otixo http://www.otixo.com
[7] Kitedesk http://www.kitedesk.com

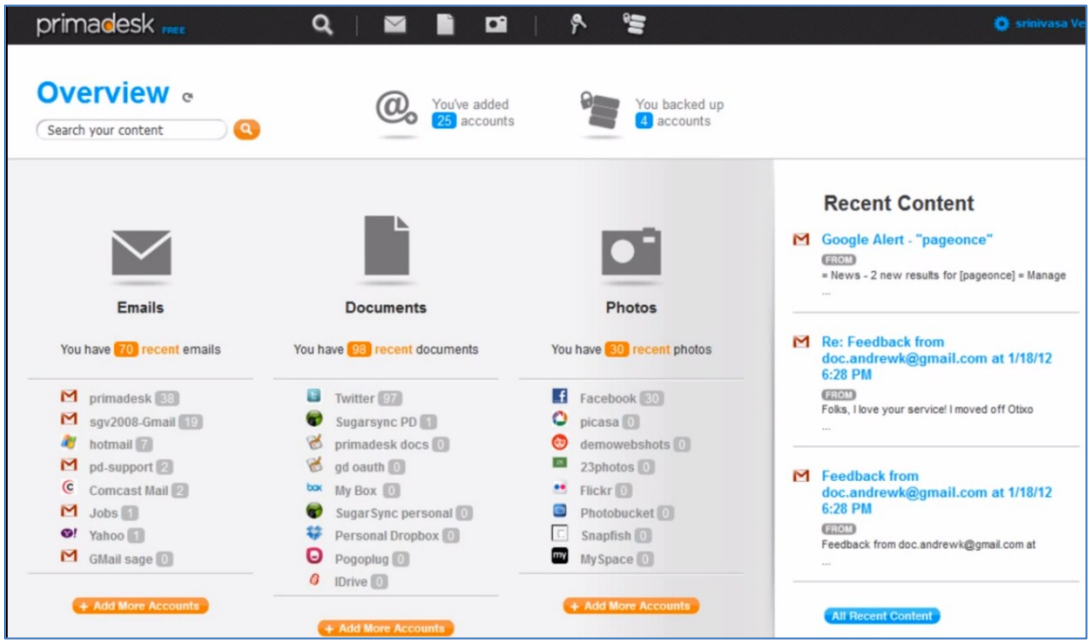


Figure 3: A User's aggregate service dashboard on Primadesk

The Primadesk dashboard shown in Figure 3 lets the user stay connected to the cloud by linking together the contents from all of the user's online accounts. The user can integrate and navigate through the personalized stream of information. For the example shown in Figure 3, the user connects the Primadesk dashboard to email, document and photo accounts through a single stream view of the cloud.

Normally providers present their services in separate domains with a well-defined privacy policy (e.g. P3P). The notifications of privacy policies will be entirely different in the cloud aggregation platform, where multiple service providers adopt different privacy policies for their services. Policy notifications in an aggregation platform will be easier to understand by users, if the aggregate service also has aggregate policy. However, the design of the P3P framework only accommodates one privacy policy file in a single HTTP request. Although is it possible for service providers to aggregate privacy policies offline, this method will only be appropriate if those service providers have appropriate machine readable policies such as P3P.

However, the expiry element in a machine readable policy such as P3P language, states how long the policy reference file or policies remains valid. In order to keep track of the validity when merging the policies, we are considered a real-time merge to reduce human error.

1.2 Thesis Objective

This thesis aims to apply a formal semantic approach and consistency constraints to build a real-time P3P aggregate system. The concept will aid service providers in an aggregation platform to merge and create a consistent P3P policy for service consumers. The first objective of this thesis is to find and resolve inconsistencies in the P3P language, and to properly summarize and categorize the privacy communication of P3P user agents. After identifying the inconsistencies, our next goal is to create merger rules or constraints for the P3P 1.1 specifications. The constraints will help create a new valid policy in an aggregation platform.

Our next objective of this thesis is to explore the usage of our policy aggregate system. Specifically, we aim to bridge policy differences in composite services or aggregate services in the cloud. The concept will not only serve the purpose of applying machine enabled privacy policies (P3P) to service providers in the cloud, but also recognize the basic task of identifying inconsistencies in privacy policies.

The final goal of this thesis is to implement the concept in a way that supports the idea, and propose an account of the status of related research on P3P and directions for future research.

1.3 Thesis Contributions

In this thesis, we propose a concept and framework to merge P3P policies of cloud service providers using an aggregation platform to serve the end users. The proposed concept will also detect inconsistencies in P3P policies and formally verify the merged P3P policies. The following are the major contributions of this thesis:

- We use the P3P language to encode service providers' privacy policies in a cloud-computing environment and we propose a concept and a framework that uses the three-way principle for aggregating P3P policies (chapter 6).

- We extend the current P3P syntax using the concept of relative keys to set up a link between purpose, recipient and data items. We propose a relative data key concept to P3P in (chapter 5).

- We propose P3PMatch, an algorithm for matching (mapping) elements in two or more privacy policies (chapter 5).

- We carry out a formal relational modeling in (chapter 7), to find and solve inconsistencies in P3P using the Alloy[8] model checker.

- We implement the concept that performs a syntactic and semantic merge on different P3P policies in a cloud aggregation platform (chapter 6 and 7).

1.4 Thesis Outline

We organized the rest of this thesis as follows:

Chapter 2 provides background information on policy languages including definitions and categories of related terms in the privacy policy topics. We also give some introductory information on P3P and user agents, including the syntax and semantics of the language, and discuss the P3P framework used in client/server architectures. Then, we present an overview of cloud computing, specifically the Software as a Service (SaaS) model. We also discuss an overview of the architectural design of the SaaS model and the aggregation platform for SaaS.

Chapter 3 discusses the related research on P3P privacy policies. In particular, it focuses on the status of the language, and the recent work on formal semantics and inconsistencies about the P3P language. We present an overview of some recent P3P user agents and techniques on composite policy in web services.

Chapter 4 describes the merging strategy and elaborates on the complete matching and merging process required for aggregating multiple P3P policies. This chapter states some desired merging strategy features, which we address in Chapters 5 and 6.

[8] Alloy is a relational model language for describing structures and a tool for exploring them as well. http://alloy.mit.edu/alloy/

Chapter 5 presents the proposed P3PMatch algorithm. In particular, this chapter focuses on introducing the relative data key notion, which consists of a pair (data-purpose and data-recipient) of data items. The notion semantically enhances the purpose and recipient information about a data item.

Chapter 6 discusses the concept proposed for merging P3P in a cloud aggregation platform. This chapter also presents the semantic merging constraints and an aggregation framework for P3P in the cloud.

Chapter 7 describes the software implementation of the P3P aggregation concept. This tool performs the validation test, matching processes, and performs a syntactic and semantic three-way merging operation of P3P policies.

Chapter 8 presents the conclusion, summarizes the contributions and implications, and proposes several future research directions.

Chapter 2

Background

To get a proper understanding of the P3P language and cloud computing services, this chapter provides basic definitions, and background information regarding the P3P policy language. In order to understand the P3P merger system, we give an overview of the predefined P3P vocabularies, the P3P user agents, and the cloud aggregation platform.

2.1 Basic Definitions

In this section, we explain below, the common concepts and terms required to understand the research context of this thesis.

2.1.1 Privacy

A.F. Westin [80] described privacy as "the ability for an entity to control how, when, and to what extent personal information is communicated to other entities." R. Agrawal et al. [4] described privacy as "the right of the users to decide for themselves when, how, and to what extent information about them will be communicated to other users."

2.1.2 Privacy Policy

Kumaraguru et al. [46] describes the privacy policy as "the formalization of an organization's privacy promises in the form of a policy statement." Nigusse et al. [58] later explained that the privacy policy has two different contextual meanings. They explained in the first context that, "the privacy policy can specify the user's private information that providers collects and the rules that may apply to the information." In the second context, "the privacy policy must specify the possible conflicts that may arise between users and service providers." In this thesis, we categorize both defined policies to have the same meaning.

2.1.3 Policy Language

The Policy language defines the standardized syntax and semantics needed to create and compare the user's privacy preferences with the provider's policies. These syntax and semantics make the privacy policies more machine-readable and enforceable [46].

PRIVACY AREAS	USED BY	PRIVACY LANGUAGES
Access Control Language (ACL)	Service providers	SAML, XACML, XACL
	Clients	XACML
Web	Service providers	P3P
	Clients	APPEL, XPref
Enterprise		CPExchange, PRML, E-P3P, EPAL, DPAL

Table 1: Classification of the Privacy Policy Language **[46]**

Kumaraguru et al. [46] describes various privacy policy languages and arranged those privacy languages in categories. They organized the privacy policy languages according to the area in which the languages are used. Table 1 illustrates a summary of how those privacy languages are classified.

2.1.3.1 Policy Language Requirements

Policy languages are important tools in any privacy assured infrastructures. Machine-interpretable languages have a major advantage over natural languages. Bournez et al. [17] argues that the policy language tools are not designed properly, because they should allow automated negotiation, composition, and enforcement of different privacy policies. The following are the few requirements enumerated in [17] to develop standard privacy policy languages.

- *Measurability* – To fulfill this property, the privacy language should allow one to confirm or verify the policy, and to test if the policy conforms to the instructions of the language.

- *Unified Model* –The Data handling policies are closely related to the access control policies. Therefore, a unified model is a key success factor that should be present in the policy language.
- *Semantic compatibility with P3P* – Other policy languages often should be compatible with the semantics of the P3P language.
- *Revocability* – It must be possible for any policy maker to revoke the data handling policy, the same way it is possible to revoke a credential related to the access control policy.
- Transparency – A service provider's privacy policies must be able to express the data flow of collecting user data. Information should be provided on how they are transferred, and how they are kept.

2.2 Platform for Privacy Preferences (P3P)

2.2.1 Introduction to P3P

The W3C (World Wide Web Consortium) has defined P3P in a pioneer web privacy project in 1995; the project is called the Platform for Internet Content Selection (PICS) [66]. The PICS privacy project started from a research laboratory at the Massachusetts Institute of Technology (MIT). The PICS project was the early standard name rating system for labeling web contents. The W3C later transformed the idea behind the project from rating and labeling the contents of web sites (PICS) into communicating websites' data handling practices (P3P).

The P3P language is an XML specification defined by the W3C. The P3P platform is the protocol that allows policy file generated from a website's privacy statement when encoded with the P3P language [43,54,62]. Some P3P editors (e.g. JRC Policy Workbench, IBM P3P Editor, P3Pwriter.com, or P3Pdeveloper.com) generate the P3P files that describe the privacy practices of a web site.

We refer to the privacy policy encoded with the P3P language as a P3P policy [24,25]. The W3C proposed the framework to automate the privacy policy verification processes. Figure 4 describes the workflow of the P3P framework. The following are the two requirements of the

P3P framework. (1) The P3P specification requires that websites should publish their privacy policies using the P3P vocabulary [24]. (2) The P3P specification requires the users to specify their privacy preferences using a preference language called A P3P Preference Exchange Language (APPEL) [26].

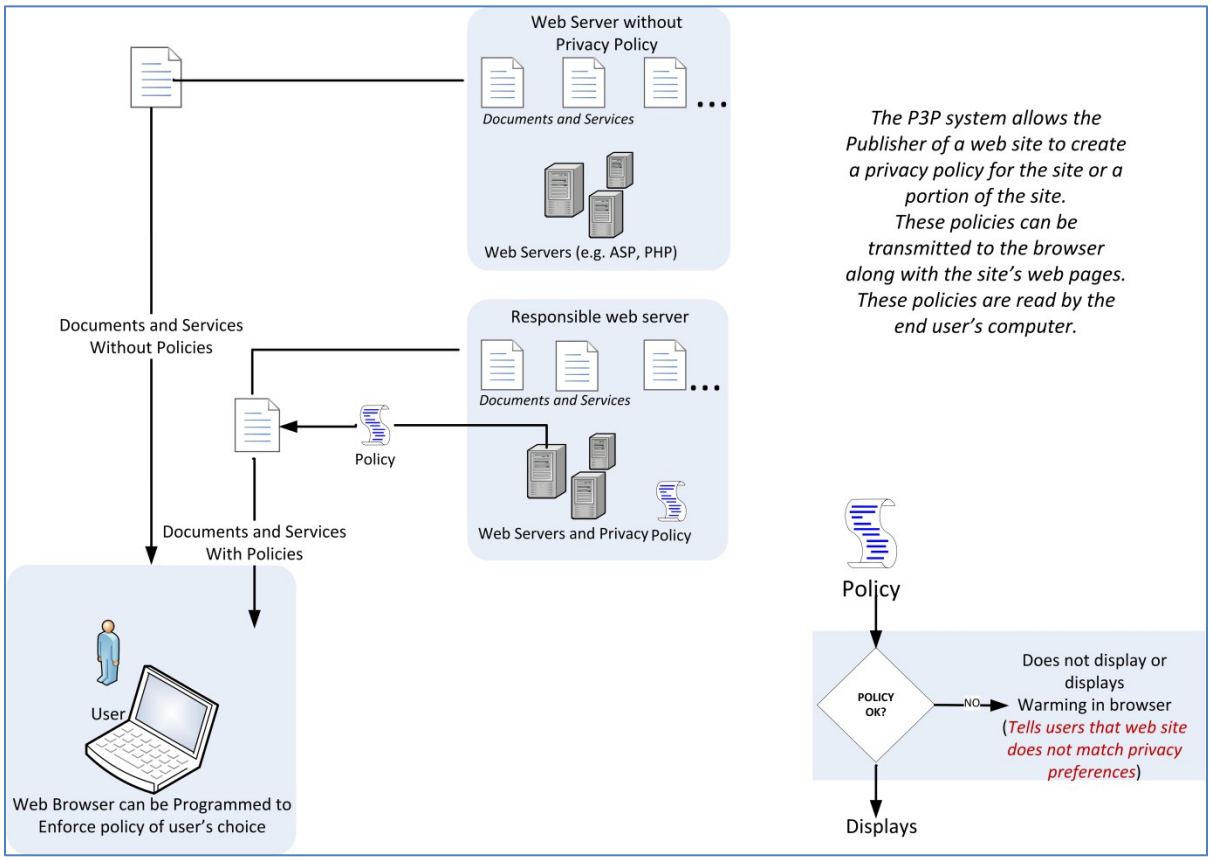


Figure 4: Framework for P3P Privacy Policies **[24]**

2.2.2 Retrieving P3P Policies

The standard Hypertext Transfer Protocol (HTTP) is the foundation of data communication for the P3P policies. The web browser uses the HTTP protocol to communicate with servers over the Internet, which enables the retrieval of P3P policies on the web browser. However, the web browsers or proxy servers parse and compare the retrieved P3P policies with the user's privacy preferences. Figure 5 illustrates the complete HTTP transaction between the web browser and a P3P enabled website. In the next section, we will explain the P3P reference and the policy file.

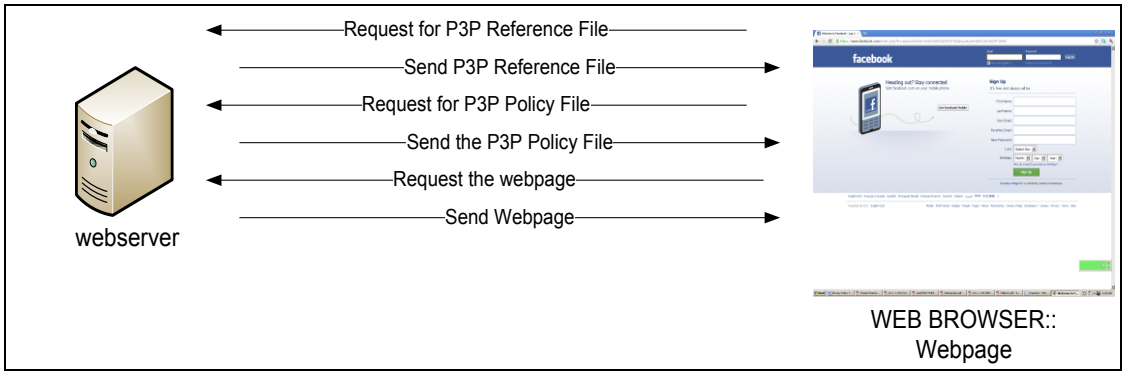


Figure 5: Retrieving a P3P policy from the server

2.2.3 P3P Policy Reference File

```
1.  <META xmlns="http://www.w3.org/2002/01/P3Pv1">
2.      <POLICY-REFERENCES>
3.          <EXPIRY max-age ="172800"/>
4.          <POLICY-REF about="/w3c/p3policy.xml">
5.              <INCLUDE>/*</INCLUDE>
6.          </POLICY-REF>
7.      </POLICY-REFERENCES>
8.  </META>
```

Figure 6: An Example of P3P Policy Reference **[24]**

The P3P policy reference file is an XML file that points to a Uniform Resource Identifier (URI) where specific P3P policies on the web server are located. The policy reference file makes it easier for a user agent to find the privacy policies that applies to the specific web page. Figure 6 illustrates an example of the P3P policy reference file; Line 4 represents the *Policy-Ref* element that specifies the address for the P3P policy. To locate the site address (Figure 6; line 4) of the policy reference file, one can use the specified ways in the P3P specification [24].The following are the three specified ways of retrieving the P3P policy from the web server.

a) A predefined address path helps to find the policy reference file (…/W3C/P3P.xml).
b) Either through the website's HTML link tag, or XHTML link tag.
c) Through an HTTP header.

2.2.4 P3P Syntax and Semantics

P3P policy is a deployable subset of XML; the W3C as well as major corporations such as Microsoft, Yahoo and IBM supports the language [7,8,24]. A P3P policy consists of machine-readable assertions [24] about the website's privacy practices written in a standard privacy vocabulary. There are two main classifications of P3P elements: the Body element and the Root element.

<POLICY Name, Discuri, opturi **>**

- **<ENTITY>** {<Data-Group>; Entity Description} **</ENTITY>**
- **<ACCESS>** <non-ident | All | Contact-and-Other| Ident-contact | Other-ident | none> **</ACCESS>**
- **<DISPUTES- GROUP>**
  - **<DISPUTE>** <Resolution-type = *{Service | Independent | Court | Law }*
    Service = "url", Verification = "url", Short-Description = "……">
    Extension
    [Long description]
    [Image] <img src = "", width = "", height = "" Alt = "">
    [Remedies]
    - **<REMEDIES>**
      - {<CORRECT/> | <MONEY/>|<LAW/>}
- **<STATEMENT>**
  - **<EXTENSION>**
    - **<STATEMENT-GROUP-DEF>** <id, Consent {opt-out | opt-in | Always}>
  - **<CONSEQUENCE>**
  - **<PURPOSE>**
    - <CURRENT>
    - <ADMIN> *{opt-out | opt-in| always}*
    - <TAILORING> *{opt-out | opt-in| always}*
    - <PSEUDO-ANALYSIS> *{opt-out | opt-in| always}*
    - <PSEUDO-DECISION> *{opt-out | opt-in| always}*
    - <INDIVIDUAL –ANALYSIS> *{opt-out | opt-in| always}*
    - <INDIVIDUAL-DECISION> *{opt-out | opt-in| always}*
    - <CONTACT> *{opt-out | opt-in| always}*
    - <HISTORICAL> *{opt-out | opt-in| always}*
    - <TELEMARKETING> *{opt-out | opt-in| always}*
    - <OTHER-PURPOSE> *{opt-out | opt-in| always}*
  - **<RECIPIENT>**
    - <OURS ; DELIVERY; SAME; OTHER RECIPIENT; UNRELATED; PUBLIC>
  - **<RETENTION>**
    - **<NO RETENTION | STATED PURPOSE | LEGAL-REQUIREMENT| INDEFINITELY| BUSINESS PRACTICES>**
  - **<DATA GROUP>**
    - **<DATA>** <optional YES | NO or ref = " ">
      - **<CATEGORY>**
        - (<physical/>, <online/>, <uniqueID/>, <purchase/>,<financial/>,<computer/>, <navigation/>, <interactive/>, <Demographic/>, <content/> ,<state/>, <Political/>, <health/>, <preference/>, <location/>,

Figure 7: Outline of P3P Privacy Policy Syntax and Semantics

In the P3P language, each element (tag) may have lists of associated attributes with different values. The P3P elements and attributes are usually not variables but they have meaningful

predefined names specified as P3P vocabularies [24]. Figure 7 presents an overview of the P3P syntax; the main elements contained in a P3P policy are all in the *policy* element. The *policy* element is the root element, which contains the combination of elements such as *Extension, Entity, Access, Disputes-Group,* and *Statement* elements.

2.2.4.1 Policies Element

The POLICIES element is the root element of a P3P file, if the file contains one or more *Policy* elements. This provides performance optimization: a single HTTP request can collect many P3P policies in a single POLICIES, improving network traffic and caching. Policies can contain the following elements [24].

- *Xml-Lang* - in the case of an HTTP request using "Accept-Language" property in the header tag.
- *Expiry* – optional expiration time length. Possible to define the absolute date of expiration or set relatively maximum age of policy in seconds.
- *Dataschema* – defines embedded Dataschema.
- *Policy* – Policies element encapsulates a set of policy elements.

2.2.4.2 Policy Element

The POLICY element contains the complete privacy policy of an entity. The POLICY element must contain one ENTITY element to identify the legal entity responsible for the privacy practices. The element also contains one ACCESS element, one DISPUTES-GROUP and one or more STATEMENT elements. Figure 8 describes the POLICY element and its required child elements [24]:

- *Name* (mandatory attribute): The *Name* attribute is the name tag given to the policy, used as a fragment identifier to be able to reference the policy.
- *Discuri* (mandatory attribute): The *Discuri* attribute is the URI of the natural language privacy statement of the service provider.

- *Opturi* (not mandatory): The *Opturi* attribute is the URI of instructions that users can follow to request or decline to have their data used for a particular purpose (opt-in or opt-out).

```
<POLICY name = " " discuri = " " opturi = " " [xml-Lang]>
        <ENTITY> … </ENTITY>
        <ACCESS> … </ACCESS>
        <DISPUTES-GROUP> … </DISPUTES-GROUP>
        <STATEMENT> … </STATEMENT>
</POLICY>
```

Figure 8: Policy Element

2.2.4.3 Entity Element

The ENTITY element provides the precise description of the provider responsible for the policy. It contains the provider's name and some of the fields defined in the business dataset of the base data[9] schema such as the contact address, and e-mail address[24,39]. As shown in Figure 9, a DATA GROUP element, which contains DATA element, extends the datatype reference mechanism.

```
<ENTITY>
   <DATA-GROUP>
      <DATA ref= "#business.name">XYX</DATA>
      <DATA ref= "#business.online.email">abc@ee.com</DATA>
   </DATA-GROUP>
</ENTITY>
```

Figure 9: An Example of Entity Element

2.2.4.4 Access Element

The ACCESS element indicates privilege given to an individual or any other user to access private information collected. The ACCESS element must contain only one of the following predefined values:

- *Nonident*: The web site does not collect identifying data.
- *All*: Access is given to all identified data.
- *Contact-all-other*: Access is given to identified online and physical contact information as well as to certain other identified data.

[9] The P3P 1.1 specification; Business Data, http://www.w3.org/TR/P3P11/#business_data

- *Ident-contact*: Access is given to certain identified online and physical contact information (e.g., Users can access things such as home address).
- *Other-Ident*: Access is given to certain other identified data (e.g., users can access thing such as their online account charges).
- *None*: No access is given to access identified data.

```
<ACCESS>
    <IDENT-CONTACT/>
</ACCESS>
```

Figure 10: An Example of Access Element

2.2.4.5 Disputes-Group

A DISPUTES-GROUP element contains one or more DISPUTES elements in a policy. It describes dispute resolution procedures that the users may follow to resolve disputes about the service providers' privacy practice. The DISPUTES element requires a resolution type (*Service, Independent, Court, or Law*), and the URI of the service to resolve the dispute page, the URL for verification, and should include the REMEDIES element, and the long and short description of the dispute.

2.2.4.6 Statement-Group-Definition Element

The STATEMENT-GROUP-DEF element indicates the statement group tag that contains all the associated statement elements. It collects the statements elements together, based on a certain typical data representation.

2.2.4.7 Statement Element

The STATEMENT element is a container that groups together a RETENTION element, a RECIPIENT element, a DATA-GROUP element, a PURPOSE element, and optionally a CONSEQUENCE element and one or more extensions. The STATEMENT element can contain a NON-IDENTIFIABLE element; which signifies that there is no data collected regarding the STATEMENT. However, the element will anonymize data which are collected. The websites that prefer to disclose separate purposes and other information for each category of data they collect can do so, by creating a separate STATEMENT for each DATA type.

2.2.4.8 Non-Identifiable Element

The NON-IDENTIFIABLE element signifies that, either only anonymized data, or no data will be collected in the STATEMENT element.

2.2.4.9 Purpose Element

The PURPOSE element describes the purpose or the main reasons for collecting private data. It must contain one or more inner purpose elements relevant to the collected data.

- *Opt-out*: The tag requires explicit user rejection before removing the purpose or recipient options from their agreed policies.
- *Opt-in*: The tag requires explicit user consent before adding the purpose or recipient options to their agreed policies.
- *Always*: The tag does not require any rejection or consent from users; it means that users cannot opt-out or opt- in for data collection with this purpose.

There are 12 predefined purpose values used in the P3P Language, which are mostly self-explanatory:

- *Current*: This purpose value represents data collected for the completion and the support of activities on websites, which data is required. The CURRENT element does not have a *required* option.

  For example, if a bus-scheduling website requires personal information such as home address and in return, the user gets the bus schedule, then the website has collected the address for current purposes.
- *Admin*: The purpose value, admin, depicts the data collected for the purpose of technical support, or the website administration.

  For example, a website may collect data about users' devices to improve the bandwidth optimization, and the communication systems. Thus, data collected for its purpose is tagged admin.
- *Tailoring*: This purpose value allows private data to be collected for customizing a user's need for a one-time visit to the website, which is not to be used for future purposes.

For example, if a website such as an online store collects information regarding the content of a user's online cart, then the information will be used to tailor the website to other relevant information that the user may prefer.

- *Develop*: Data is collected for research and development purposes alone, which does not include research on personal information of users.

  For example, a website may collect information regarding the sections of the website users' visit often, or the browsing history of the user on the website.

- *Pseudo-analysis*: Data collected is tied to a pseudonymous identifier to build reports for the purpose of research, analysis, and reporting on the website. The website collects only data that are not identifiable.

  For example, if the website wishes to understand the interest of users in different sections of the site, information tied to a pseudonymous identifier will be collected for the purpose of pseudo-analysis.

- *Pseudo-decision*: Data collected is tied to a pseudonymous identifier to generate reports for determining the habits, interests or other characteristics of the individual to make decisions that directly affect users.

- *Individual-analysis*: Data collected is used to determine the habits, interests or other characteristics of the user and to combine the private data with other identifiable data for the purpose of research, analysis and reports.

  For example, if an online store collects information to analyze how online shoppers make offline purchases in the physical store, data collected is used for the purposes of individual analysis.

- *Individual-decision*: Data collected is used to determine the habits, interests or other characteristics of the individual and to combine the private data with other identifiable data to make a decision that directly affects the individual.

  For example, an online store suggests items a visitor may wish to purchase based on items he has purchased during previous visits to the website.

- *Contact*: Data collected is used to contact users for the purpose of product or service promotion. This purpose value is only used for communication media other than voice telephone or web advertisements embedded in sites the user is visiting.
- *Historical*: Data collected is archived or stored for the purpose of preserving social history as governed by an existing law or policy. This law or policy must be referenced in the DISPUTES element.
- *Telemarketing*: Data collected is used to contact the users via a voice telephone call for promotion of a product or service. This does not include a direct reply to a question or comment, or customer service for a single transaction.

2.2.4.10 Retention Element

The RETENTION element indicates how long the entity will keep collected information; it describes the category of retention value that applies to the data referenced in the STATEMENT element. The RETENTION element must contain one of the following pre-defined values: *no retention, stated-purpose, legal-requirement, or business-practices, indefinitely.*

- *No-Retention* – Information collected from users are not retained for more than a brief period of time required for a single online interaction.
- *Stated-Purpose* – Data collected is retained to meet the stated purpose for which the data were collected.
- *Legal-Requirement* – Information collected is as required by law or liability under applicable law. The retention period is longer because of a legal requirement or liability.
- *Business-Practices* – Information is retained under a service provider's stated business practices.
- *Indefinitely* – Information is retained for an indeterminate period of time. The recipient value when a retention value is indefinitely MUST be public fora.

2.2.4.11 Recipient Element

The RECIPIENT Element indicates the entity that collects data and the agents that shares collected data. The entity must declare their recipients by using one or more of the following pre-defined values: *Public, Other-party, Ours, Delivery, Same*, and *Unrelated*.

- *Public* – Public fora such as bulletin boards, public directories, marketplace, or commercial CD-ROM directories.
- *Other-Party* – Legal entities following different practices: legal entities that are constrained by and accountable to the original service provider, but may use the data in a way not specified in the service provider's practices.
- *Ours* – Ourselves or entities acting as our agents, or entities for which we are acting as an agent. E.g. some specific department of our company.
- *Delivery* – Delivery services possibly following different practices: legal entities performing delivery services that may use data for a purpose other than completion of the stated purpose. This should be used for delivery services whose data practices are unknown.
- *Same* – Legal entities following our practices; legal entities who use the data on their own behalf under equitable practices e.g. business partner with equitable policies.
- *Unrelated* – Unrelated third parties; legal entities whose data usage practices are not known by the original service provider.

2.2.4.12 Data Group Element

The DATA-GROUP element encapsulates data types. Those data types specify which data is collected. The element must include one or more DATA elements.

- *Data*: The element is used to specify the type of information collected; it has two attributes. The required attribute is *ref* (ref: identifies the data collected by name), for example <DATA ref="#user.name.given"/>. The other attribute is *optional*; it indicates whether or not the data collection is optional, the required values are (Yes or No).

In the following figures, Figure **11** and Figure **12**, there are examples of a natural language privacy policy translated into a full P3P privacy policy.

> At Blue Yonder Airlines, we care about your privacy. When you browse thro ugh our site, we collect information on the efficiency and working of our We bsite. This information includes the number of times a web page is accessed, the browser used, and paths taken when moving through the Web site. We pu rge this information yearly.
>
> We also collect your zip code but will prompt you to enter it. With your per mission, this information is aggregated with information collected from all vi sitors to our Web site and used for market analysis. This information might be provided to third parties. Once prompted for your zip code, you will not b e prompted again if your browser privacy preferences allow cookies.
>
> Blue Yonder Airlines use cookies (small files on your computer) to store wh ether you have entered your zip code or declined to do so. We access this inf ormation each time you visit our site so that you are only prompted the first t ime you visit our site.

Figure 11: Sample natural language policy[10]

[10] Microsoft Library : How to deploy P3P Privacy Policies on Your Web Site; http://msdn.microsoft.com/en-us/library/ms537341(v=vs.85).aspx

```
<POLICY xmlns="http://www.w3.org/2000/12/p3pv1"
    discuri="http://www.blueyonderairlines.com/ourprivacypolicy.html"
    opturi="http://www.blueyonderairlines.com/optin.html">
 <ENTITY>
  <DATA-GROUP>
   <DATA ref="#business.name">Blue Yonder Airlines</DATA>
   <DATA ref="#business.contact-info.postal.street">3456 Main St.</DATA>
   <DATA ref="#business.contact-info.postal.city">Tampa</DATA>
   <DATA ref="#business.contact-info.postal.stateprov">Fl</DATA>
   <DATA ref="#business.contact-info.postal.postalcode">77062</DATA>
   <DATA ref="#business.contact-info.postal.country">USA</DATA>
   <DATA ref="#business.contact-info.online.email">molly@blueyonderairlines.com</DATA>
   <DATA ref="#business.contact-info.telecom.telephone.intcode">1</DATA>
   <DATA ref="#business.contact-info.telecom.telephone.loccode">800</DATA>
   <DATA ref="#business.contact-info.telecom.telephone.number">5550158</DATA>
  </DATA-GROUP>
 </ENTITY>
 <ACCESS><nonident/></ACCESS>
<STATEMENT>
  <PURPOSE><admin/><develop/></PURPOSE>
  <RECIPIENT><ours/></RECIPIENT>
  <RETENTION><stated-purpose/></RETENTION>
  <DATA-GROUP>
    <DATA ref="#dynamic.clickstream.server"/>
    <DATA ref="#dynamic.http.useragent"/>
  </DATA-GROUP>
</STATEMENT>
<STATEMENT>
  <PURPOSE><pseudo-analysis required="opt-in"/></PURPOSE>
  <RECIPIENT><other-recipient/></RECIPIENT>
  <RETENTION><indefinitely/></RETENTION>
  <DATA-GROUP>
    <DATA ref="#user.home-info.postal.postalcode">
      <CATEGORIES><demographic/></CATEGORIES>
    </DATA>
  </DATA-GROUP>
</STATEMENT>
</POLICY>
```

Figure 12: The corresponding P3P policy derived from the preceding natural language policy[11]

[11] Microsoft Library : How to deploy P3P Privacy Policies on Your Web Site; http://msdn.microsoft.com/en-us/library/ms537341(v=vs.85).aspx

2.3 Tree Model Structured Data

The organization of XML expresses the syntactical structure of documents, and the most important components of that structure are *elements*. Luuk Peters [50] stated that, "Considering XML documents a structured data tree, two trees can be compared together, equal nodes and equal sub-trees in both XML trees can be matched. The nodes and sub-trees that aren't matched are the difference between the two XML trees." In this thesis, we also consider P3P as a structured data tree.

Figure 13 illustrates sample XML code; the root element is a node-indicating <student> and its closing tag </student>. The root node has one attribute associated with it: *id* with the value *7456*. The content of the root element is the subject node, the score node, and the completed node. We will map XML documents to trees according to the following rules:

```
1.  <student id= "7456">
2.  <subject>mathematics</subject>
3.      <score>
4.         90
5.      </score>
6.  <! -- Comment to say the student passed -->
7.  <Completed/>
8.  </Student>
```

Figure 13: Sample XML code

1. The children of a node *n* are the elements and character data contained in the element corresponding to *n*.
2. The content of a node corresponding to an element has a type value of the element, the student field is initialized to the name of the element (e.g. "Student" from the example above), and the list of attributes and values are initialized with the attributes and values of the element. The text field is empty.
3. The content of a node corresponding to subject data has a type value of mathematics, and the text field is initialized with the character data.
4. Content that is not character data or elements (such as comments) is not mapped to the tree.

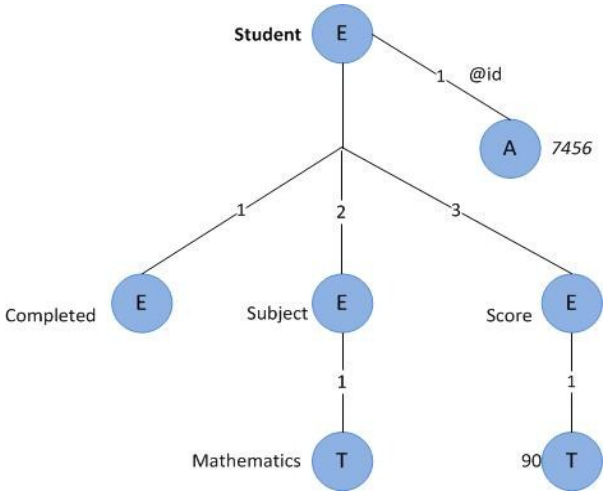


Figure 14: The Tree Corresponding to the XML code in Figure 13

Nodes, whose type field has the value element, are *element nodes*, and those whose type field is text are *text nodes*. Figure 14 shows the tree corresponding to the XML document in figure 11.

Type	Name	Text	Attribute and Values
Element (root)	Student	Null	Id = 7456
Element (2)	Subject	Null	Null
Element (3)	Score	Null	Null
Element (1)	Completed	Null	Null
Text	Null	90	Null
Text	Null	Mathematics	Null

Table 2: Table explaining XML document in Figure 13

2.4 P3P User Agents

A P3P User Agent is a tool or service that translates websites' P3P policies into a complete human readable format. It also checks the website's policies against the user's preferences for privacy warnings [1,9,21,23,24]. The Microsoft Internet Explorer 6 (IE6) and the Netscape Navigator 7 were the first web browsers to adopt the P3P functionality. "A user agent's action can range from showing a simple information message to blocking the requested web site [58]." The P3P user agents assist users to specify their privacy preferences and can be classified into informative-only or automatic user agents; the former simply

inform "whether website's privacy policies match with user privacy preferences or not [58]." However, the P3P user agent can automatically take certain actions such as blocking cookies, or blocking access to certain web sites. The AT&T lab developed a P3P user-agent called the Privacy Bird [7,26,22,25]. Privacy Bird is a privacy extension for the Microsoft IE6; it allows users to specify privacy preferences. "A P3P compatible user agent might use the APPEL rule sets to express the user's privacy preferences. However, a user agent can be compliant without implementing the APPEL rule sets [58,21,7]." A P3P user agent is a browser extension, a browser plug-in, a standalone user application, or any mobile platform that can run a browser.

2.5 A P3P Preference Exchange Language (APPEL)

2.5.1 Introduction of APPEL

The W3C specifically developed a language that complements the P3P1.0 language by encoding the users' privacy preferences; this language is called APPEL [26]. An APPEL-based policy is the P3P counterpart on the client-side, used by the P3P user agents to automatically make privacy decisions on behalf of the user. The preference-rules (ruleset) group the privacy preferences specified by users. Figure 15 illustrates how a user can create the APPEL ruleset either manually or by interacting with an APPEL enabled user agent, which in turn automatically generates the APPEL ruleset [39].

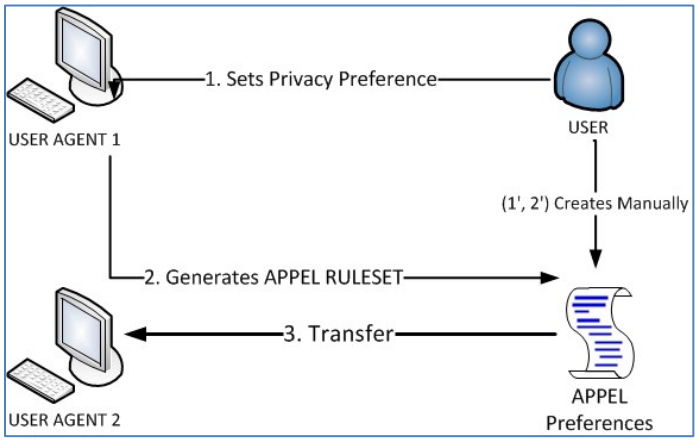


Figure 15: How users can create APPEL Preferences **[26]**.

The APPEL policy consists of multiple RULE elements, which specify user requirements for comparing P3P policies. *Behavior* attributes in the APPEL language specify the action taken upon successful matching of a RULE; the three standard behaviors supported are *Request, Limited, and Block*. APPEL has a rule-based syntax encoded in an XML based language [26]. The following attributes describe the behavior output:

- *Request* – The provided P3P policy that covers the URI is acceptable (the resource at the URI should be accessible).
- *Limited* – The provided P3P policy is partially acceptable (the resource at the URI should be accessible with limitations).
- *Block* – The provided P3P policy is not acceptable (access to the resource may be blocked).

2.6 Cloud Computing

2.6.1 Introduction to Cloud Computing

Cloud computing is the recent revolution in the Information Technology (IT) industry, following the personal computer revolution and the internet revolution. Chang et al. [20] pointed out that, "cloud computing not only matters for the IT industry, but also for technology consumers because its services are directly accessible to consumers over the internet." Tim Grance and Peter Mell of the Information Technology Laboratory (ITL)[12] at the NIST (the US National Institute of Standards and Technology) defined cloud computing in a NIST special publication 800-145 [55].

*"*Cloud computing is a model for enabling convenient, on-demand network access to a shared pool of configurable and reliable computing resources (e.g., networks, server storage, applications and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction."[55]

[12] ITL at the NIST promotes the U.S economy and public welfare by providing technical leadership for the nation's measurement and standard infrastructure. ITL develops tests, test methods, reference data, proof of concept implementations, and technical analysis to advance the development and productive use of information technology [3, p.4].

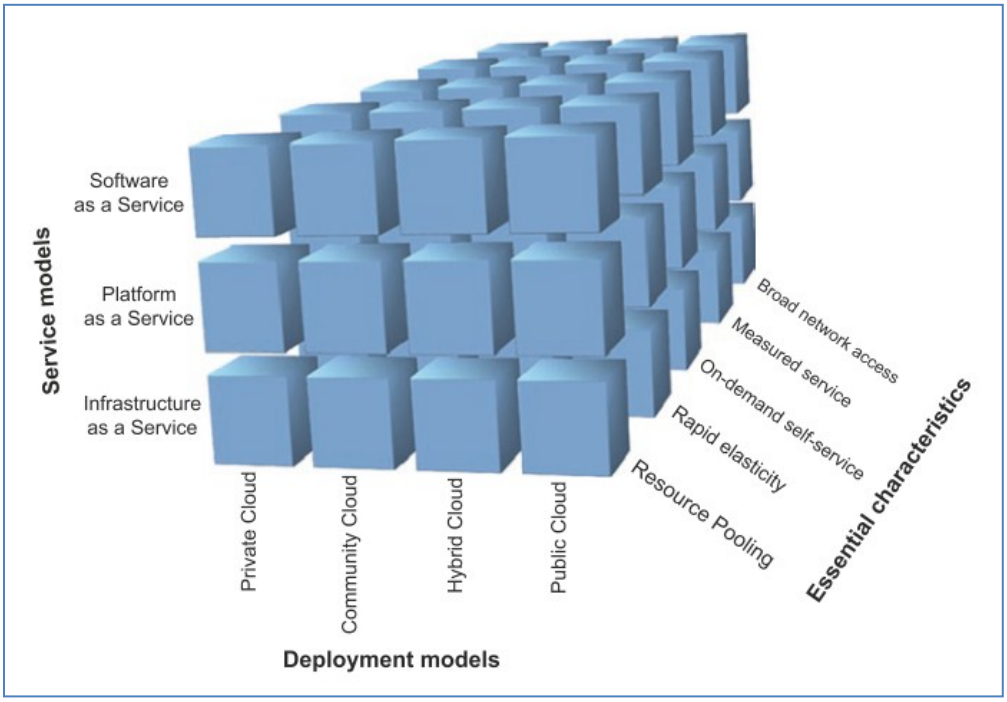


Figure 16: The NIST's Model of Cloud Computing **[12]**

Cloud computing resources are typically exploited by a pay-per-use model and the guarantees are offered by means of a Service Level Agreement (SLA) [81,12]. Cloud computing is not a "one-size-fits-all" technology. Rather it is a continuum of service delivery options, often referred to as Software as a Service, Platform as a Service, and Infrastructure as a Service. Figure 16 illustrates the platforms and models available in cloud computing.

2.6.2 Cloud Service Models

The concept of cloud computing can be classified according to the resources or services offered by the CSP. "Cloud computing can be categorized into three basic service models; Software, Platform and Infrastructure as a Service. These service models describe the degree of control an enterprise will get from the CSP." [20] Figure 17 illustrates a brief comparison of cloud computing models and traditional computing.

2.6.2.1 Infrastructure as a Service (IaaS)

In the Cloud IaaS model, the CSP offers a platform virtualization environment as a service to the enterprise. "The CSP offers processing power, storage, networks, and other fundamental computing resources. The customer is able to deploy and run operating systems and applications but the control given to the customer is only for the operating systems, storage, deployed applications, and possibly limited control of the selected networking components like firewalls" [12,55]. Computing hardware resources described above provide services via the internet and users would typically obtain an instance of the computing or storage resource, and then connect to it remotely through the server available to them. Examples of IaaS are Amazon EC2, Flexiscale etc.

2.6.2.2 Platform as a Service (PaaS)

In the Cloud PaaS model, "The customer is able to write their applications created using programming languages (e.g. Java, python) and tools supported by the provider of the cloud infrastructure. The consumer does not manage or control the underlying cloud infrastructure, but has control over the deployed applications and possibly application hosting environment configurations." [12,55] Examples of PaaS are the Google App Engine, Force.com, etc.

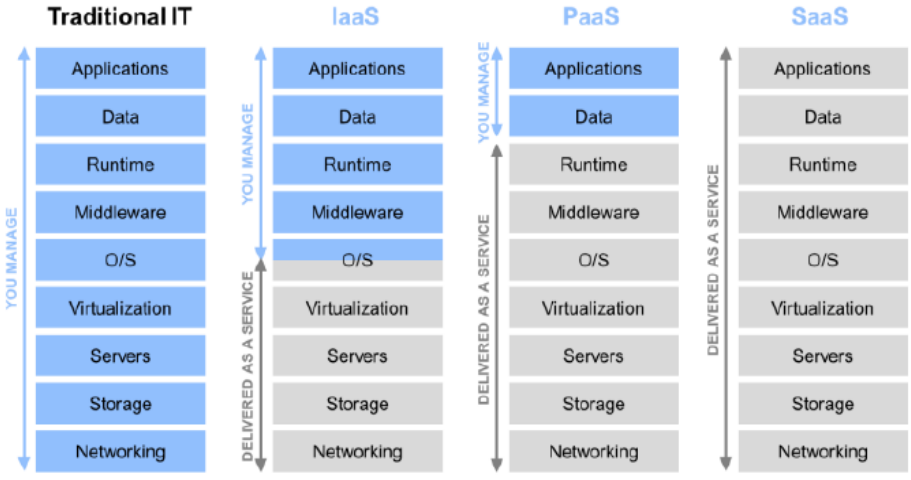


Figure 17: Overview of Cloud Service Models

2.6.2.3 Software as a Service (SaaS)

SaaS is a cloud-computing model for delivering software applications over the internet, "The consumer is provided with the ability to use the service provider's applications hosted on a cloud infrastructure. The consumer is not usually concerned where the applications are running or where the data is stored" [55]. The applications in a SaaS model are accessible from various consumers' devices through a thin generic client interface such as a web browser [49]. Some of the examples of SaaS are online webmail (Yahoo Mail, Gmail, etc.). "The customer does not manage or control the underlying cloud infrastructure, but may be able to set limited user-specific application configuration settings" [55,12].

2.6.3 Cloud Deployment Models

Yang et al. [81] explained that cloud deployment models "can be delivered with four deployment models; public, private, community, and hybrid". These deployment models describe who owns, manages, and is responsible for services. NIST [55] described the deployment models mentioned above, as follows:

2.6.3.1 Public Cloud

In a public cloud, multiple service providers share delivered services or resources provided by a cloud service provider. "The cloud infrastructure is made available to the general public or a large industry group and is owned by a provider selling cloud services" [55]. The public can access resources remotely and quickly. Furthermore, they can only operate on the service they pay for.

2.6.3.2 Private Cloud

In a private cloud, only one private enterprise can use the services and computing resources. "The cloud infrastructure is operated solely for an organization to manage on site or off premise. Resources in this model are dedicated only to the customer."[55]

2.6.3.3 Community Cloud

In a community cloud, several organizations can share the same cloud infrastructure. "The combination of several organizations jointly constructs and share the same cloud

infrastructure as well the same policies, requirements, values, and concerns." An organization or a third party vendor can also manage the cloud infrastructure [55].

2.6.3.4 Hybrid Cloud

Hybrid cloud combines multiple deployment models. "The cloud infrastructure is a composition of two or more clouds (Private, Community, or Public) that remain unique entities but are bound together by standardized or proprietary technology that enables data and application portability (e.g. Cloud burst for load-balancing between clouds) " [55].

2.7 Service Aggregation Model

In this thesis, we consider the model of service aggregation presented in [78] .The diagram in Figure 18 illustrates the high-level architecture of service aggregation in a cloud-computing environment. For this thesis, we extend the types of parties presented in [78]. There are five types of parties in service aggregation in a cloud: users, service providers (SP), service aggregator (SA), aggregation platform, and cloud service provider (CSP).

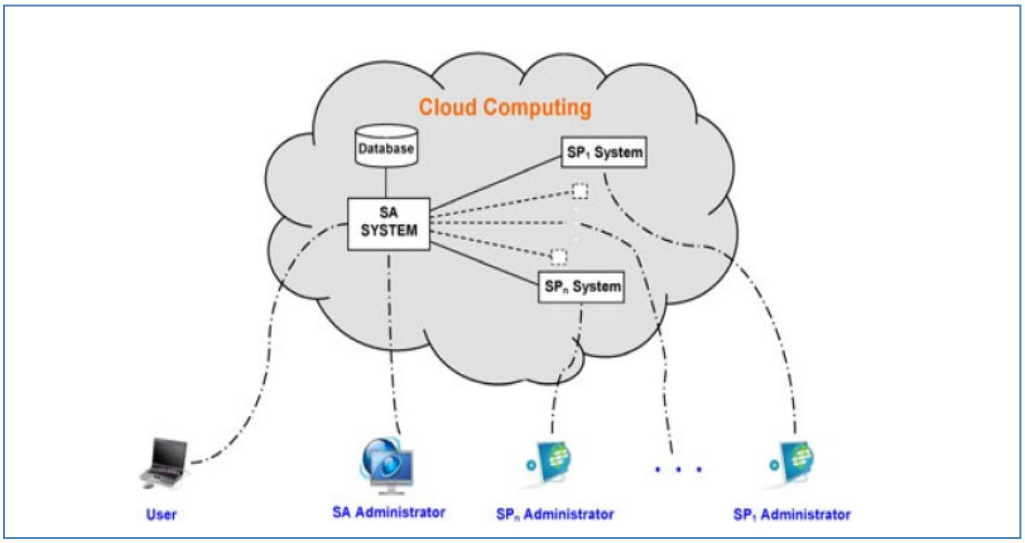


Figure 18: Service aggregation in a cloud-computing environment **[78]**

2.7.1 Cloud Service Provider (CSP)

A cloud service provider is a company, which offers some component of a cloud computing service model: Infrastructure as a service (IaaS), Platform as a Service (PaaS), or Software as a Service (SaaS). The following are examples of CSPs: The Amazon EC2, the Google App Engine, Open stack, GoGrid, and Microsoft Azure.

2.7.2 Cloud Users

Deepa et al. [28] expressed a cloud user, "as a person or entity that makes use of the cloud capabilities and services as opposed to the cloud service providers and cloud providers that supply those services."

2.7.3 Service Provider (SP)

A service provider is an organization or an enterprise that purchases services (Software, Platform, or Infrastructure services) from the cloud provider. They enhance their own services and capabilities by exploiting cloud platforms from cloud service providers. The enhanced services can thus effectively become software as a service [16,28].

2.7.4 Service Aggregator (SA)

The Service Aggregator[13] offers a web portal service or a platform that combines multiple cloud applications and SaaS offerings such as (e-mail, cloud storage, or hosted desktops) with similar characteristics (geographic area, target market, size, etc.) into a single point of access, format, and structure. Users derive the value of aggregate services from cost savings and greater efficiency found from the ability to leverage multiple service providers [68]. The service aggregator works as an agency for service providers to aggregate various services and policies offered by service providers, along with some of its own services and policies to serve users. The service aggregator is an entity that manages the service use, the

[13] A cloud aggregator can also be referred to as a cloud broker; http://h30507.www3.hp.com/t5/Cloud-Source-Blog/Defining-cloud-service-intermediation-and-aggregation-IT-as-the/ba-p/122769.

performance, and service delivery to cloud consumers. Three types of aggregators are recognized.[14]

2.7.4.1 Service Intermediation

The service aggregator enhances a given service by improving some specific capacity and providing value-added services to the cloud consumer. The improvement can be managed access to cloud services, identity management, performance reporting, enhanced security, etc.

2.7.4.2 Service Aggregation

Service aggregation combines and integrates multiple services offered by various service providers into one or more new services; the providers sometimes may be in a different domain from some of the organization's own services to form more complex and powerful services to serve users[72]. The service aggregator provides data integration and ensures the secure data movement between the cloud user and multiple cloud providers.

2.7.4.3 Service Arbitrage

Service arbitrage is similar to service aggregation except that the aggregated services are not fixed. Service arbitrage means an aggregator has the flexibility to choose services from multiple agencies or providers. For example, the aggregator can use a credit-scoring service to measure and select a provider with the best score.

2.7.5 Aggregation Platform

The cloud aggregation platform [78] "is a way to bring together different point solutions and services to address a single goal; it is much more beneficial to a single service provider or enterprise that has one service to offer."

Originally, SaaS applications were mainly used by Small and Medium Business (SMB) for non-core business applications. Typical examples are Google Apps (an office suite, e-mail, calendar, etc.) and Salesforce (CRM). Recently however, large enterprises are increasingly

[14] Ibid.

starting to use SaaS for core business applications [51]. "SaaS is increasingly popular because it allows business to access specialist and complex services on demand, with no initial investment or ongoing maintenance required."[15] Service providers also benefit from SaaS by making their services available globally, dramatically increasing the potential customer base. An aggregation platform for SaaS is a single point of access that integrates and aggregates multiple SaaS and hosted service offerings. This platform enables service providers to accelerate revenue growth by providing SMBs and large enterprises with a "*one-stop-shop*" experience for multiple SaaS and hosted service offerings from a unified web portal access (Marketplace portal)[16].

Aggregation platforms include: HP Aggregation Platform for SaaS (HP AP4SaaS)[17], or Prevalent Cloud Services Aggregation Platform[18], etc.

- The Prevalent Cloud Services Aggregation Platform (Prevalent ONE) is a cloud-based solution that provides unified management of service providers' compliance, applications, and security services[19].
- HP AP4SaaS eases the integration of IT/ Communication SaaS (CaaS) applications and other hosted services into the providers' environment and expedites the delivery to individual customers[20]. Figure 19 illustrates an overview of HP AP4SaaS.

[15] HP/INTEL project on Cloud computing (SaaS); HP Cloud Computing SaaS offering http://www.hpintelco.net/pdf/solutions/SB_HP_Cloud_Computing_SaaS_Offering.pdf
[16] A marketplace portal is a web based application where customers can discover, order, and manages services and products via a web browser.
[17] HP CMS resources central; HP A4SaaS, http://devresource.hp.com/partner/ngsd/HPAP4SaaS.html
[18] The Prevalent Cloud Services: http://www.prevalent.net
[19] The Prevalent Cloud Services: http://www.prevalent.net/index.php?option=com_content&task=view&id=110&Itemid=82
[20] HP CMS resources central; HP A4SaaS, http://devresource.hp.com/partner/ngsd/HPAP4SaaS.html

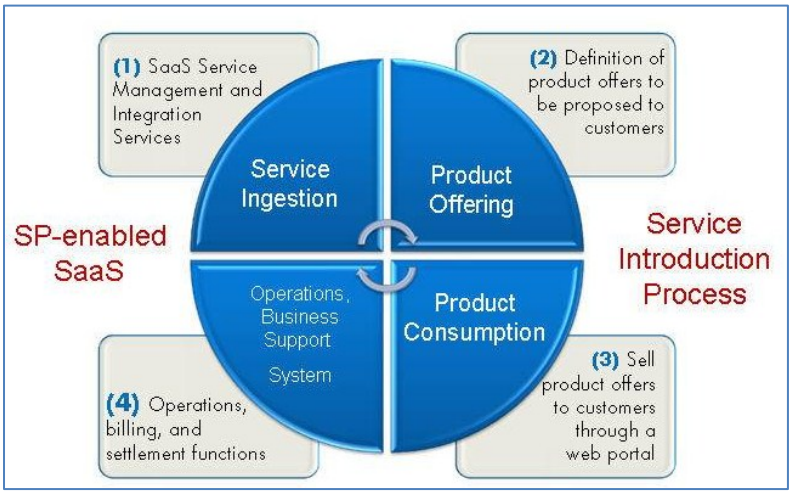


Figure 19: Overview of HP AP4SaaS [21]

2.8 Conclusion

In this chapter, we have discussed privacy policy techniques, the P3P language, cloud computing, and the cloud aggregation platform. We have overviewed P3P User agents and presented a summary of all the noticeable elements (vocabularies) in the policy language. We have also introduced the syntax and semantics of the P3P language. We have then discussed the P3P1.1 specification from the W3C.

In the following chapter, we will provide a review of previous research work in the areas of privacy languages, P3P consistency, and privacy policies in cloud computing.

[21] Ibid.

Chapter 3

Related Work

This chapter presents the findings from the initial study that investigated current approaches for presenting privacy policies. It also summarizes and examines the related research work in P3P privacy policies by presenting analysis, existing semantic solutions, and a recent discussion on P3P user agents.

3.1 Privacy Policy Language

The research area of privacy policies has made significant progress over the last ten years [46]. This includes remarkable research articles such as [24], which is an introduction to the P3P language. The research on P3P also helped facilitate the emergence of the Platform for Enterprise Practices (E-P3P) [11]. Ashley et al. [11] proposed the E-P3P language; E-P3P is designed for enterprises to create and enforce enterprise-related privacy policies. The other policy language includes the Enterprise Privacy Authorization Language (EPAL 1.2) [63]; EPAL is an interoperability language for exchanging privacy policies in a structured format between applications and enterprises. The EPAL architecture uses XACML as the structural language [2]. Agarwal et al. [3] proposed the Paranoid Platform for Privacy Preferences (P4P); this platform allows users to commit their data to an agent, and the agent then creates an alias of the data to be delivered to the service provider.

In addition to the research on privacy policy languages, the research also helped to develop policy user agents for web browsers. The user agent tool is used to produce a thorough analysis of machine-readable policies, compare privacy policies against user preferences and communicate a precise overview of the policy to the user. P3P user agents include products such as Privacy Bird [22,27,53], the user preference generator [45] etc. However, the majority of past research has focused on P3P inconsistencies [74], privacy enforcement and negotiations [2].

3.1.1 Platform for Privacy Preferences

In 2002, the W3C released the P3P1.0 specification for website owners that collect sensitive data from users to express their data handling practices. At the initial stage of the P3P project, there were other attempts to express machine-readable policies such as EPAL. However, P3P is the most popular policy language [41,62] being used by large Internet companies like Yahoo™, Microsoft™, IBM™, etc. [7]. P3P 1.0 is a good research tool with multiple suggestions, solutions, and criticisms [38,7,61]. Cranor et al. [24] defined P3P1.1 as a working specification for P3P, after taking into consideration some of the suggestions in [74,71,19,38,42].

In addition to the fundamental problem of inconsistencies in P3P1.0, there have been research efforts to make P3P less prone to semantic errors. Yu et al. [82] addressed some of the semantic problems in P3P. As a solution, they proposed relational formal semantics and integrity constraints. For the purpose of this thesis, we will adopt some of the solutions provided in [82,43].

3.1.2 A P3P Preference Exchange Language (APPEL)

Cranor et al. [26] published APPEL as the W3C recommendation for declaring user preferences; unfortunately, "there are subtle interactions between APPEL and P3P policy language that make APPEL extremely hard to use correctly." Agrawal *et al*. [4] showed the limitations of APPEL with a series of plausible examples [5]. They proposed the X-Path Privacy Preference Language (XPref). XPref "retains the two outermost XML elements in APPEL: RULE-SET and RULE. However, rather than using the sub elements of RULE to specify an acceptable or unacceptable combination of P3P elements, XPref uses an X-Path expression for this purpose."[4]

Bennicke et al [14] proposed an extension to the P3P framework and the user agent APPEL; the work aimed to provide better ways of communicating non ambiguous P3P policies via user agents, and the initiation of negotiation processes respectively. They also proposed concepts that enable service users and service providers to adapt their negotiation strategies.

3.1.3 Formal Semantics for the P3P Language

Recent research has shown that the P3P language has limited adoption by new websites and remains stagnant [59,25,10,27]. Some of the reasons why the global adoption of P3P seems slow are "there have been few incentives driving websites to embrace the policy format and there are no privacy regulations that require websites to express their privacy policies in the P3P format" [39]. Frequently, P3P policies may have syntactic or semantic errors, because they do not adhere to the P3P standard while constructing the policies [27]. Very few websites perform maintenance on their P3P policies because there is no law enforcing this process [7]. Another reason for the low adoption of P3P technology is that the tools do not accommodate other languages beside the English language. Lastly, P3P does not have precisely specified semantics and is not defined in such a way as to force specific policies to be attached to specific data items [82]. Papanikolaou et al. [62] described how formal modelling and verification can be applied to the analysis of privacy policies expressed in the P3P language; they discussed how to automatically generate a P3P policy using the Communicating Sequential Process (CSP) model.

Even though a given P3P policy may be syntactically correct, Karjoth et al. [42] noted that the policy should also be semantically defined as well. They explained that the policy given should be able to answer the following questions: Can a given recipient use a given data element for a given purpose? What is the retention policy for some data?

They argue that if these questions cannot be answered on the basis of the given P3P policy, then the policy is meaningless. Cranor [21] recognized the semantic problem in P3P when using invalid combinations of P3P elements: *purpose*, *retention*, and *recipient*. An invalid combination thus makes the policy semantically inconsistent. Yu, et al. [82] identified many areas where potential conflicts may occur as a result of invalid combinations of P3P elements.

3.2 P3P Inconsistencies

In the last few years, several papers have been published to show how semantic inconsistencies can arise in P3P policies due to a lack of correspondence between the values of P3P elements [82,21]. Yu et al. [82] argued that the following conflicts may occur as a result of the inconsistencies in P3P specification:

3.2.1 Conflicts between multiple retention values and Datum

The P3P specification [24] allows a datum to be reusable or appear in multiple statement elements, which introduces semantic problems and allows one retention element for each statement element. For example, a datum cannot have two different retention values (*e.g. no-retention,* and *indefinitely*). However, this is possible when a datum appears in multiple statements that have multiple retention values.

3.2.2 Conflicts between purpose and retention element

A situation of conflict exists when a statement contains the *purpose* value (*develop*) and the *retention* value (*no-retention*). This introduces a conflict; *no-retention* allows data collected from users to be retained for a short period of time which is required for a single online interaction, but *develop* would require more retention time [82,44].

3.2.3 Conflict between purpose and data category element

Another situation of conflict exists when a statement contains the purpose value (*individual-analysis*) but does not contain a data element from certain data categories: *physical, online, financial, purchase or government*. However, this does not correspond to the P3P specification since individual-analysis requires identifying data [82].

3.2.4 Conflict between purpose and recipient element

A situation of conflict also exists when a statement contains purpose values *(admin, develop and /or tailoring)* including recipient values (*only, delivery, same, or public).* However, this does not correspond to the P3P specification since specific purpose values and the recipient value, *ours,* must be present [82].

3.2.5 Conflict between retention and recipient element

This conflict occurs when a statement has a defined recipient value that does not correspond to the retention value. For instance, a statement may contain the recipient value, *public*, and any retention value but *indefinitely* [82].

3.2.6 Conflict from optional attributes

Where attributes expressing whether data, purpose and recipient elements are either *required* or *optional*, ambiguities may exist in the policy. For example, when data is *required*, and purpose and recipient are *optional*, it is unclear whether the entity will collect data. In cases where the STATEMENT-GROUP-DEF element is used, the value of these attributes must correspond to the value of the *consent* attribute [82].

In [73], the aim is to further classify the conflicts in P3P. Suntisrivaraporn et al. [73] classified the policy conflicts into: (1) The syntax-based conflicts and (2) The pre-defined vocabulary conflicts.

3.3 Semantic Approach to P3P

Khurat et al. [43] introduces a semantic approach to solving inconsistencies in P3P policies. This research shows that one of the disadvantages of the P3P language is a lack of formal semantics. As a result, the error verifications only associate with the syntax check, which creates the possibility of ambiguities. Yu et al. [82] addresses the problem by identifying and formally describing semantic inconsistencies in P3P policies that have been addressed by researchers in this field of research. Ninghui Li et al. [59] proposed adding semantics to policies by translating them into a relational schema. If a P3P policy is translated into a semantic database the constraints would be violated, making the policy invalid. The semantic approach introduced in [82] focuses on handling semantic inconsistencies by using integrity constraints in the semantics. Such a set of integrity constraints can benefit both end users and websites. The concepts in [82,43] use semantic constraints to detect and fix inconsistencies in P3P policies. The following are integrity constraints to solve inconsistencies:

- Data-Centric Constraints - For example, three possible values for requirement are mutually exclusive (*always, opt-out, opt-in*). In data-purpose and data-recipient relations there can only be one requirement value. It is not semantically correct to have a data-type with a purpose that can be *opt-out* and at the same time is required *always.*

```
+ #user.home-info
-  #user.home-postal
-  #user.home-info.telecom
-  #user.home-info.online
```

Figure 20: Example of data hierarchy

- Data Hierarchy Constraints – For example as illustrated in Figure 20, if the collection of *#user.home-info* is required, then it does not make sense for the collection of *#user.home-info.online* to be optional. However, it may be reasonable for the collection of *#user.home-info* to be optional, but the collection of *#user.home-info.online* to be required, which would mean that the collections of *#user.home.info.postal* and *#user.home-info.telecom* are *optional.*

- Data-Purpose Constraints 1: For any two tuple data-purpose$_1$ (data$_1$, purpose$_1$, retention$_1$) and data-purpose$_2$ (data$_2$, purpose$_2$, retention$_2$), if data$_1$ is more specific than data$_2$ and purpose$_1$ = purpose$_2$, then retention$_1$ $\geq$ retention$_2$. (*Always* $\geq$ *Opt-out* $\geq$ *opt-in*). They argued that similar constraints apply to the data-recipient relation and the data-collection relation.

- Data-Retention Constraints 2: For any two tuple data-retention$_1$ (data$_1$, retention$_1$) and data-retention$_2$ (data$_2$, retention$_2$), if data$_1$ is more specific than data$_2$, then retention$_1$ $\geq$ retention$_2$ (*indefinitely* > *business-practices*, *legal-requirement*, *stated-purpose* > *no-retention*).

However, the integrity constraints explore only a subset of P3P policies; as shown above, listed constraints will not be able to discover any inconsistencies if P3P policies are merged.

3.4 Related Research on P3P

Halpern et al. [36] proposed reasoning about policies using first-order predicate logic. Choi et al. [79] later applied predicate logic to workflow security management; their security policy focuses on access control for a specific workflow, but the transformation from workflow into logic is not mentioned in their work. Glasgow et al. [33] proposed a formal framework for privacy policies called *security logic*, which applies modal logic to generic security policy considerations. He et al. [37] analyzed security policy integration between different application domains. They provided the requirements for policy integration and patterns. Satoh et al. [70] used security policies of the external services to solve inconsistencies. They showed how hard it is for policy developers to define composite policies by hand. The composition rules are not clear for maintaining consistency and the composite processes and policies are themselves complicated. They proposed a logic-based approach to composite security policies for an aggregate process. Khurat et al. [43] built on the early research work from [82,70] and they enhanced the P3P language to be able to support composite services by proposing a formal semantics for the P3P language to preserve semantic consistency and defined a method for combining privacy policies for composite services.

3.4.1 Combining Privacy Policies

Khurat et al. [43] defined formal semantic constraints for P3P policies in a composite environment. They assumed the P3P policies obtained from each service do not conflict with issues mentioned in section 3.1.4. They defined a combining mechanism and described some constraints to verify inconsistencies of privacy policies between service providers.

3.4.1.1 Constraints for Integrity between services

Additional integrity constraints were introduced in [43] to solve the inconsistencies that may arise from combining multiple service policies. The following are brief explanations of the constraints.

- *Multiple statement for a datum*: P3P specification [24] allows more than one STATEMENT element which declares the same data-purpose. Thus, multiple retention values may exist for the same data-purpose [43].

  CONSTRAINT

$$\forall p, q \in DPRD, p.(data, purpose) = q.(data, purpose)$$

$$\rightarrow p.retention = q.retention$$

- *Data Hierarchy*: With the P3P language, the data structure is hierarchical. It will not make sense if the upper level has more restrictions than its lower level.

  CONSTRAINT

$$\forall p, q \in DPO, \quad p.data\ is\ the\ upper\ level\ of\ q.data, p.optional = no$$

$$\rightarrow q.optional = no$$

- *Purpose and Data Optional Constraints*: If the purpose value of a data is CURRENT, the optional value of that data must be *no* since it is obvious that the website will collect data. Otherwise, the website will not provide the service [43].

  CONSTRAINT

$$\forall p \in DPO, \quad p.purpose = current \rightarrow p.optional = no$$

- *Data Collection*: If data are needed to fulfill the service provisioning, then its collection must be *required*. Otherwise, this composite service cannot function.

  CONSTRAINT

$$\forall p \in DPO, \quad p.data \in shared.data \rightarrow p.optional = no$$

- *Purpose Element*: For the same reason, the PURPOSE values of these data must contain at least a current value with a *required* value such as *always*.

  CONSTRAINT

$$\forall p \in DPR, \quad p.data \in shared.data \rightarrow p.purpose$$

$$= current \wedge p.required = always$$

3.4.1.2 Mechanism for Composite P3P Policies

Khurat et al. [43,82] discusses the overview of data purpose-relation by using table 3; this table illustrates elements associated with different key relations.

Relation Tables	Fields	Key of the relation
DPO	Data Purpose Optional	(Data, Purpose)
DPR	Data Purpose Required	(Data, Purpose)
DPC	Data Purpose Category	(Data, Purpose, Category)
DPRD	Data Purpose Retention Duration	(Data, Purpose)
DPRTR	Data Purpose Recipient Third_Party_Service Required	(data, Purpose, Recipient, Third_Party_Service)
Service-Entity	Entity(#business.name) Access Dispute(resolution-type) Dispute (Remedies)	Entity

Table 3: Khurat's Details of relation table [43]

3.4.1.2.1 Data – Purpose – Optional (DPO)

The research [43] assumed that the P3P policy from each service is validated before combining the policies. The optional value of a data-purpose (*dp)* for a composite service must be only one value. *n* tuples from DPO (dp_1, $optional_1$), (dp_2, $optional_2$) ,…, (dp_n, $optional_n$) that have the same data-purpose ($dp_1=dp_2= \ldots = dp_n$). They defined that *optional* values of the composite service are the most restrictive. This means that if at least one of the *optional* values is *no,* then the optional value of the composite service must be *no*. The following is a composite service rule for data-optional values:

- *No* (means data is required) >*Yes* (means data is optional)

3.4.1.2.2 Data – Purpose – Required (DPR)

Similar to the DPO explained above, the *required* value of a *dp* from each service must be only one value i.e. *always*, *opt-out* or *opt-in*. *n* (any real number) tuples from DPO (dp_1, $optional_1$), (dp_2, $optional_2$) … (dp_n, $optional_n$) that have the same data-purpose ($dp_1=dp_2= \ldots = dp$). The following is a composite service rule for the data – purpose required:

Always > *Opt-out* > *Opt-in*

3.4.2 Related Approach to Merging Privacy Policies

In [30], Dong et al extended the proposed idea from [43] (section 3.2.1), they also adopted the two-way merging approach discussed in [56]. They used the approach to compare two different P3P policies. This merging process does not allow information from the semantic constraint discussed in [82] to be used when resolving conflicts. The merging algorithm introduced in [30] demonstrates how to put into practice composite privacy policies. However, the merging algorithm cannot determine whether the P3P policies used or the resulting P3P policy is semantically consistent. It also cannot match two or more P3P STATEMENT elements. For the purpose of this thesis, we adopted the following formal model of privacy policy and its elements.

3.4.2.1 Formal Modeling of Privacy Policies

According to the research presented in [57,30], Michael et al [57] presented a formal definition of privacy policy and its contents.

Definition 1 (Privacy Policy) "A Privacy Policy is a tuple of a vocabulary (Subject, Object, Operation, and Purpose), a set of authorization rules, and a default ruling." Here Subject (S) is the set of end-users, Object (O) is the set of data, Operation (OP) is the set of operations performed and Purpose (P) is the set of data purpose.

In [30], Dong et al. assumed that the components of a privacy policy (*Pol*) are called (*Vocabulary (V), Rulesets (R), Authorizations (A)*). For example, the following is a privacy policy that consists of a tuple of *n* rules, where r_i is an instance of *(V, R, A)*.

$$Pol = (r_1, r_2 \ldots r_n).$$

Definition 2 (Vocabulary) "A vocabulary is a tuple $V \in (S, O, OP, P)$." Authorization rules $A = \{+, -, *\}$ denotes *allow*, *do not allow*, and *do not care* respectively; for entities and policy developers. $A = \{+, -, *\}$ denotes *require privacy information, do not require privacy information,* and *do not specify* respectively, for policy users.

Definition 3 (Ruleset) "A ruleset (*R*) for a vocabulary *V* is a subset of (*I* x *S* x *O* x *OP* x *P* x *A*)," where *I* denotes an index set. In [30] an instance of a ruleset (*rs*) is denoted by a complete $\subseteq$ of (*i, s, o, op, a*).

3.4.2.2 Semantics of Privacy Policies Aggregation

Dong et al. [30] presented a formal definition for policy aggregation and override rules. However, they assumed the policy statements relate to the data items.

Definition 4 (Policy Aggregation) "Assuming the $Pol_k = (r_{k,1}, r_{k,2}, r_{k,3} \ldots r_{k,n})$ for k = 1, 2… m; m is a policy set. $r_{k,i}$ denotes a ruleset for the policy." Figure 21 denotes the aggregate privacy policy of Pol_k.

$$Pol(R_I) \leftarrow \bigoplus_{k=1}^{m} Pol_k(r_{k,i})$$

Figure 21: Equation for Aggregate Policy

Dong et al [30] denotes the generic aggregator operator as $\oplus$, which represents the union, intersection, differences, and conflicts depending on the properties of the policy rule. [30,43] used $\leftarrow$ to denote an assignment in a policy merging process.

Definition 5. (Policy Operations)

Two policies A and B are said to be *related* ($A \sim B$) if and only if they have associated policy properties, otherwise they are not related ($A \nsim B$).

For example, if both policies A and B are associated with gender, then the policies will be "related." However, if *A* is about gender information and *B* is about driver's license information, then they are "not-related."

$$\text{Pol}\,(A, B) \leftarrow \text{Pol}_1\,(A) \oplus \text{Pol}_2\,(B)$$

The rules in a privacy policy are subject to the following operations:

- Equivalent (=): If two policy rules, *A* and *B*, represent the same privacy policy, then they are equivalent; that is $A = B$.
- Override (>): If a policy rule *A* dominates another rule *B*, *A* overrides B ($A > B$).
- In Conflict (< >): Given two related rules *A* and *B*, if neither rule overrides the other, then they are in conflict or ($A <> B$).

3.4.3 Formal Modeling & Verification of P3P Policies

In the case of formal verification of privacy policies, the research is still ongoing to address the need to automatically discover or check for privacy violations [76]. A typical use of formal methods in privacy policies is to study the semantics of security policies, privacy policies, and access control policies. Using formal methods to model privacy policies has been an area of interest in the field of privacy policies for a substantial amount of time [31].

Graham et al. [35] translates XACML policies into a simplified mathematical model. They interpreted mathematical models to logical expressions (*first order logic*). Logical expressions evaluate to be true if and only if the corresponding relation holds. Jackson [40] presents the logic with the type system, the syntax, and the formal semantics. He reported some case study applications of the analysis to check the consistency of a formula, and to check the validity of a theorem. The act of creating aggregate policy out of multiple policies may leave it vulnerable to unintended conflicts [35,52,34]. Graham et al. [35] used Alloy Analyzer to "automatically check the truth value of logical expressions." They check whether a combination of privacy policies will faithfully reproduce the properties of its sub-policies, and thus discover unintended conflicts before the policies appear in the services.

3.4.3.1 Formal Policy Languages

The privacy right of exclusion requires that the users know how service providers will use their sensitive information. Thus, providers must encode their practice into publicly available privacy policies [76]. Graham et al. [34] introduced "a formal model for XACML policies which partition the input domain to four classes: permit, deny, error, and not-applicable."

XACML is an XML-based language for expressing access rights to arbitrary objects that are identified in XML. XACML provides rules and policy combining mechanisms for constructing policies from rules and meta-policies from policies respectively.

Xiang [31] proposed Privacy Verification (PV), a framework built upon first order logic. He used the P3P policy language as a standard language because of its wide acceptance and its first order logic properties. "The policy language such as P3P, which has a formal notation, informs website visitors of the website's privacy practices and enables automated methods for finding privacy-conscious sites. However, the P3P language lacks formal semantics"[24].

3.4.4 Automated Tool

One of the advantages of formal methods is that formal specifications are amenable to machine manipulations and machine analysis (e.g., finding bugs or proving properties). Automation not only helps us catch human errors, but also enables us to scale up pencil-and-paper techniques [75].

3.4.4.1 Alloy Modeling System

Alloy [40] is a modeling language based on relational first order logic developed at MIT by Jackson and his team. Unlike a programming language, an Alloy model is declarative. This allows the construction and analysis of very brief and partial models. It is similar in structure to these formal specification languages: Z, B, etc. However, Alloy is amenable to a full automatic analysis in the style of a model checker.

The Alloy Analyzer is a tool that permits users to verify and analyze Alloy models. The alloy language is a structural and declarative language. Alloy models consist of sets of concrete objects, called *signatures*. The signatures contain *facts* and *relations* about the *sets*.

3.5 Conclusion

In this chapter, we have discussed related research work in privacy policies and in formal semantic policy research areas. We have provided an overview of some existing solutions and the on the limitations of the P3P language. We talked about the existing constraints used to develop a formal semantics for the P3P language. We have described different conflicts that may arise due to inconsistencies in P3P specifications. In addition, we have mentioned the current solutions to combine privacy policies in composite services and improvements in the research area of the P3P Language. Finally, the Alloy Analyzer and other related work on formal modeling of privacy policies were discussed.

In the next chapter, we will begin our introduction to the merging strategy, which specifies the requirements and desired features for the P3P merger system. We will use the three-way merge principle to specify how the merger system should perform its task.

Chapter 4

Merging Strategy

This chapter presents the merging processes required to aggregate P3P policies. It states some requirements and desired features that must be satisfied when merging privacy policies. Furthermore, we divided the merging strategy into four different logical processes. Lastly, this chapter defines the P3P merging problems that may occur while merging policies.

4.1 Overview of Merging Strategy

Based on the various ways [18,15,77] in which structured data have been previously analyzed for XML merging processes, we adopted some of the techniques and requirements used in [18,15,77] to form a complete merging strategy. However, to create a consistent aggregate P3P policy, there are some additional requirements that have to be satisfied. Those additional requirements should be in agreement with the P3P specification. First, we need to match the P3P policies to detect the conflicts that may occur during the merging process, and to establish a logical relationship between the STATEMENT elements in the policies. The most important requirement for the merging process is to avoid data duplication and data loss when creating the new policy.

4.2 Requirements for P3P Merge Process

This section introduces the merging strategy for aggregating P3P policies. To create a P3P policy, one needs to revise the P3P schema[22] in a manner that agrees with the data handling practices of the provider and the P3P 1.1 specification. Because the P3P specification allows the usage of one or more STATEMENT elements in a P3P policy, a given P3P policy may have a structure that varies from other P3P policies. Figure 22 illustrates a scenario where P3P policies have different statement structures. Since services have different privacy practices, merging two or more P3P policies may lead to a conflict. Figure 23 illustrates the merge process involved in aggregating multiple P3P policies.

[22] http://www.w3.org/TR/p3p-rdfschema/

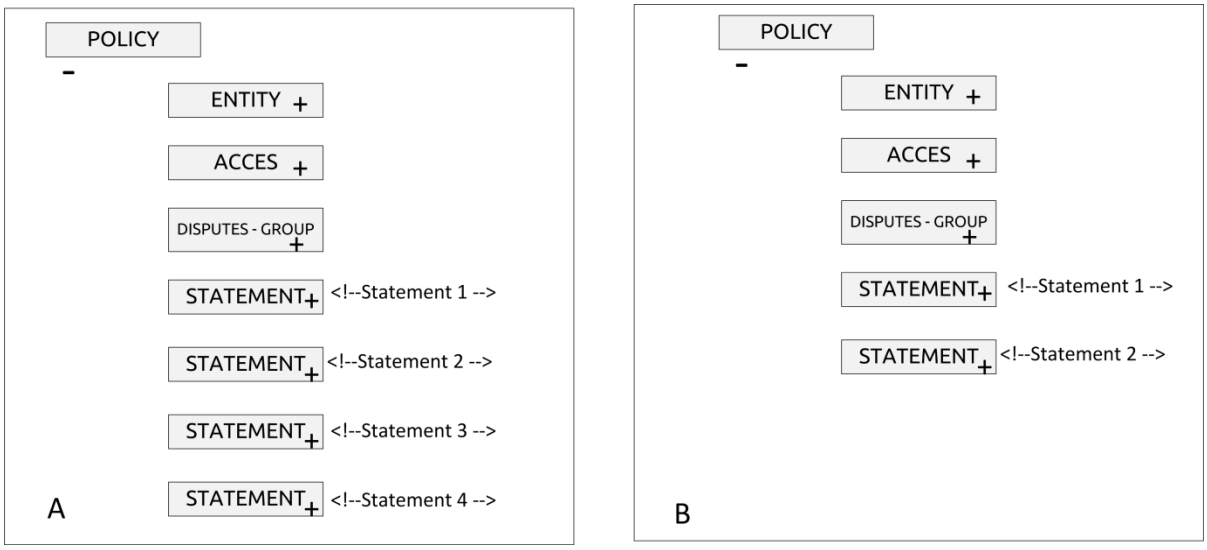


Figure 22: Multiple Statement Element Scenario in P3P policies.

Luuk et al. [50] discusses several merging algorithms which handle the structural merging process in a syntactic manner. Merge algorithms use different amounts of knowledge about the language to merge source code from the language. The following sections present an overview of the merging algorithms.

4.2.1 Text-Based Merge

Traditional merging algorithms are the foundation of the textual merge techniques. "This gives high flexibility to the merging process, since any structured data can be treated as a text file" [50]. The text-based techniques aggregate texts in parallel to their programming languages. For example, the text-based merging algorithm detects parallel modification such as insert actions, delete actions, and changes of a text line in a syntactical manner. The main disadvantage of a text-based merging algorithm is that it identifies only very basic conflicts which involve text errors. This drawback is due to the lack of structured syntactic and semantic knowledge. The text-based merging algorithms do not take into account the syntax and semantics of the language in which text streams exist. To build a policy-merging algorithm, the text-based merging techniques will not fully capture the consistency we wish to accomplish while merging P3P policies.

4.2.2 Syntactic Merge

The syntactic merging algorithms use language-specific knowledge and can therefore evaluate and merge structured data better than the text-based algorithms. The techniques take into account the syntax of the language of the texts before merging. The underlying data structure for syntactic merges are usually trees, which requires the merge tool to parse the data in advance and generate the corresponding trees. However, Philipp [65] stated that the main problem with syntactic mergers for policy merging is that it lacks semantic interpretation of the structured trees.

4.2.3 Semantic Merge

The semantic merging techniques address sets of conflicts that are not detectable by the syntactic merge algorithms. To detect and correct these kinds of conflicts, the merging algorithms must obtain more information about the semantics of the language. The P3P schema and specifications provide semantic information that the semantic merging algorithms need. In our work, we will use both the semantic and syntactic merging algorithms.

4.3 Merging Strategy

Merging pairs of privacy policies is usually an interactive task. However, the entire merging process in this thesis consists of different tasks, which trigger automatically. We utilized the three-way merge principle (section 4.3.3.2) while merging the P3P policies with the P3P schema defined by the W3C, in order to avoid some syntactic conflicts that may occur.

In the cloud aggregation platform, the aggregator may have no chance to modify and merge all the P3P policies. The only way is to use normal editing operations and to make merging decisions automatically. Sometimes it is necessary to undo the decisions because the solutions of a merged policy may imply a different meaning from what the service policies intended. The aim is not just to merge P3P policies, but to merge the policies with additional merging instructions (semantic merging). Therefore, the merging process in Figure 23 consists of five main steps altogether: 1) retrieving the P3P policies from service providers;

2) P3P policy validation and conflict detection; 3) P3P policy match detection; 4) merging P3P policies and solving conflicts, and finally 5) creating the merged P3P policy.

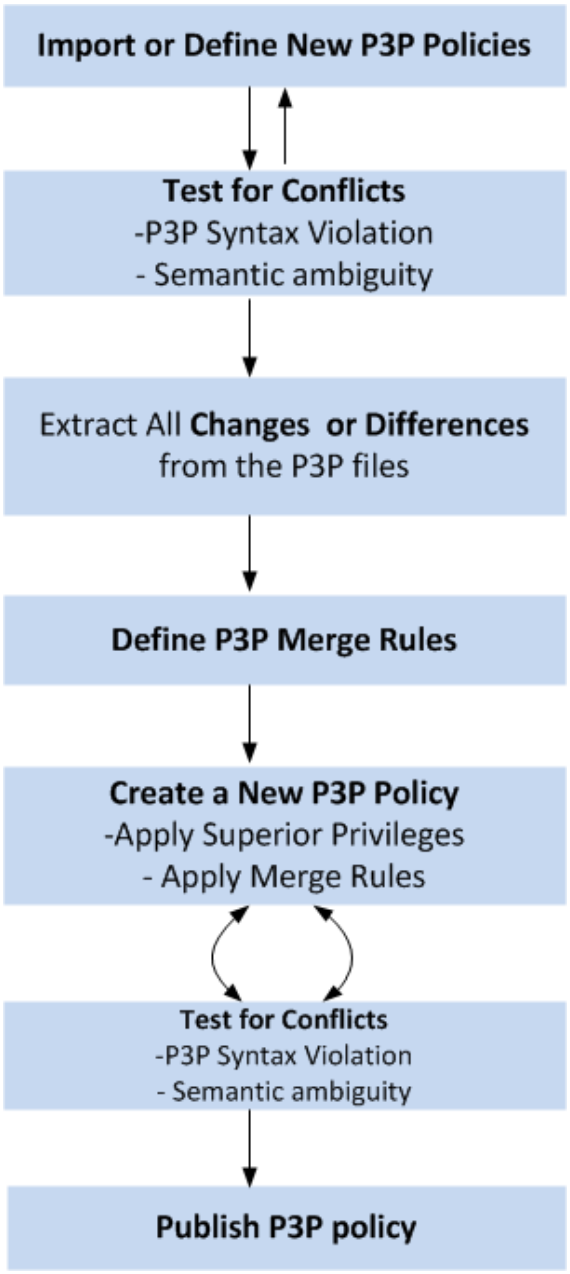


Figure 23: P3P Merge Process

4.3.1 Policy Validation and Conflict Detection

As shown in Figure 23, the P3P policy validation and conflict detection steps are required in the merging process after uploading the P3P policies. We use the validation process to check if the P3P policies agree with the P3P1.1 specification [24]. Instead of our own validation scripts for retrieving and validating P3P policies, we adopted the P3P Validator[23] developed by W3C.

The P3P Validator checks policies for compliance with P3P1.1 using the following steps[24]:

- *Access*: The Validator checks if the Internet can access the policy URI, using the HTTP GET method.
- *Syntax*: Checks if the syntax of the policy is correct using the following sub steps: 1) checks if the policy is well formed as an XML document. 2) Checks if the policy has the correct namespace URI for P3P. 3) Checks the root element of the policy file if it is either POLICY or POLICIES. 4) Checks the policy if it conforms to the XML schema for P3P[25].
- *Vocabulary*: Checks if the P3P data item in the <DATA ref= "…"> directives are included in a P3P base data schema.
- *Link: checks* if the URI references included in the policy are accessible. The Validator checks for the following:

  1) Discuri and Opturi (attributes of *policy* element)

  2) Service (attribute of *disputes* element)

  3) SRC (attribute of *img* element).

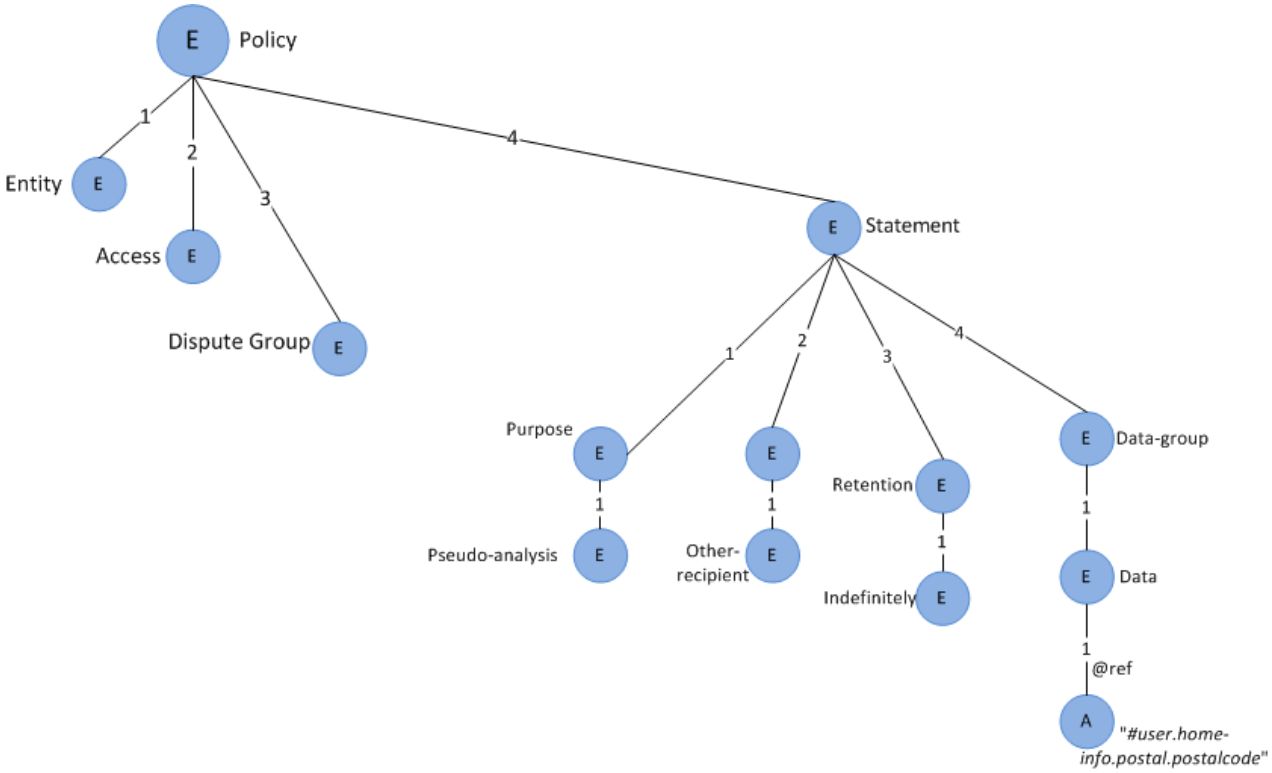
After detecting the validity of P3P policies, some valid P3P policies might remain inconsistent and semantically incompatible for merging. P3P Validator may fail to identify such policies because it lacks semantic information on of the P3P language. For example, the

[23] The W3C P3P Validator is available as a free service at http://www.w3.org/P3P/validator. It was implemented in Perl by Yuichi Kioke and Shojima Taiki.
[24] Ibid.
[25] http://www.w3.org/TR/p3p-rdfschema/

validator cannot detect if two service providers have different policies that collect separate *data* items, *purpose* values, and *retention* values with their relationship. However, the lack of semantic verification may cause inconsistencies to occur in an aggregate policy. Therefore, the result of an aggregate policy may be inconsistent, as discussed in section 3.1.4. Therefore, we adhere to the constraints discussed in section 3.2.1 on how to apply constraints to P3P policies, and to avoid or detect policy inconsistencies.

4.3.2 P3P Match Detection

In order to merge multiple P3P policies, it is essential to compare them, so as to identify the differences and similarities. The match process is crucial to produce a quality aggregate policy. The problem of matching P3P trees is concerned with constructing a semantic match between two policy trees (for example *A* and *B* in Figure 24 below). The solution to the problem will be given in section 5.3.1.



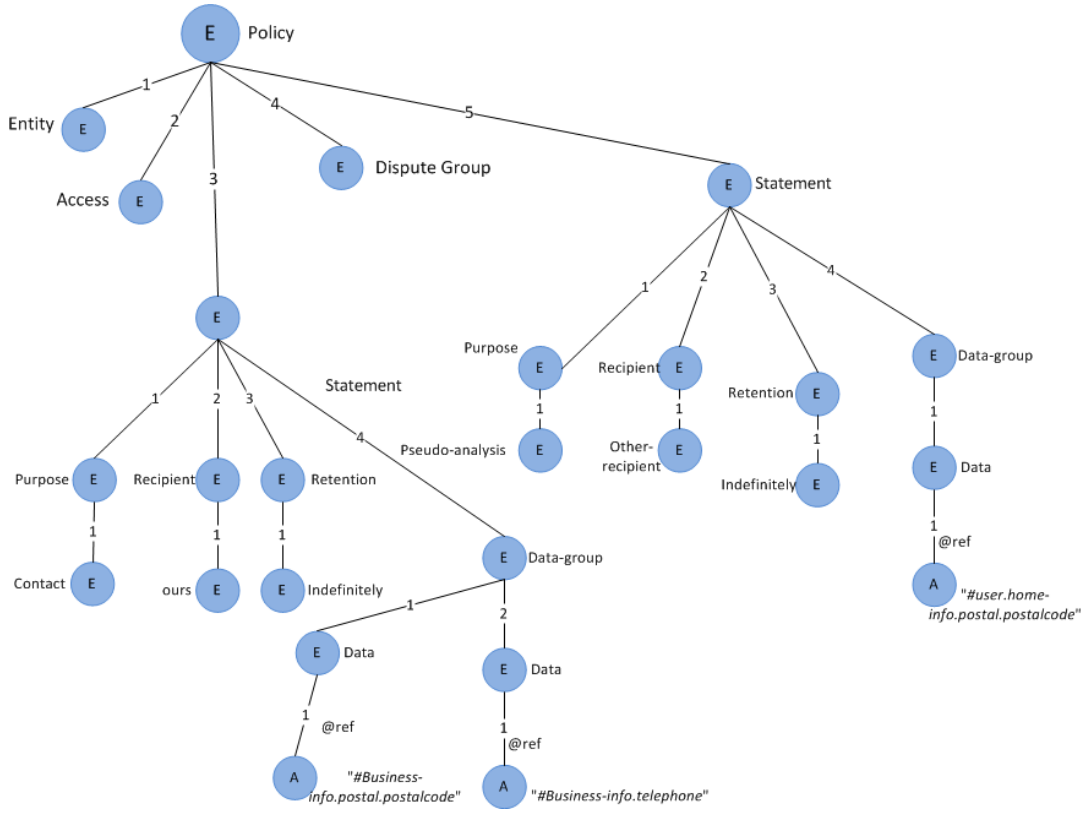


Figure 24: P3P Tree matching problems (P3P A and B)

P3P match detection means that we have to recognize similar elements (for each elements e.g. STATEMENT, DATA, PURPOSE etc.) in all the policies to be merged. This process is the match operation. Every element in XML needs an identifier by which it can be recognized, and the P3P specification provides an element name for all tags [24]. In order to have a consistent merge, we will consider two ways to match the policies.

- We will apply the syntactic match: if the *element name* of a given node is equal to the element in the same tree level. In other words, if the P3P policies have the same *name* and contents, so we have a match.

- We will also apply the semantic match: if the *element name*s are equal but the contents are not semantically and syntactically equal. In this case, we will need a matching algorithm which recognizes the similarities between the contents of the policies.

Before applying the match algorithm, the semantic and syntactic inequalities between elements that have equal names may indicate match problems. A match problem may arise if corresponding nodes in both tree tags have the same <*E* node> name; see Figure 24. Figure 24 illustrates that two different STATEMENT elements may exist in P3P *B*, and only one statement in P3P *A*. The description may result in a semantic match problem; because the corresponding data nodes in the P3P tree for *B* may not be identical to *A*. Matching both trees will result in a policy conflict or change the meaning of the aggregate policy from what the providers intended. The existing match mechanisms and P3P specification will often recognize elements using their similarities. These match problems and the need for a semantic match leads us to the conclusion of modifying the *data element* in the P3P language. We introduced two attributes to the P3P specification to enhance the semantic match operation. This will be explained in chapter 5.

4.3.3 P3P Merge Process

In this sub-section, we will describe a possible way to perform a merge process based on the P3P specification, which satisfies the syntax and semantic specifications of P3P. As illustrated in Figure 23, the order is important because each step requires an output from the previous step. In [50] a few methods for merging unordered trees and identifying conflicts are outlined. For achieving a semantic and syntactic merge, we considered a three-way merge over two-way merge. The following sub-sections will provide a brief overview to both merge principles previously mentioned.

4.3.3.1 Two-way Merge

The Two-way merge Principle [48] performs an automated merge between two different versions of XML files by comparing the two XML files without the need of any other information from the language specification or semantics. Consequently, the differences between the two XML files may lead to policy conflicts when merged because the result was not semantically analyzed.

Figure 25 shows how the two-way merge principle operates; the elements extracted from *File A* and *B* are directly merged together to generate a new file with some shared attributes.

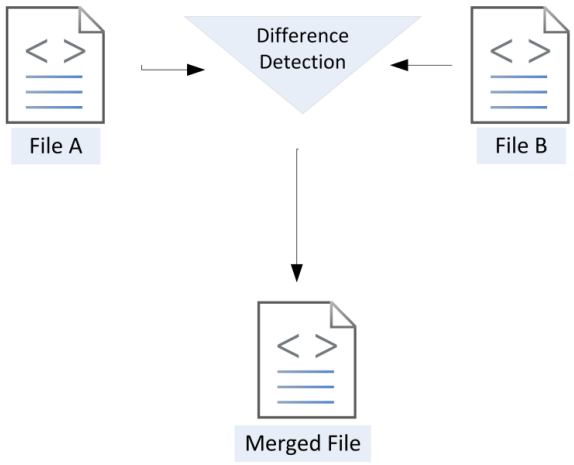


Figure 25: Two-Way merge Principle [50]

For example, the P3P merged result from a two way merge may be error prone or may require additional verification to correct the policies prior to completing the merge operation.

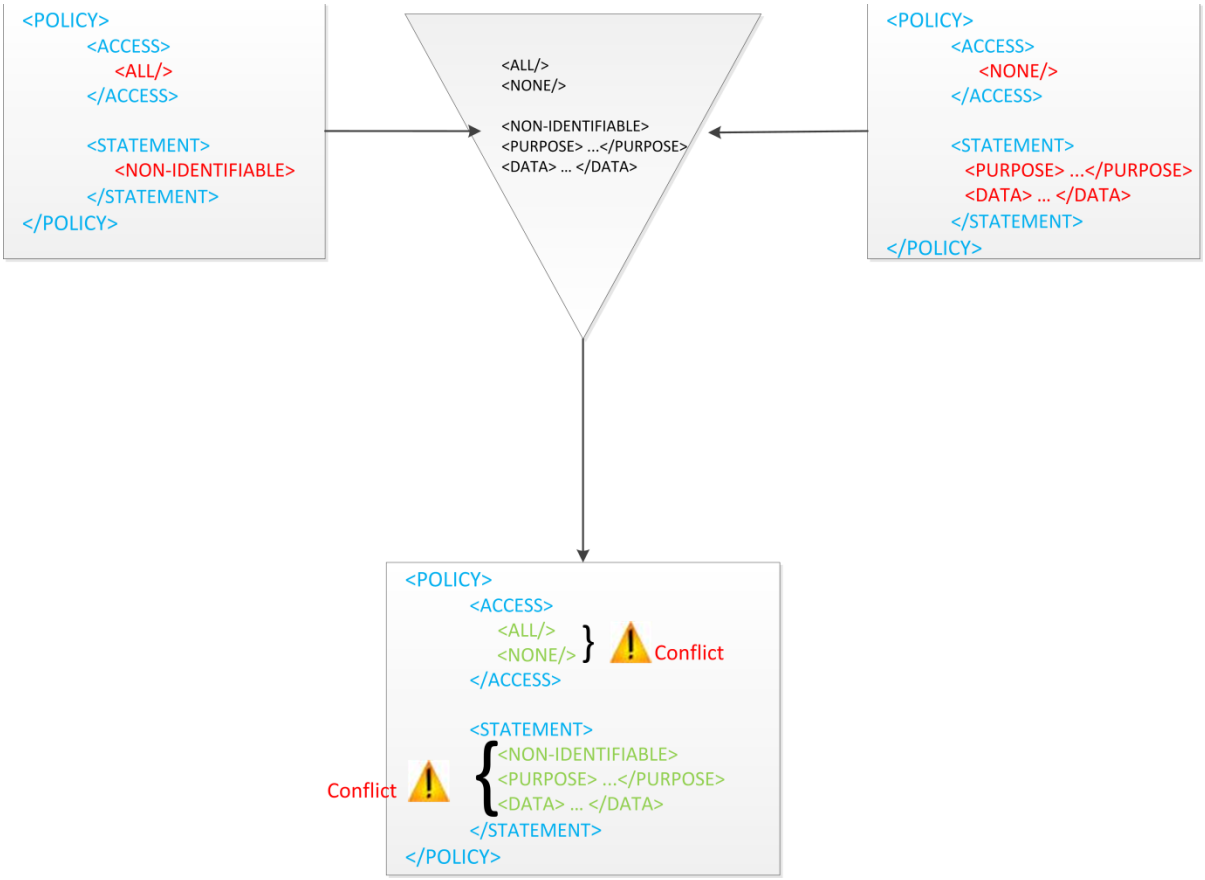


Figure 26: Two-Way merge operation on P3P policies

4.3.3.2 Three-Way Merge

The three-way method takes two replicas of an XML document into account and thus has more information about the syntax and semantics at hand to decide where a modification is required and whether the aggregate result can create a policy conflict or not. As Figure 27 illustrates, the three-way merge method needs three files: the original file (the P3P schema) and two different files *A* & *B*. The two files were modified from the original file independently; this means they have the same pool of semantic and syntactic rules.

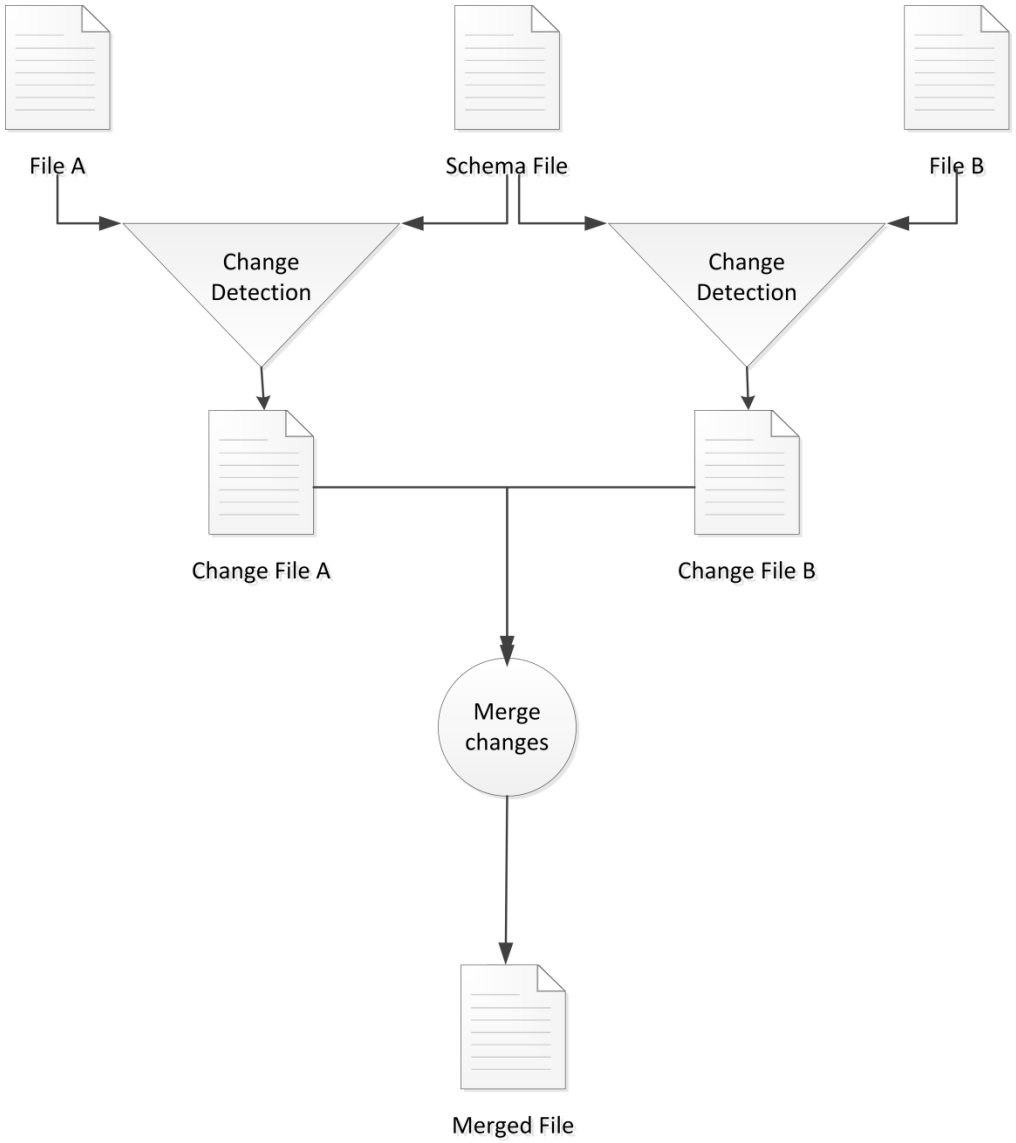


Figure 27: Three-Way merge Principle, [50]

The original file compares separately itself with the modified files; the differences achieved from those files (*A* and *B*) are gathered to create the change file. The change files describe the modifications to the original file to create files *A* and *B*. The result of the *combine change* stage merges with the original file to produce a merged file which contains all modifications from (A and B).

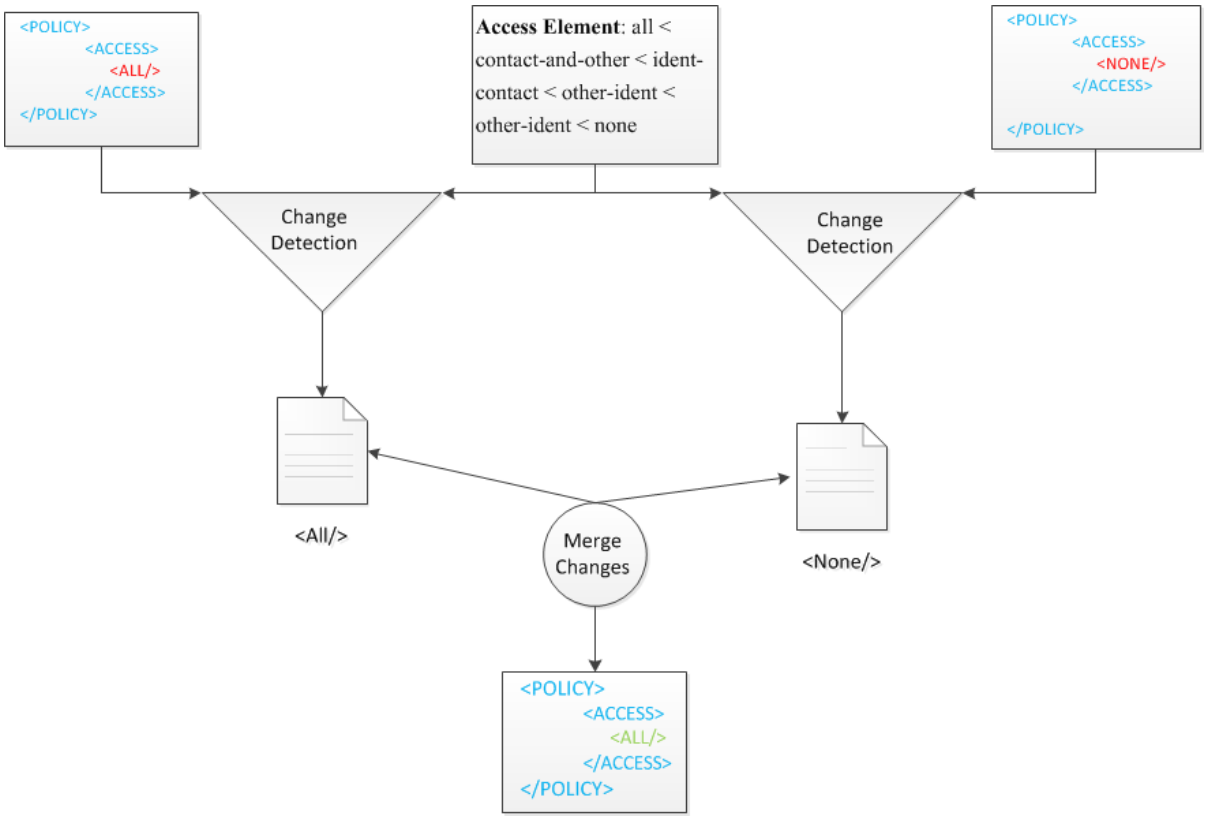


Figure 28: Three-Way merge operation on P3P policies

4.3.4 Creating New Policy

In producing a new policy from the merging process, we took into consideration that the merged policy should be symmetric and should avoid data loss. When generating the new aggregate file, we highlight and show a brief report of where changes and merges were done; this allows the aggregator to manually override any decision.

The merge process for aggregating P3P should be reputable. If we perform the merging process on different P3P policies several times, we should always obtain the same output.

Moreover, it should be symmetric in the sense that the aggregate result should depend neither on the order of the elements in the policy nor on the order of match detection. For example, if we load policies *A* and *B* into a merge process, the result should be the same as if we had loaded policies *B* and *A*.

4.4 Conclusion

In this chapter, we have presented the steps required for our P3P merging strategy and the matching and merging processes required to aggregate multiple P3P policies. We have also presented an overview of the merging strategy, which describes the features and requirements that must be satisfied when aggregating privacy policies. We described the common merging methods that are used in aggregating structured data (e.g. the text based merge, the syntactic merges, and the semantic merge). We have discussed the differences amongst the merging methods by illustrating examples where they are most useful. This chapter also elaborates on the merge principles: the two-way and the three-way principles.

We have introduced the P3P merging strategy, the merge processes (the policy validation and conflict detection, the P3P match detection, the P3P merge process, and formulating a new P3P policy). Merge processes consist of different tasks, which trigger automatically. We have adopted the 3-way merging principle, which consists of both semantic and syntactic merge methods.

In the next chapter, we will introduce the P3P match algorithm; the algorithm that finds the similarities and relationships amongst P3P elements. We will present the tree structure of the P3P language using the Document Object Model (DOM) representation of W3C. We will define and use match types to classify the similarities amongst P3P elements.

Chapter 5

The P3P Matching Algorithm

This chapter introduces the P3PMatch algorithm. Prior to the merging phase, we first detect the differences and similarities amongst the P3P elements. The matching algorithm finds the relationships amongst the P3P STATEMENT elements. We introduce a notion called the P3P relative data key, which consists of two pairs (data-purpose and data-recipient) for the P3P DATA element. This notion will semantically enhance the matching algorithm to detect the PURPOSE and RECIPIENT elements that relate to a DATA element; this will improve the quality of matching for multiple P3P policies.

5.1 P3P Match Prerequisites

In this section, we introduce the prerequisites for the P3P matching phase. To satisfy the objective of this thesis, we need a matching algorithm that can iterate through multiple P3P elements to identify matching P3P elements. The matching process is one of the four main stages of the merging strategy discussed in (Section 4.3). We introduce matching prerequisites that must be satisfied when matching policies. The following are the two prerequisites to be considered:

- Structural prerequisite: Prior to matching policies, it is very important to understand the structure of the policy. The algorithm should be able to model the exact number of elements, attributes, and contents of the policies.
- Semantics prerequisite: To improve the quality of our algorithm, we consider the P3P semantics when detecting the similarities and differences in P3P policies. To attain the accurate semantic matching property in the P3P matching algorithm, we introduce the notion of P3P relative data key (section 5.3).

5.2 Tree Structure of the P3P Policy

The format of XML files is hierarchical and it can be represented as a tree structure [6,77]. Section 2.3 shows the tree model of a sample XML document. In order to design an efficient merge system for the P3P language, we need to detect the differences and similarities in the policy elements. Since the match detection is an important phase in merging P3P policies, it is also necessary to understand the hierarchical structure of the P3P policies. Considering the structure of XML, it is reasonable to use either trees or graphs as the underlying data structure of the P3P language. However, for the purpose of this thesis, we will represent the structure of P3P policies as trees.

5.2.1 Document Object Model (DOM)

The W3C Document Object Model (DOM) is a "platform and language-neutral interface that allows programs and scripts to dynamically access and update the content, structure, and style of a document."[26]

The Document Object Model (DOM) [64,6] provides some insight into the semantics for XML documents. According to the survey on DOM, a document is a hierarchical structure of nodes. Nodes are of several types, but the following are the most important for P3P:

- Element node: element node (E node), which has a label (name) only; the element node may have children.
- Attribute node: attribute node (A node), which has a label and carries text.
- Text node: text node (T node), which has no label but carries text; attribute and text nodes are terminal nodes.

For Example, we represent the element (E), attribute (A), and text (T) nodes as a tree in Figure 30. The sample P3P code presented in Figure 29 corresponds to the tree described in Figure 30.

[26] W3C Document Object Model (DOM) http://www.w3.org/DOM/

```
<POLICY>
   <STATEMENT>
        <PURPOSE><pseudo-analysis/></PURPOSE>
        <RECIPIENT><other-recipient/></RECIPIENT>
        <RETENTION><indefinitely/></RETENTION>
        <DATA-GROUP>
        <DATA ref="#user.home-info.postal.postalcode"></DATA>
        </DATA-GROUP>
   </STATEMENT>
</POLICY>
```

Figure 29: A sample P3P code

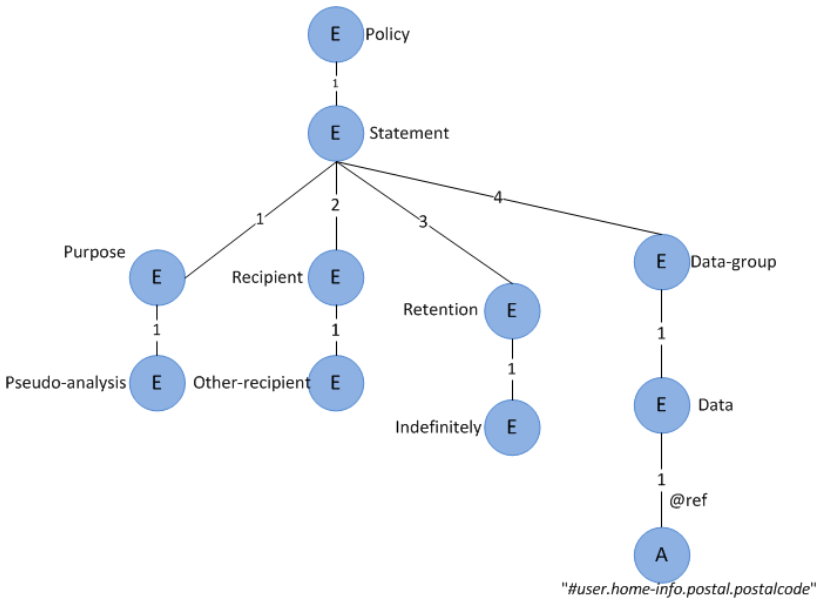


Figure 30: Tree corresponding to the P3P Code in Figure 29

Type	Name	Text	Attribute and Values
Element	Policy	-	-
Element	Statement	-	-
Element	Purpose	-	-
Element	Recipient	-	-
Element	Retention	-	-
Element	Data-Group	-	-
Element	Pseudo-Analysis	-	-
Element	Other-Recipient	-	-
Element	Indefinitely	-	-
Element/ Attribute	Data	-	ref="*#user.home-info.postal.postalcode*"

Table 4: Node tables to explain the P3P code in Figure 29

5.3 The P3P Relative Data Keys

In a database, constraints make it possible to enforce restrictions on attributes [64]. For instance, "a constraint can restrict a given integer attribute to values between 1 and 10". The essential parts of a database are the keys (primary and foreign keys). The keys provide the means by which one *tuple* in a relational database may refer to another *tuple*; they are important for updates, for they enable us to guarantee that an update will affect precisely one tuple. Peter et al. [64] explained that with the *id* attribute in the Document Type Definition (DTD), one can uniquely identify properties of an element within an XML file.

With the same idea in [64], the PURPOSE, RECIPIENT elements and their attributes in the P3P language would only help identify the set of service purposes and recipients declared by the service provider. However, there is no connection that can help the P3P language identify the relationship between the PURPOSE and RECIPIENT elements.

In our solution, we consider the PURPOSE and RECIPIENT values in P3P as key relations to the data type collected. By doing so, we are not assuming that every data-type listed in a policy will have its own statement element.

```
<STATEMENT>
   <PURPOSE>
            <pseudo-analysis/>
            <current/>
            <contact required = 'always'>
            < telemarketing/>
   </PURPOSE>
   <DATA-GROUP>
            <DATA ref= "#user.name.firstname"/>
            <DATA ref= "#user.address.postal"/>
            <DATA ref= "#home-info.telecom.mobile"/>
   </DATA-GROUP>
</STATEMENT>
```

Figure 31: Example of Multiple Purposes and Data

Figure 31 shows an example of a STATEMENT element with three data items (#user.name.firstname, #user.address.postal, and #home-info.telecom.mobile) to be used for the following purposes (pseudo-analysis, current, contact, and telemarketing). According to the P3P specification [24], one cannot allow data items collected for the PURPOSE of

telemarketing to be used for *contact*. Moreover, the P3P user agent will not be able to identify how those purposes relate to the various data items.

The P3P code in Figure 31, depicts that the STATEMENT element can collect the data item in the DATA-GROUP element, and the data can be used for the purposes in the PURPOSE element. The example described above does not clearly state how to connect the relationships that exists in a data item, and its recipients or purposes. Finally, in agreement with the P3P specification, one can specify purpose values in STATEMENT elements that have nothing to do with a data type collected. In this thesis, we modified the XML key notion defined in [64] to improve the relationship of data items to its purposes and recipients. For example, DATA element listed in Figure 31 can uniquely have a target purpose (datapurpose) and recipient (datarecipient) that will identify a set of purposes and recipients in the P3P language.

5.3.1 Definition of Relative Data Key

A *path* is an expression involving element names (*tags* and *attribute* names) that describe a set of paths in the structured document or file [64,69]. In defining a data key, we specify the attribute (a relational terminology for purpose and recipient values) that when used together can uniquely identify the purpose and recipient values of collecting a particular data item.

To describe hierarchical key structures, we introduce the notion of a relative key discussed in [64,77]. The notion consists of a pair (Q, K) where:

- Q is a *path* and
- K is a key

Q consists of *Data-Purpose* (DP) and *Data-Recipient* (DR). We specified the target element and attributes to follow the syntax in [64].

We define the relative key as

$$(K: (DP, \{P_1; \ldots; Pn\}, DR, \{R_1; \ldots; Rn\}))$$

where *K* is the name of the key. The key is the data item collected, path expressions DP and DR are the sets of purpose values and recipient values respectively, and P_1 ;…; P_n and R_1 ,…, R_n are the key paths.

```
1   <Data ref = "#URI reference"
2                 Datapurpose = "P1;…; Pn"
3                 Datarecipient = "R1;…;Rn">
4   </Data>
```

Figure 32: Syntax for using Relative Data keys

The idea is that the path expression DP (datapurpose) identifies a set of PURPOSE values and DR (datarecipient) identifies a set of RECIPIENT values. The *Ref* is an attribute name in the DATA element that identifies the data collected. The goal of relative data keys is to specify the purpose and recipient values in a DATA element.

We express a sample P3P DATA element that uses the relative data keys as follows. For example, from Figure 32 a data element with ref value *#user.home-info-postal* represents a data item collected. We assume that the entity collecting the data will use the data item for the following purposes: current, and contact. We also assume the entity will not share the data collected with any other party.

- < Data ref = *#user.home-info-postal*
  datapurpose = {current; contact = 'always'}, datarecipient = {ours}>
  </Data>

The P3P tree in Figure 33 illustrates an example of how we represent the relative data key syntax in Figure 32.

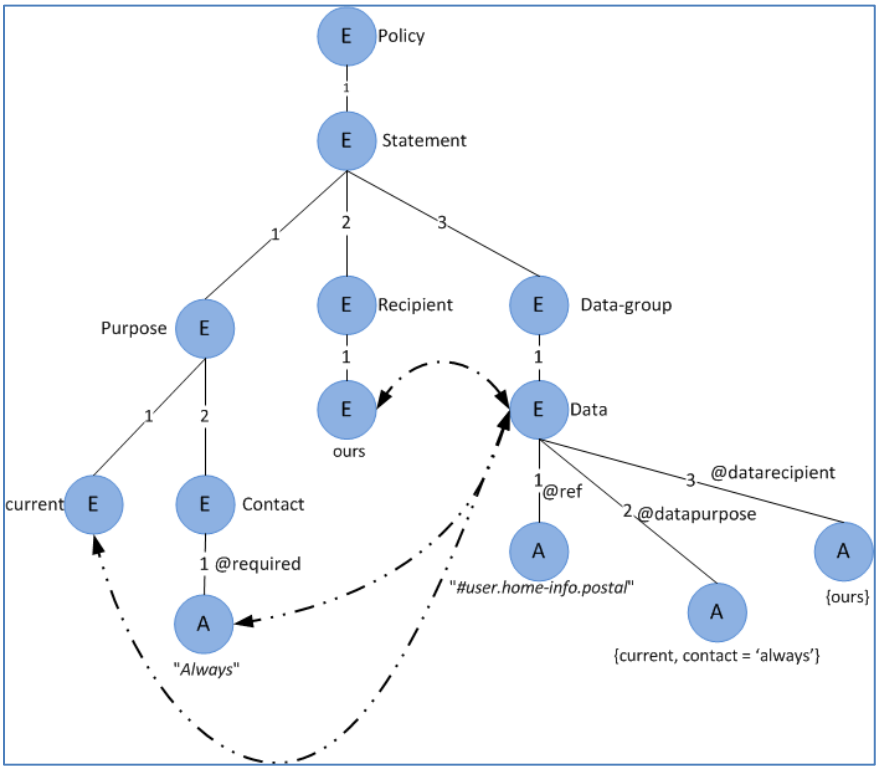


Figure 33: P3P Tree using Relative data keys

5.4 P3P Match Algorithm

In this section, we present *P3PMatch*, an algorithm to match corresponding elements in two different versions of P3P trees. We base this algorithm on the P3P relative data key explained in the last section. The P3PMatch algorithm is designed to complete its operation before the merging operation. Figure 34 depicts the P3PMatch architecture.

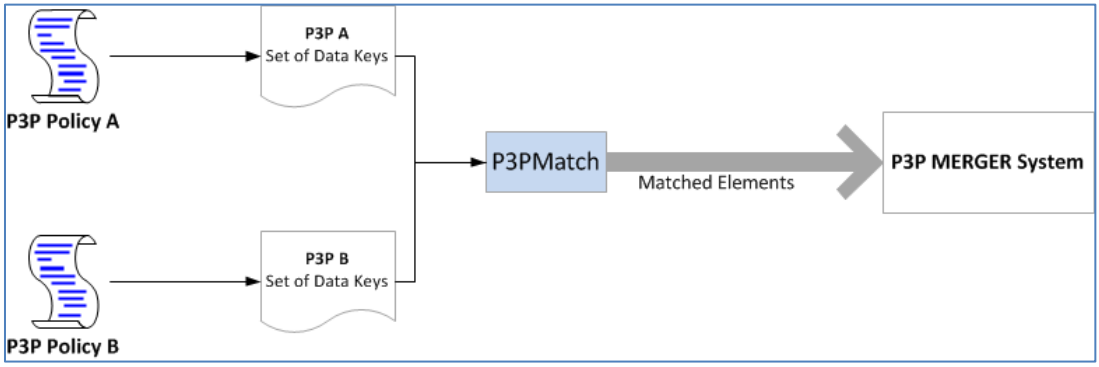


Figure 34:P3PMatch Algorithm with relative data keys

According to the general tree matching principle defined in [60], "Let X be a tree and X [i] be the i^{th} node of X in an ordered walk through the tree." The match between the nodes of a tree is a set of elements, attributes, and text-nodes that connect the corresponding nodes in both trees. We measure the correspondence by comparing the contents (elements, attributes, and text-nodes) of both P3P nodes [64,67,48].

Definition P3PMatch: A matching between two policy trees (A and B) is a set of nodes (p, q), $p \in$ A, and $q \in$ B,
Where p and q are equal by leaf nodes (content, name, or attribute). We denote matches in this thesis with the letter $\asymp$, with optional subscripts that connect the P3P trees involved in the match. For example, $\asymp_{AB}$ represents the matching operation between the P3P trees: A and B.

5.4.1 P3PMatch

The P3PMatch algorithm receives as input two P3P trees, A and B, where two sets of data keys $\sum K$ exist in A and B. The algorithm outputs a set of node pairs $\sum N$ after matching.

For example, $\sum N$ contains nodes p and q, where node $p \in$ A and node $q \in$ B. To identify $\sum N$ from the P3P elements POLICIES, POLICY, ACCESS, ENTITY, and DISPUTE-GROUPS, we compare the element names at every stage to find matching elements. However, when both policies have two or more STATEMENT elements, we use the relative data key set $\sum K$ to find the closest STATEMENTs to match. The set $\sum N$ will be the input of the merging algorithm, which aggregates the elements in the set $\sum N$.

5.4.2 Matching Similar P3P Elements

Matching nodes: Given matches $\asymp_{AB}$, the P3P elements $p \in$ A and $q \in$ B are matched if and only if the labels for elements p and q are equal and if p and $q \in \asymp_{AB}$. For merging, we use the match function to detect the changes and the similarities in different P3P policies. By inspecting the P3P policy structure and its relative data keys, we identify the changes that

transform a P3P schema into a P3P policy and we can detect the similarities in similar policy STATEMENT elements.

To match the P3P policies effectively, we divide the matching procedures into three different parts.

- **Text Match**: Two elements $p \in A$ and $q \in B$ match only by the text contents, if the nodes $(p, q) \in \asymp_{AB}$ are of type text node, where p and q are related contents of P3P elements. For example

Service A	Service B
<Statement> //Statement <Consequence> ***p*** </Consequence> </Statement> //End Statement	<Statement> //Statement <Consequence> ***q*** </Consequence> </Statement> // End Statement
Text Match	
Text matching (***p, q***)	

Figure 35: Example of Text Match

- **Label Match**: Two nodes $p \in A$ and $q \in B$ match only by the structure of the elements, if the nodes $(p, q) \in \asymp_{AB}$ are of the same structure type, where p and q are related elements syntactically. For example

Service A	Service B
<Statement> //Start Statement <***p***> <Retention> //Start Retention <ours> </Retention>// End Retention </***p***> </Statement> //End Statement	<Statement> //Start Statement <***q***> <Retention> //Start Retention <Indefinitely> </Retention> End Retention </***q***> </Statement> //End Statement
Label Match	
Label match (<***p***>, <***q***>)	

Figure 36: Example of Label Match

- ***Relative Match:*** Two elements ($p \in$ A) and ($q \in$ B) match *ref* values in DATA elements, if and only if the *ref* values in the set of *p* have non-empty intersection with the *ref* values of the set of *q*. If more than one data-group elements exist in policies A and B, then the relative match algorithm will map the sets that have the larger intersection together for merging.

Service A	**Service B**
<Data-Group> //Start Data-group <***p***> <data ref= "#user.name.family"> <data ref= "#thirdparty.name.given"> <data ref= "#business.name"> </***p***> </Data-Group> //End Data-group	<Data-Group> //Start Data-group <***q***> <data ref= "#user.name.family"> <data ref= "#business.cert.key"> <data ref= "#business.name"> </***q***> </Data-Group> //End Data-group
Relative Match	
Relative match (<***p***>, <***q***>) = 2 items {<data ref= "#user.name.family"> , <data ref= "#business.name">}	

Figure 37: Example of Relative Match

The following is a match method that combines all matches together.

Full match: Two elements ($p \in$ A) and ($q \in$ B) match fully if and only if the content and structure of the elements matches.

Service A	**Service B**
<Data-Group> //Start Data-group <***p***> <data ref= "#user.name.family"> <data ref= "#business.name"> </***p***> </Data-Group> //End Data-group	<Data-Group> //Start Data-group <*q*> <data ref= "#user.name.family"> <data ref= "#business.name"> </*q*> </Data-Group> //End Data-group
Related Match	
Full match (<***p***>, <***q***>) = true	

Figure 38: Example of Full Match

5.4.3 P3PMatch Algorithms

So far we have presented the overall process and flow of the P3PMatch algorithm. We will now give a more detailed descriptions of the algorithm: the overall structure of the P3PMatch algorithm, validating the P3P policy using the P3P validator, matching P3P elements with P3P relative key.

*Algorithm **P3PMatch***

Algorithm 1 presents the general P3PMatch algorithm for matching P3P elements. The algorithm accepts as input two P3P policies (A, and B), after comparing the P3P vocabularies or elements in both policies the P3PMatch returns as output an array of best matched element *M*[*i, j*]. The algorithm sets Integer variables *i,* and *j* to null and two Boolean variables (verifypolicy1, and verifypolicy2) to store the validation result of P3P A and B if the validation results are both true (meaning P3P A and B are valid P3P policies). The algorithm then loops through both policies to retrieve the content element in P3P A[i], and B[j]. If an element exists in both arrays, both instances will be retrieved and matched in pairs using the matchElement algorithm. The result of the match process will be stored in *M*[*i, j*].

Algorithm 1 P3PMatch

```
1.  Input: P3P policies P3P A, P3P B
2.  Output: Best matched elements after comparing the P3P elements
3.          int i, j = null
4.          boolean verifypolicy1 ← validateP3P(A)
5.          boolean verifypolicy2 ← validateP3P(B)
6.              if verifypolicy1 && verifypolicy2 == TRUE  then
7.                  i ← number of first-level subelements of P3P A
8.                  j ← number of first-level subelements of P3P B
9.                  element[][] M[i, j]
10.                 do
11.                 foreach i subelement ← A
12.                    foreach j subelement ← B
13.                             element collectElement1 ← A[i]
14.                             element collectElement2 ← B[j]
15.                            M[i, j] ← matchElement(A[i], B[j])
16.                    end for
17.                end for
18.             end if
19.         return M[i, j]
```

Algorithm ***ValidateP3P***

Algorithm 2 describes the validation of initial P3P policies (A and B), already discussed in subsection 4.3.1. The validateP3P method receives P3P policies or target URI to the policy as an input and returns a boolean value of either True or False after testing the validity of the P3P policy.

For example, we uploaded the P3P policy from an online business website (BennettGold LLP[27]). The image in Figure 39 illustrates the result of the P3P policy after testing for validity.

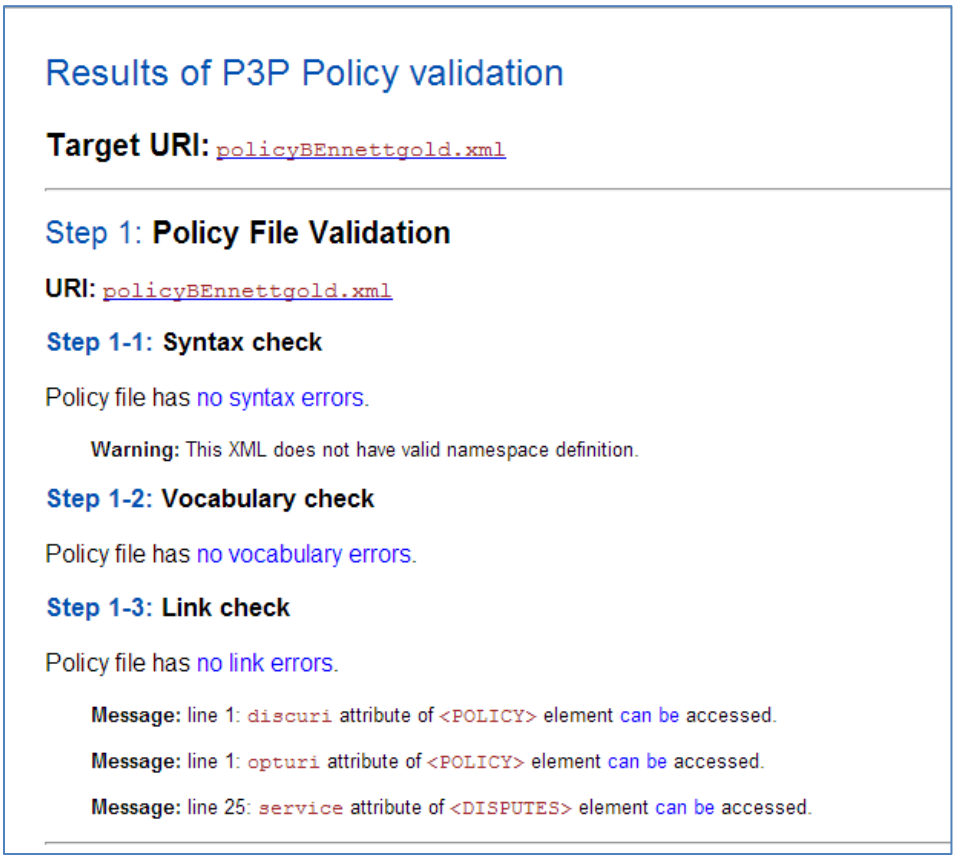


Figure 39: Sample P3P validation result

[27] http://www.bennettgold.ca/w3c/policy1.xml#Bennett_Gold_Privacy_Policy

Algorithm 2 validateP3P

```
1.  Input: P3P policy or target url to P3P policy
2.  Output: return Boolean value True or False
3.          boolean verifypolicy = null;
4.          String result ← upload policy to http://www.w3.org/P3P/validator.html
5.              or
6.          validateurl ← http://www.w3.org/P3P/validator.html
7.          String result ← validateurl.append("/" + url)
8.  // Syntax check
9.              if result contains (syntax error) then
10.                 return verifypolicy = False
11. // Vocabulary check
12.             else if result contains (vocabulary error) then
13.                 return verifypolicy = False
14. // Link check
15.                 else if result contains (link error) then
16.                     return verifypolicy = False
17.                 end if
18.             end if
19.           else return verifypolicy = True
20.         end if
```

The matchElement is presented in algorithm 3: this operation finds out the appropriate element in P3P policy A that corresponds to a given element in P3P policy B.

Algorithm 3 matchElement

```
1.  Input: element A[i] and B[j]
2.  Output: calls a match method for Entity, Access, Disputes-Group, and Statement
3.          Element elementA = A[i], Element elementB = B[j],
4.          do
5.            foreach i subElementA ← elementA
6.              foreach j subElementB ← elementB
7.              Compare the element names using the Label Match operation
8.  // Entity Match
9.            if subElementA.elementName and subElementB.elementName == "ENTITY" then
10.             matchEntity(subElementA, subElementB)
11.           end if
12. //Access Match
13.           if subElementA.elementName and subElementB.elementName == "ACCESS"
14.             then
15.             matchAccess(subElementA, subElementB)
16.           end if
17. // Disputes-group Match
18.           if subElementA.elementName and subElementB.elementName == "DISPUTE-
19.            GROUP" then
20.             matchDispute(subElementA, subElementB)
21.           end if
```

```
22. // Statement Match
23.             if subElementA.elementName and subElementB.elementName == "STATEMENT"
24.                 then
25.                 matchStatement(subElementA, subElementB)
26.             end if
27.           end for
28.         end for
```

*Method **elementName***

Method *elementName* (in Algorithm 3) returns the attribute value for the P3P element with a given name or returns null if there is no such attribute value assigned. The method may also return the empty string if the attribute value is empty.

For example, if elementA = <STATEMENT>, then elementA.*elementName* = "STATEMENT."

The algorithm uses the *elementName* in one policy to find a similar element with equal names in the other policy. The algorithm later groups identical elements, which are later distributed to appropriate match classes (e.g. matchEntity, matchAccess, matchDispute, or matchStatement).

Algorithm 4 matchEntity

```
1.  Input: element elementA and elementB
2.  Output: Matched result in M[element, element] with a match type to mergeEntity
3.          int i = elementA.size
4.          int j = elementB.size
5.
6.        foreach i subElementA ← elementA
7.          foreach j subElementB ← elementB
8.            if subElementA.elementName && subElementB.elementName == "DATA-GROUP"
9.                then
10.  // Repeat the loop process from line 3 to 7 for the element "DATA-GROUP"
11.                 if subElement2A.elementName && subElement2B.elementName == "DATA"
12.                     if Label Match condition == TRUE then
13.                     mergeEntity(subElement2A, subElement2B)
14.                     end if
15.                 end if
16.             end if
```

Algorithm ***matchEntity***

The *matchEntity* described in Algorithm 4, uses the matching type label match (explained in subsection 5.4.2) to find out the data group element in P3P policy A that corresponds to a given element in P3P policy B. The matchEntity collects matched pairs of data-group elements along with the match type, and then sends the collection to the mergeEntity method.

Method ***size***

Method *size* (lines 3 – 4) is a public integer method that returns the number of elements in the list and uses the result to initialize the integer variables (*i, j*) declared on lines 3 – 4.

For example, the XML code below has a parent "purpose" and the *size* ( ) of this element equals four. E.g. purpose.*size* ( ) = 4.

```
<PURPOSE>
        <pseudo-analysis/>
        <current/>
        <contact required = 'always'>
        <telemarketing/>
</PURPOSE>
```

Algorithm ***matchAccess***

The *matchAccess* described in Algorithm 5, explains the matchAccess for matching Access elements. The algorithm uses the matching type (Label match) to access element in P3P policy A and B, and passes along both elements with their match type to the mergeAccess class (if and only if the sub-elements are not empty or null).

Algorithm 5 matchAccess

```
1.  Input: element elementA and elementB
2.  Output: Matched result in M[element, element] with a match type to mergeEntity
3.          subElementA = elementA.childElement
4.          subElementB = elementB.childElement
5.
6.          if subElementA.elementName && subElementB.elementName != "NULL OR EMPTY"
7.                  then
8.                    mergeAccess(elementA, elementB)
9.          end if
```

Algorithm ***matchDispute***

The *matchDispute* in Algorithm 6 uses the matching type (Label and Text match) to group the resolution type elements in P3P policy A and B, then passes along the matched pair to the mergeEntity class.

Method ***remove***

The *remove* method (lines 17 and 18) is a public integer method that returns and removes the element at the specified position in the list. . Any subsequent elements are shifted to the left (one is subtracted from their indices). Parameters: *Index* – is the index of the element to be removed

Method ***add***

The *add* method inserts the specified elements at the specified position *M[k, l]* in the list. It shifts the element currently at the position (if any) and any subsequent elements to the right (it adds to the indices specified). Parameters: *index* – index at which the specified element is to be inserted and *element* – the Element to be inserted.

Algorithm 6 matchDispute

```
1.  Input: element elementA and elementB
2.  Output: Matched result in M[element, element] with a match type to mergeDispute
3.          int i = elementA.size ( )
4.          int j = elementB.size ( )
5.
6.          foreach i subElementA ← elementA
7.           foreach j subElementB ← elementB
8.            if subElementA.elementName && subElementB.elementName == "DISPUTES"
9.                  then
10.                       int k = subElementA.size ( )
11.                       int l  = subElementB.size ( )
12.                         foreach k elementA ← subElementA
13.                           foreach l elementB ← subElementB
14.                 if elementA.elementName && elementB.elementName == "RESOLUTION"
15.                  if Label Match && Text Match condition == TRUE then
16.                     M[k, l].add ( ) = (elementA, elementB)
17.                       subElementA.remove(k)
18.                       subElementB.remove(l)
19.                       end if
```

```
20.                 end if
21.             end for
22.             end for
23.
24.             if subElementA.size ( ) != 0 then
25.               M [k + subElementASize( ), (l+1)].addAll = subElementA.childElement
26.              end if
27.
28.             if subElementB.size ( ) != 0 then
29.               M[k + subElementBSize( ), (l+1) ].addAll = subElementB.childElement
30.              end if
31.
32.           end if
33.         end for
34.       end for
35.     mergeDispute(M[k, l])
```

Method ***addAll***

The method addAll ( ) inserts more than one element in a specified collection into the list, starting at the specified position. The method shifts the element currently at the position (if any) and any subsequent elements to the right (increases their indices). The new elements will appear in the list in the order returned by the specified collection's iterator.

Parameters: *index* – index at which to insert the first element from the specified collection. *collection* – collection containing elements.

Algorithm ***matchStatement***

The *matchStatement* algorithm described in Algorithm 7, uses the matching type (Label and Text match) to group the resolution type elements in P3P policies A and B. However, label and text match is not enough to match the statement element, as discussed in chapter 4; the use of a P3P relative data key must be checked. If the data element contains the relative data key, then the relative match will return the number of data-purpose and data-recipient occurrences for a data.

Algorithm 7 matchStatement

```
1.  Input: element elementA and elementB
2.  Output: Matched result in M[element, element] with a match type to mergeDispute
3.          int i = elementA.size
4.          int j = elementB.size
5.          int data[ ]
6.          int count = 0
7.
8.          foreach i subElementA ← elementA
9.            foreach j subElementB ← elementB
10.            if subElementA.elementName && subElementB.elementName == "DATA-GROUP"
11.              then
12.                   int i = subElementA.size
13.                   int j = subElementB.size
14.                   data[count] = countDataOccurance(subElementA, subElementB)
15.              increment count
16.            end if
17.           end for
18.         end for
19.
20.         Sort(data[count], subElementA, subElementB) ← highest to lowest
21.         i  Max(subElementA.lenght, subElementB.lenght)
22.
23.         foreach i elements[ ] ← data[count]
24.           M[ ]  ← pairElement(subElementA[i], subElementB[i])
25.         end for
26.
27.         mergeStatement(M[ ])
```

Method ***countDataOccurance***

The countDataOccurance ( ) used on line 14, is a public integer method that returns the number of occurrences of an attribute ("ref" with a unique value) in Policies A and B. Parameters: *element1, element2* – List of elements to compare.

Method ***sort***

The sort ( ) method on line 20, sorts the specified array of elements in descending order, according to the DATA occurrence of its DATA-GROUP element. This sort is to be stable: equal elements will not be reordered because of the sort. Parameters: *data [index]* - the array of data occurrences counted. *element*– List of elements and their properties to be sorted.

Method ***pairElement***

The *pairElement* ( ) method on line 24, is a public method that checks the collection statement elements in Policies A and B, and pairs them based on the sort results. The method returns a match array *M* [*element, element*] that contains paired sorted statement elements, then *remove* ( ) the paired group from the statement array.

5.5 Conclusion

In this chapter, we have presented the P3P Match algorithm to match corresponding elements in two different P3P trees. We described how the processes required in matching multiple P3P policies relate to the P3P match process. We explained how the inner classes and methods in this algorithm interact. We then adopted the relative data key concept in Relational Database Management (RDBM) to extend the data element defined in the P3P specification; the P3P relative data key. Lastly, we defined the match types (Text, Label, Relative, and Full match) and the P3P relative data key (an extension to the P3P specification) required for matching privacy policies.

In the next chapter, we will present the P3P Merging algorithm for aggregating and integrating P3P privacy policies. We will begin by introducing the overview of our P3P aggregator framework, which describes the framework for merging policies of service providers on an aggregated platform.

Chapter 6

The P3P Merging Algorithm

This chapter explains our concept for merging P3P policies in a cloud aggregation platform. We propose an algorithm that implements the merging process described in section 4.2. Part of our methodology is to describe a framework for merging policies of service providers on an aggregation platform. To achieve this goal, we use the Three-Way Merge principle for aggregating structured data and constraints for the P3P language.

6.1 Design Overview

Following the introduction of our merging process in chapter 4, here we present the framework for using the P3P aggregator in the cloud-computing environment. This may require a formal semantic model, which should help avoid loss of data and correct semantic errors when creating a new policy. To perform the merger on an aggregation platform for SaaS, we illustrate the layout of our concept in Figure 40 and define merger constraints.

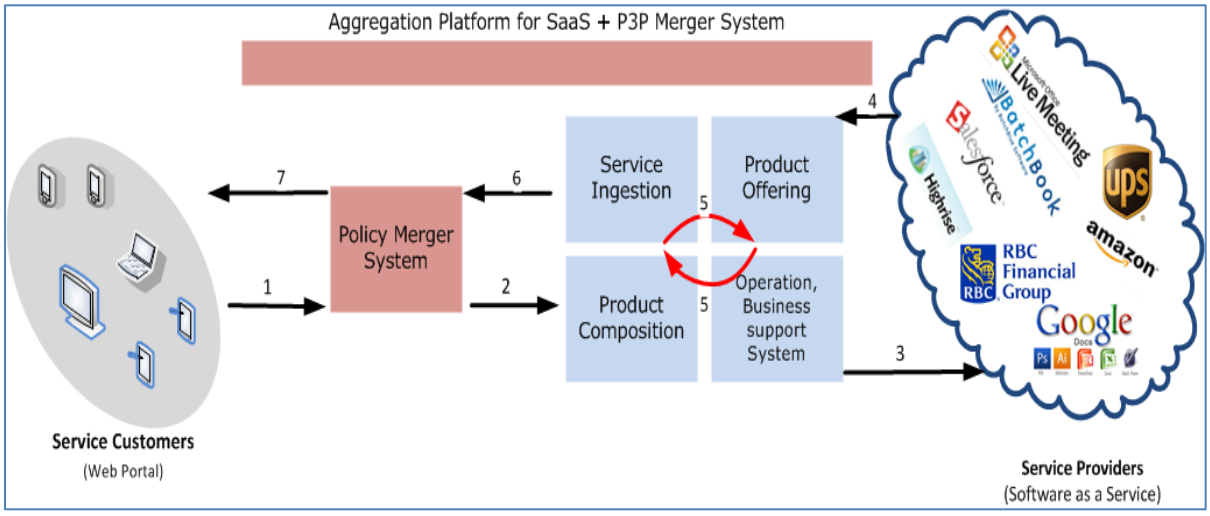


Figure 40: Aggregation Platform for SaaS & P3P Merger System

6.2 Defining P3P Merge Constraints

The purpose of defining P3P policy constraints is to include the semantic rules in the merging algorithms defined in [43,82] and discussed in section 3.2.1.

Note that all the following example were actually executed and in this sense they constitute a fairly complete set of test cases for our merge algorithm.

6.2.1 Policy Element

The POLICY element is the root element that starts the P3P policy; the element should have the following attributes: *names, discuri,* and *opturi*. To create a new merged P3P policy, the cloud aggregator must declare a new *name*, *discuri*, and *opturi* for the merged policy.

For example, if two services A and B in the cloud are merged to produce service C, the service aggregator is responsible to give a new *name, discuri,* and *opturi* for service C's P3P policy.

Policy A	**Policy B**
<Policy name = "service A" opturi = "http://www.serviceA.com/opturi" discuri = "http://www.serviceA.com/discuri" > </Policy>	<Policy name = "service B" opturi = "http://www.serviceB.com/opturi" discuri = "http://www.serviceB.com/discuri" > </Policy>
Aggregate Policy Element	
<Policy name = "Aggregate Policy" opturi = "new opturi address" discuri = "new discuri address" > </Policy>	

Figure 41: Example of Aggregate POLICY Element

6.2.2 Entity Element

The ENTITY element gives a precise description of the legal entity that owns the privacy policy. The P3P specification [24] permits service providers to declare their company information using the DATA-GROUP extension.

For example, if we merge services A and B, the new ENTITY element ($Entity_{new}$) will merge the company's service information into a different DATA-GROUP element in the same new ENTITY element.

Rule 1: If two or more ENTITY elements have the same data text values for *#business.name* (representing the company name), the aggregate result will only inherit one data-group for ENTITY because the P3P merger will assume that the providers are the same.

Algorithm 8 mergeEntity

```
1.  Input: element elementA and elementB
2.  Output: returns a merged result for Entity Element
3.      element Entity = null
4.      if elementA.elementName and elementB.elementName == "DATA-GROUP"
5.          then  Loop through elementA and elementB
6.              boolean result ← Compare all attribute values in elementA and B
7.              if (result == "True") then
8.                  Entity. add(elementA or elementB)
9.              end if
10.             if (result == "False") then
11.                 Entity. add(elementA)
12.                 Entity. add(elementB)
13.             end if
14.         end if
15.     end if
16.  return Entity
```

For Example

Service A	**Service B**
<Entity> <Data-Group> <Data ref="#business.name"> Service_A </Data> <Data ref="#business.address"> 180 King Edward Drive</Data> </DataGroup> </Entity>	<Entity> <Data-Group> <Data ref="#business.name"> Service_B </Data> <Data ref="#business.address"> 243 Hurmand Drive</Data> </DataGroup> </Entity>

Aggregate Entity Element

<Entity>
<Data-Group>
<Data ref="#business.name"> Service_A </Data>
<Data ref="#business.address"> 180 King Edward Drive</Data>
</DataGroup>
<Data-Group>
<Data ref="#business.name"> Service_B </Data>

```
    <Data ref ="#business.address">  243 Hurmand Drive</Data>
  </DataGroup>
</Entity>
```

Table 5: Example of Aggregate ENTITY Element

6.2.3 Access Element

The ACCESS element indicates if the service provides access to various kinds of user information. The ACCESS element must contain one of the following child elements:

- NONIDENT element (no identifying data is collected)
- ALL element (the user can access all identifiable information)
- IDENT-CONTACT element (the user can access only identifying and contact information)
- CONTACT-AND-OTHER element (the user can access contact information and other identifiable)
- OTHER-IDENT element (other identifiable data)
- NONE element (the user cannot access any information)

Algorithm 8 mergeAccess

```
1.  Input: element elementA and elementB
2.  Output: returns a merged result for Access Element
3.        element Access ← null
4.        element subElement ← null
5.        if elementA.elementName && elementB.elementName != null then
6.              subElement ← accessRule(elementA, elementB)
7.              Access.add(subElement)
8.        end if
9.    return Access
```

For example, if we merge the two P3P policies A and B, the new ACCESS element (*Access* Element) will be equal to the *Access* element of policy A or the *Access* element of policy B, if and only if *Access* element for both A and B are identical. Otherwise, the method accessRule ( ) determines the access value between the two access elements.

The accessRule (*element, element*) method used on line 6, is a public element method that returns the element at the specified access element. The method accepts access values from P3P policy A and B as input and compare the values based on the access values on the privacy information.

Rule 2: Access values can be prioritized according to the privacy information they possess:

All **<** Contact-and-other **<** Ident-contact **<** Other-ident **<** Nonident **<** None

For Example, Table 6 describes access values of two services A and B, if service A allows user access <all/> and service B allows access <ident-contact/>.

Service A	**Service B**
<Access> <all/> </Access>	<Access> <ident-contact/> </Access>
New Service	**RULE**
<Access> <ident-contact/> </Access>	<all/> **<** <ident-contact/>

Table 6: Example of aggregate Access Element

6.2.4 Disputes-Group Element

The DISPUTES-GROUP element has one or more DISPUTES elements. The DISPUTES element describes the resolution rules that users will follow in case of a user-provider dispute. The DISPUTES element requires the resolution type (*Service* | *Independent* | *Court* | *Law*), the URI of the service to resolve the dispute, the URL for verification (which should include the REMEDIES element) and the long and short description of the DISPUTES element.

Rule 3: The merge system should aggregate service values (*Service* | *Independent* | *Court* | *Law*) in the dispute element based on equality, and if the values are not equal then the apply method will pick a greater value in the following order: *Correct* **<** *Money* **<** *Law*.

Algorithm 8 mergeDisputes

```
1.  Input:  element elementA and elementB
2.  Output:  returns a merged result for Disputes-Group Element
3.          element Disputes-Group = null
4.          element Disputes = null
5.            int i = elementA.size ( )
6.            int j = elementB.size ( )
7.            foreach i subElementA ← elementA
8.             foreach j subElementB ← elementB
9.              if subElementA.elementName.equal(subElementB.elementName) && != null
10.                   Disputes.add( ) ← disputesRule ( subElementA, subElementB)
11.                   Disputes-Group.add( ) ← Disputes
12.                   Disputes.clear( )
13.                   elementA.remove(subElementA)
14.                   elementB.remove(subElementB)
15.             end if
16.           end for
17.         end for
18.
19.         if(elementA.size( ) <= 0 ) then
20.           Disputes-Group.addAll(elementA)
21.         end if
22.
23.         if(elementB.size( ) <= 0 ) then
24.           Disputes-Group.addAll(elementB)
25.         end if
26.   return Disputes-Group
```

For Example

Service A	**Service B**
<Disputes-Group> <Dispute> <Resolution ="Service"> <Remedies> <Money/> </Remedies> </Dispute> <Dispute> <Resolution ="Court"> <Remedies> <Correct/> </Remedies> </Dispute> </Disputes-Group>	<Disputes-Group> <Dispute> <Resolution ="Service"> <Remedies> <Law/> </Remedies> </Dispute> </Disputes-Group>
New Service	
<Disputes-Group> <Dispute> <Resolution ="Service"> <Remedies> <Money/> </Remedies> </Dispute>	Service = Service Apply rule (Correct < Money < Law)

<Dispute> <Resolution ="Court"> <Remedies> <Correct/> </Remedies> </Dispute> </Disputes-Group>	

Table 7: Example of aggregate Disputes-group Element

6.2.5 Statement Element

Algorithm 9 mergeStatement

```
1.  Input: element elementA and elementB
2.  Output: returns a merged result for Disputes-Group Element
3.      element Statement = null
4.        int i = elementA.size ( )
5.        int j = elementB.size ( )
6.      foreach i subElementA ← elementA
7.       foreach j subElementB ← elementB
8.         if subElementA.elementName.equal (subElementB.elementName) && !=
9.          "NON-IDENTIFIABLE"
10.              if subElementA.elementName.equal(subElementB.elementName) &&
11.               subElementA.element.equal("CONSEQUENCE") then
12.                statement.addUnique(subElementA, subElementB)
13.              end if
14.              if subElementA.elementName.equal(subElementB.elementName) &&
15.               subElementA.element.equal("PURPOSE") then
16.                statement. addUnique (subElementA, subElementB)
17.              end if
18.              if subElementA.elementName.equal(subElementB.elementName) &&
19.               subElementA.element.equal("RECIPIENT") then
20.                statement.addUnique (subElementA, subElementB)
21.              end if
22.              if subElementA.elementName.equal(subElementB.elementName) &&
23.               subElementA.element.equal("RETENTION") then
24.                mergeRetention(subElementA, subElementB)
25.              end if
26.              if subElementA.elementName.equal(subElementB.elementName) &&
27.               subElementA.element.equal("DATA-GROUP") then
28.                statement.add
29.               foreach l dataElementA ← subElementA
30.                foreach k dataElementB ← subElementB
31.                 if dataElementA.elementName.equal(dataElementB.elementName) &&
32.                  dataElementA.element.equal("DATA") then
33.                  statement(DATA-GROUP).addUnique(dataElementA, dataElementB)
34.                 end if
35.               end for
```

```
            end for
        end if
    end for
end for
return statement
```

The STATEMENT element is a container that groups together a PURPOSE element, a RECIPIENT element, a RETENTION element, a DATA GROUP element, a NON-IDENTIFIABLE element and a CONSEQUENCE element. When the STATEMENT element in P3P does not collect any data from the user, the childnode of that STATEMENT element will have a NON-IDENTIFIABLE element.

For example

Service A	**Service B**
<Statement> //Statement 1 <non-identifiable/> </Statement> <Statement> //Statement 2 < non-identifiable/> </Statement>	<Statement> //Statement 1 <purpose/> <retention/> <recipient/> </Statement>
New Service	
<Statement> //Statement 1 <purpose/> <retention/> <recipient/> </Statement> <Statement> //Statement 2 < non-identifiable/> </Statement>	

Table 8: Example of aggregate NON-IDENTIFIABLE Element

6.2.6 Retention Element

The RETENTION element indicates the time of retaining the collected information, and it describes the kind of retention policy that applies to the data referenced in that STATEMENT. The element must contain one of the following pre-defined retention values: *indefinitely, no retention, stated-purpose, legal-requirement, or business-practices.*

For example, when merging two services (A and B); the new RETENTION element of the aggregate statement depends on the greater retention value between the two matched values.

Rule 4: The merge system should aggregate retention values based on their longevity

Indefinitely **>** business-practices **>** legal-requirement **>** stated-purpose **>** no-retention

For example

Service A	**Service B**
<Statement> //Statement 1 <Consequence> *As a consequence of the use of the service, or by virtue of having sent an e-mail through the "contacts"…* </Consequence> </Statement> <Statement> //Statement 2 Null </Statement>	<Statement> //Statement 1 Null </Statement>
New Service	
<Statement> //Statement 1 <Consequence> *As a consequence of the use of the service, or by virtue of having sent an e-mail through the "contacts"…* </Consequence> </Statement> <Statement> Null </Statement>	

Table 9: Example of aggregate Consequence Element

Algorithm 8 mergeRetention

```
1. Input: element subElementA and subElementB
2. Output: returns the aggregate value of a retention element
3.         element retention ← null
4.             if subElementB.elementName && subElementA.elementName != null then
5.                retention ← accessRule(subElementA, subElementA)
6.                retention.add(subElement)
7.         end if
8.     return retention
```

For example

Service A	Service B
<Statement> //Statement 1 <Retention>Stated-purpose<Retention/> </Statement>	<Statement> //Statement 1 <Retention>Legal-requirement <Retention/> </Statement>
New Service	
<Statement> //Statement 1 <Retention>Legal-requirement <Retention/> </Statement>	

Table 10: Example of aggregate Retention element

6.3 P3P Merge Framework

In this section, we extend the model of service aggregation in [78]. We apply the P3P policy aggregator module to the service aggregation model.

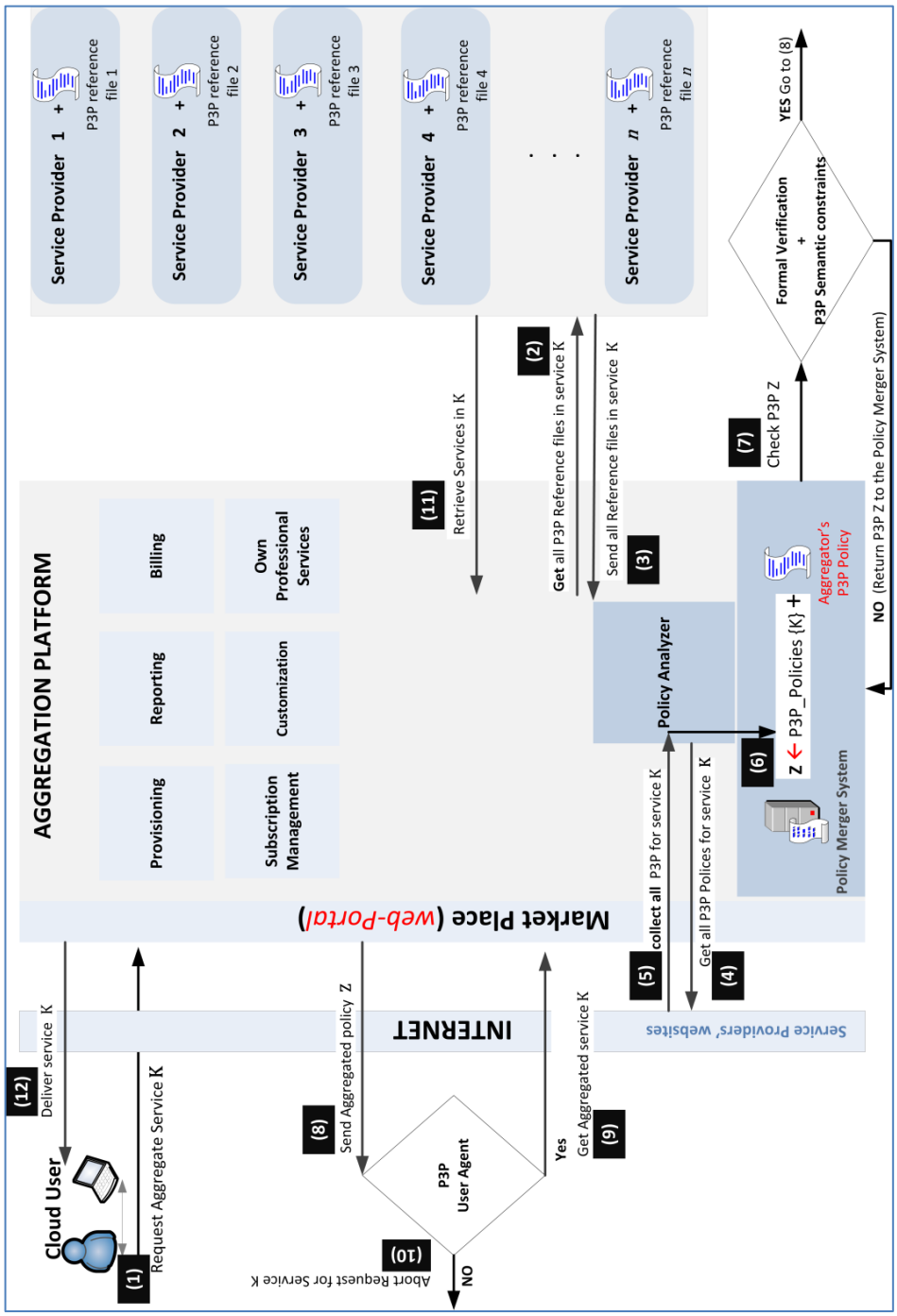


Figure 42: Design Overview of Policy Merger in an Aggregation Platform

6.3.1 P3P-Enabled Service Providers

When service providers create and deploy P3P policies, they will have to configure the server to issue P3P response headers to the cloud upon request. Here is an overview of the file and HTTP responses involved in the process.

- *Create P3P policy*: The policy statements of a service provider are the subset of the services offered; the data handling practices have always influenced the policy statements. A P3P policy[28] is an XML document that describes those policy statements. The P3P policy can either be a standalone file, or be included in a policy reference file.
- Create Policy Reference File: A P3P policy reference file is an XML file that tells user agents which P3P policy applies to which URLs and cookies on a web site [24]. The service providers' policy reference files contain addresses to P3P policies. Every P3P-enabled service provider must have at least one policy reference file [24].

6.3.2 Aggregate Service *K*

We defined a service user in section (2.7.2) as an individual or entity, which makes use of aggregate services on the aggregation platform.

According to the framework in Figure 42, the user-request for aggregate service is denoted K. Service K is the set of all services that are included in the requested service.

$$K \in \{service\ 1, service\ 2, \ldots, service\ j\}$$

The P3P specification includes a protocol for requesting and transmitting P3P policies, which we discussed in (Section 2.2.2). The protocol relies on the HTTP protocol that the web browsers use to communicate with the aggregator's market place (web portal). As shown in Figure 42, the customer sends the list of services in K via the marketplace to the service aggregator.

[28] A service provider may have more than one privacy policy

6.3.3 Retrieving P3P Reference and Policy files

In a typical client/server model, the web browser uses the HTTP protocol to communicate with the servers over the Internet. With this protocol, the web browser can retrieve P3P policies from the server. In this framework, the service aggregator uses a standard HTTP request to fetch P3P policy reference files for service K.

Service K equals a list of services $\{service\ 1,\ service\ 2,\ldots,service\ j\}$ from different service providers. On obtaining the appropriate policy reference files, the service aggregator can use those reference files to locate the P3P policy files corresponding to the services $\{service\ 1,\ service\ 2,\ldots,service\ j\}$.

6.3.4 Policy Merger System

The policy merger system is associated with a method that creates the aggregate P3P policy. The policy merger system merges the P3P policies retrieved from the service K with the aggregator's P3P policy. We denote the aggregate P3P policy to be Z, as shown in Figure 43.

We adopt the aggregate and operator notation used in [30] to represent the aggregate operation in our model. By definition, $\oplus$ denotes an aggregate operator, which represents the intersection of P3P elements, union of related elements, or conflicts. Our algorithm for this operator is presented in Section 6.2.

We denote the assignment operation as $\leftarrow$, where polices in K and the aggregator's policy are aggregated to form a new policy Z. Note that although we considered (K + 1) P3P policies in Figure 43, they are merging two by two according to the three-way merging method discussed in chapter 4.

Figure 43: Policy merger stage from Figure 49

$$\text{policy Z} = (\text{policy 1} \oplus \text{policy 2} \oplus \ldots \oplus \text{policy j}) \oplus \text{aggregator's P3P policy}$$

6.3.5 Consistency Check for Policy *Z*

To formally verify and semantically check inconsistencies in the P3P policy *Z*, we implement the P3P constraints introduced in section 3.2.1.1 using a SAT solver called "Alloy Analyzer."

We rely on the Alloy Analyzer for model checking; it must be verified that the semantics and syntax of policy *Z* conform to the P3P1.1 specifications. The model in the Alloy analyzer will also check for any conflicts that may exist in the policy before it reaches the user.

6.3.5.1 Alloy Concepts

The alloy is a first order relational logic, and is intended to model complex structures through extensive set manipulation, and this manipulation permits an easy translation from mathematical models (usually in a set or object-oriented form) by using the Alloy Analyzer tool.

An Alloy model consists of sets of concrete objects, called signatures (*sig*), facts about these sets, and relations on these sets. In the alloy language, distinguished subsets of signatures can be created; these new signatures are said to extend the superset of an object. Unlike some other modeling languages, Alloy does not require that these relations are completely specified.

- Signature: A signature (*sig*) declares one or more sets of atoms, and their relations to the other sets.
- Function: A function (*fun*) defines relations (sets, or atoms). It can take parameters that are used in getting its results. It can define a relation (usually using the implication→) and make use of it in order to produce relationship results. Such functions are a FOL (First-Order Logic) function for the Alloy logic, in which expressions are relations.
- Predicate: A Predicate (*Pred*) defines a formula that one assumes to be valid. The Alloy analyzer uses facts as axioms in constructing its examples and counterexamples.

- Assertion: An Assertion (*assert*) defines a formula that is claimed to always be true. It differs from a *fact*, because the Alloy analyzer will always check an assertion to make sure that it is true for all other modules. However, the analyzer assumes that each fact is used as a constraint, which is only true for the module it is declared in.

6.3.5.2 Alloy Model of Aggregate P3P Policy Z

In this section, we show a logical model for the P3P policy Z. To obtain the alloy model we need to translate the logical notations into the Alloy notion. Alloy defines sets to be signatures. The signatures connect the events of elements in a function. We translate constraints in the P3P language defined in section 3.4.1.1 into Alloy's facts, assertions, or functions.

For example, the P3P policy expressed as POLICY = {ACCESS, ENTITY, DISPUTES-GROUP, STATEMENT} can be translated in Alloy to the diagram shown in Figure 44.

```
absrtact one sig Policy{
   statement: some Statement,
   entity: one Entity,
   access: one Access,
   disgroup: one DisputesGroup
   }
```

Figure 44: Modeling a P3P policy element in Alloy

```
one sig Statement {
        purpose: one Purpose,
        retention: one Retention,
        recipient: one Recipient ,
        data : some DataGroup
        }
```

Figure 45: Modeling a P3P statement element in Alloy

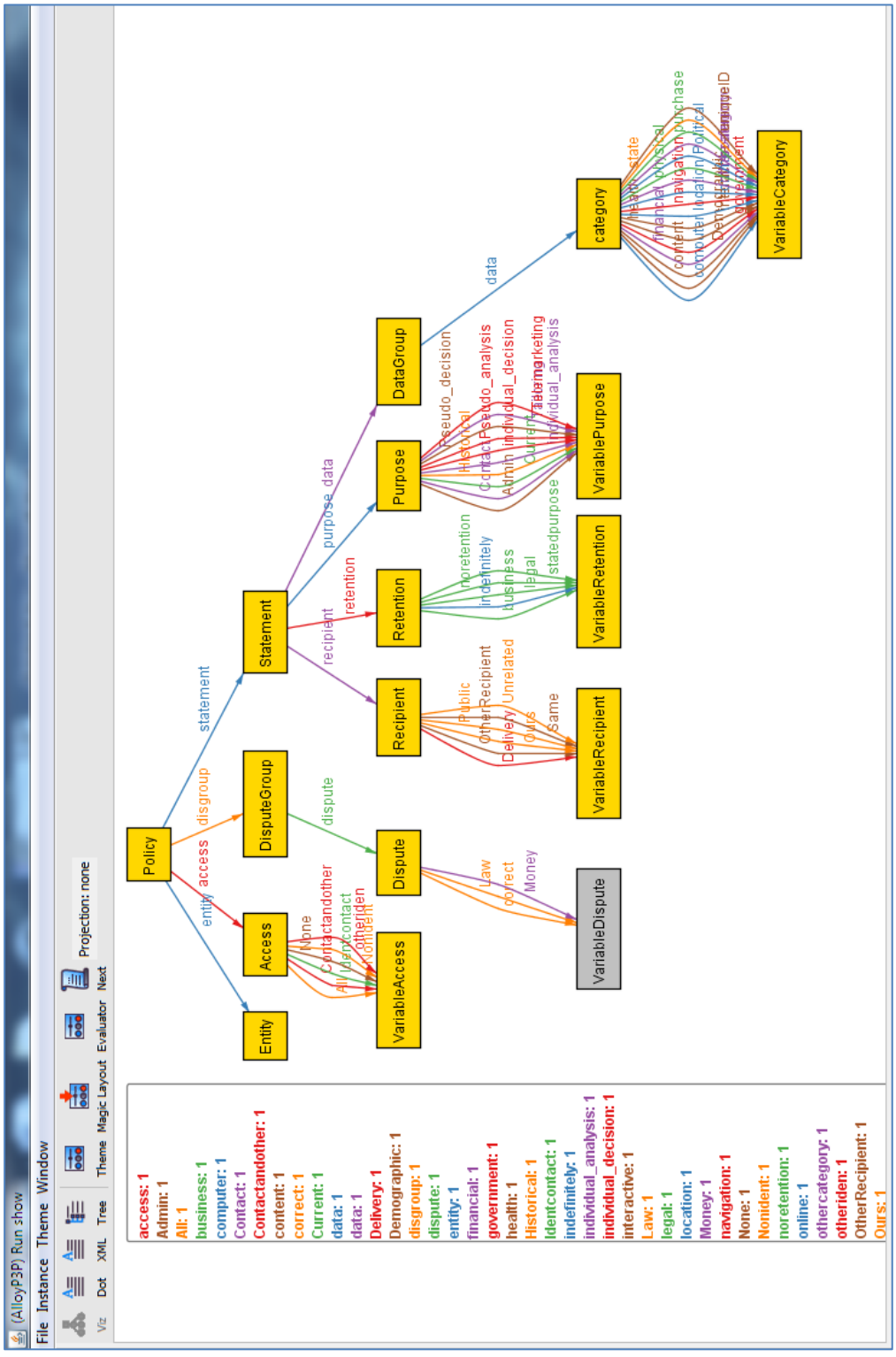


Figure 46: Alloy Model of P3P Privacy Policy Z

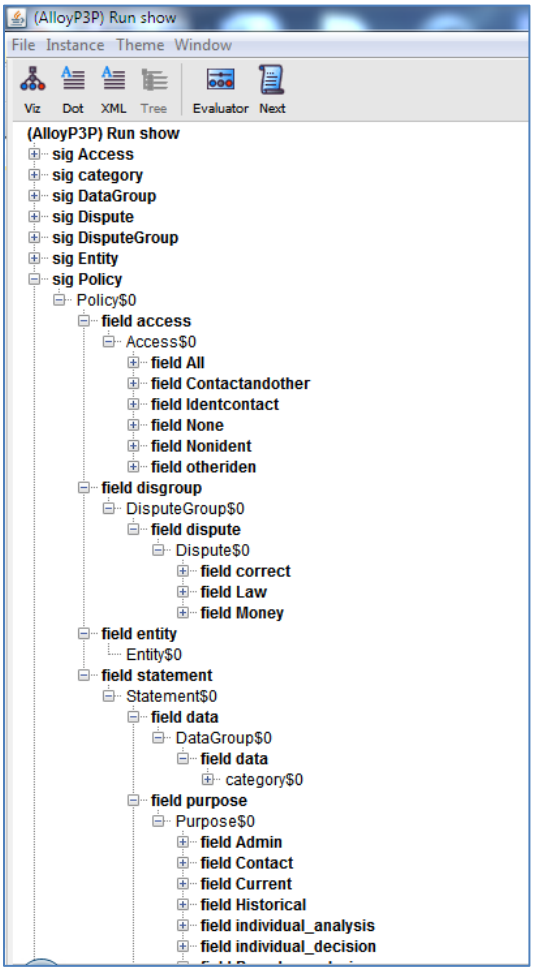


Figure 47: Alloy Tree Model of P3P Privacy Policy Z

6.3.5.3 Checking Consistency in policy Z

Consistency is a condition for aggregating P3P models, since there should not be conflicts in an aggregate policy. We will therefore check the P3P policy Z to determine if there were any inconsistencies. The constraints defined in sections 3.2.1 can be translated into facts that check for P3P conflicts. For example, Figure 48 illustrates a P3P constraint translated to Alloy.

P3P CONSTRAINT

$$\forall p, q \in DPRD, p.(data, purpose) = q.(data, purpose)$$

$$\rightarrow p.retention = q.retention$$

```
fact dataRention{
   all dataP, dataQ - some ElementData
   all dataPurpose - some ElementPupose
   all retentionP, retentionQ - one ElementRention

      ((dataP = [data, dataPurpose]) and
      (dataQ = [data, dataPurpose]))
        ==>
      (retentionP == retentionQ)
}
```

Figure 48: P3P Constraint in Alloy Language

The Alloy analyzer will take policy constraints such as the one given in Figure 48 to verify whether a P3P policy is satisfied. In other words, it checks constraints against a given P3P policy, and outputs true if the P3P policy comply with the constraints or false otherwise.

6.3.6 P3P User Agent Checking Policy *Z*

After verifying consistency in a policy Z, the P3P user agent on the user's web browser retrieves the policy, which is then compared with the user's privacy preferences in the browser. In the case where the policy matches with the user's privacy preferences, the browser will request the aggregate service. However, if the policy does not match the user's preferences, the browser will inform the user about the privacy warnings and attempt to abort the service request. User intervention is necessary to recover from this situation. Most probably, the user will either have to modify its preferences or renounce using the service.

6.4 Conclusion

In this chapter, we have discussed the P3P Merge algorithm, an algorithm to aggregate corresponding elements that have been matched by the P3P match algorithm. We described how some semantic rules required in merging P3P policies are added through the *apply* method. We have explained our approach by showing how the methods in the merge algorithm interact with the P3P elements used in a policy. We have then presented our contribution to the aggregation platform by developing a model of an aggregation framework to merge the P3P policies of service providers on an aggregation platform. We adopted the

Alloy Analyzer (a SAT solver) as an extension to determine if the aggregate policy is consistent or not. We call this the policy validation process or consistency check.

In the next chapter, we will present the implementation of our P3P policy aggregation tool, the tool is part of the goal of this thesis. We use the Java programming language and Dom4J as the main programming languages. We will also present some example scenarios of how the tool developed can be used.

Chapter 7

The Merger Implementation

In this chapter, we explain some core modules in the prototype implementation of the P3P merger system. The tool performs a syntactic and semantic three-way merge operation on two P3P policies. We also describe the technologies used in the implementation of the merger system.

7.1 Implementation Technologies

As part of the goal of this thesis, we develop a P3P policy aggregation tool, which can interact with service providers in the cloud, and over the Internet. Our choice of implementation technologies was driven by cloud platform standards, the need for platform independence, and the language requirements from cloud Platform as a Service (PaaS) providers such as Google App Engine (GAE). The GAE supports apps written in several programming languages like Java™, Python, and the Go programming language[29].

Cloud aggregators run on the PaaS platform; this platform lets users run web applications on the cloud provider's infrastructure. The aggregator can easily build, maintain, and scale applications in the language's runtime environment (e.g. Java runtime environment JRE). For example, if an aggregator runs the JRE platform in the cloud, add-on or plug-in applications can be developed using the standard Java technologies, including the Java virtual machine, Java servlets, JavaScript, Ruby, and the Java programming language.

In this thesis work, we used Java[30] as the main programming language, HTTP as the transport protocol, DOM4J[31] as the DOM API and XML parser, and P3P1.1 as the format for all aggregate policies. We used Java™ Platform, Standard edition 7 APIs. We also relied on

[29] What is Google App Engine?
https://developers.google.com/appengine/docs/whatisgoogleappengine
[30] Java™ Platform, Standard Edition 7 API specification. http://docs.oracle.com/javase/7/docs/api/
[31] DOM4J, an open source library for working with XML; http://dom4j.sourceforge.net/

the Eclipse Integrated Development Environment IDE for Java developers[32] to edit and create our code.

7.1.1 DOM4J XML Parser

Dom4J is an open source XML framework library for parsing XML, XPath and XSLT on the Java platform. It uses the Java-Collections framework and has full support for other XML parsers such as DOM, SAX and JAXP.

The main features of Dom4J are listed below:

- Was designed for the Java platform with full support for the Java collection framework
- Provides full support for JAXP, TrAX, SAX, DOM, and XSLT
- Has fully integrated XPath support for easy navigation of XML documents
- Is based on Java interfaces for flexible plug and play implementations
- Provides support for XML schema Data Types

7.2 Design

To merge two or more policy files, the aggregator tool has to parse the policy code, compare all the differences, constrain and validate the P3P trees, and finally assemble the aggregate policy in a pretty-print XML format. Being a prototype, the current implementation is not fully automated in terms of determining the policy URI from the P3P reference file. The tool is in its alpha stage.

The current parser API essentially allows us to model XML in a tree format or in a pretty print XML format. A side feature of the aggregator tool is the additional comment function that explains each merge step in the aggregate policy; this allows the entities to trace the merging steps and manually verify them if the need arises. We include comments where aggregate actions affect the policy. The comment element in this implementation appends a brief summary of how an aggregate element was obtained.

[32] http://www.eclipse.org/downloads/packages/eclipse-ide-java-developers/junor

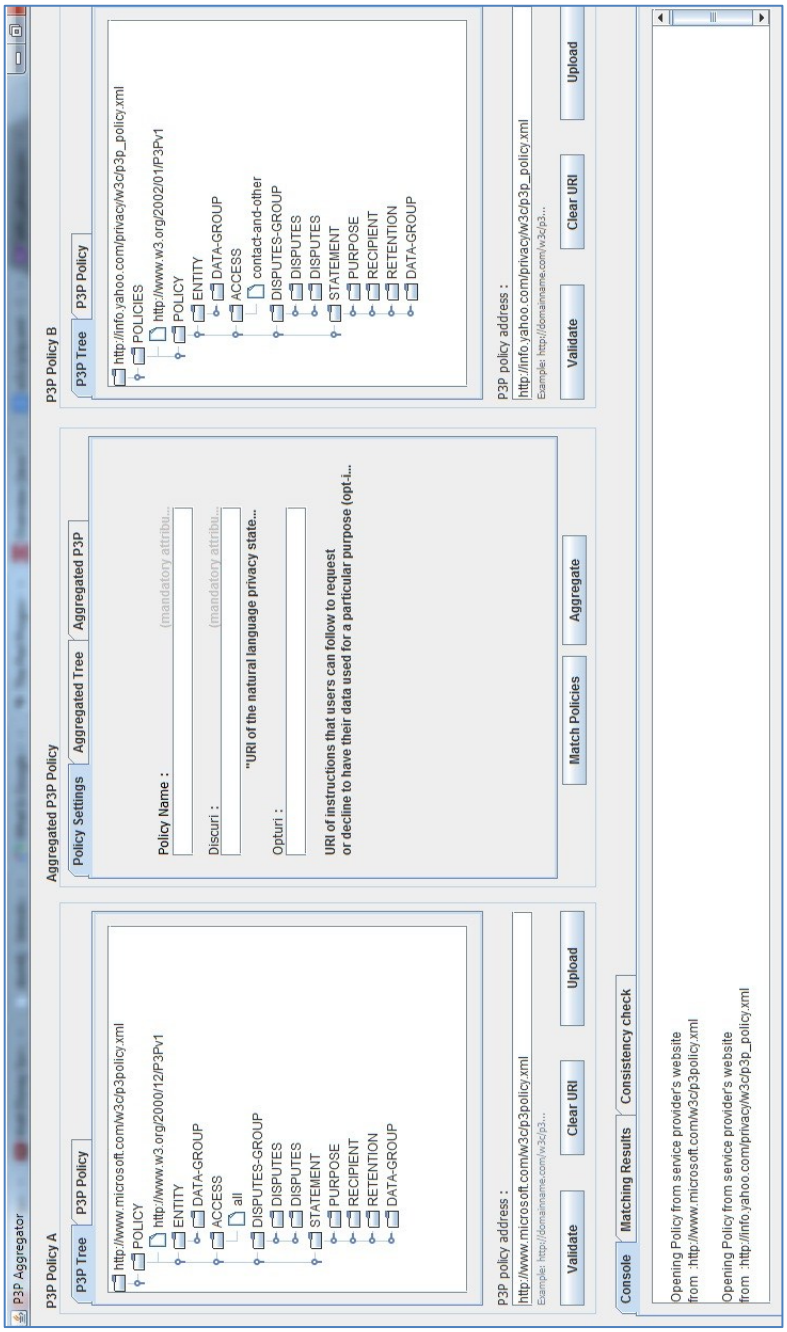


Figure 49: Design Overview of Policy Merger in an Aggregation Platform

7.3 Test Environment

For each match and merge scenario, three matches, and one merge operation are performed consecutively:

- The text matches
- The label matches
- The relative matches
- The full matches
- The three-way merge principle

The framework for the evaluation is a Java program which already performs the basic three-way merge rules at the directory level with valid P3P policies as input arguments, as shown in section 4.3.3.2,. This decision was made in order to allow a more direct and consistent aggregation between the policies. The framework program validates the P3P policy after completing the upload process, and analyzes the P3P elements before matching the P3P STATEMENT elements.

All tests were carried out on an Intel Core i5 Pentium with 4 cores at 3.4GHz with 6GB of RAM. The machine runs Ubuntu Linux 12.04 and uses an up-to-date version of the eclipse Juno 32-Bit, DOM4J version 1.16 and Java SE 7.02.

7.4 Program Structure of the Merger Tool

In this section, we give a brief overview of the program structure. The main class is the *mainEngine*; the class contains the Java main method that controls the application's execution. The *frameMerger* class contains the method that generates the interface and parses the P3P policies to build the P3P tree. The interface from this method contains buttons that execute the P3P matches and invoke the P3P validator and the P3P merge methods.

The P3PMatch algorithm is implemented in the *matchPolicy* class; this implementation performs the text, relative, label and full matches (*matchEntity, matchAccess, matchDisputes,* and *matchStatement*), see section 5.4.

The *validatePolicy* class invokes the P3P constraints and validations algorithms. The class encapsulates the the *checkConstraint* and the *getValidate* methods.

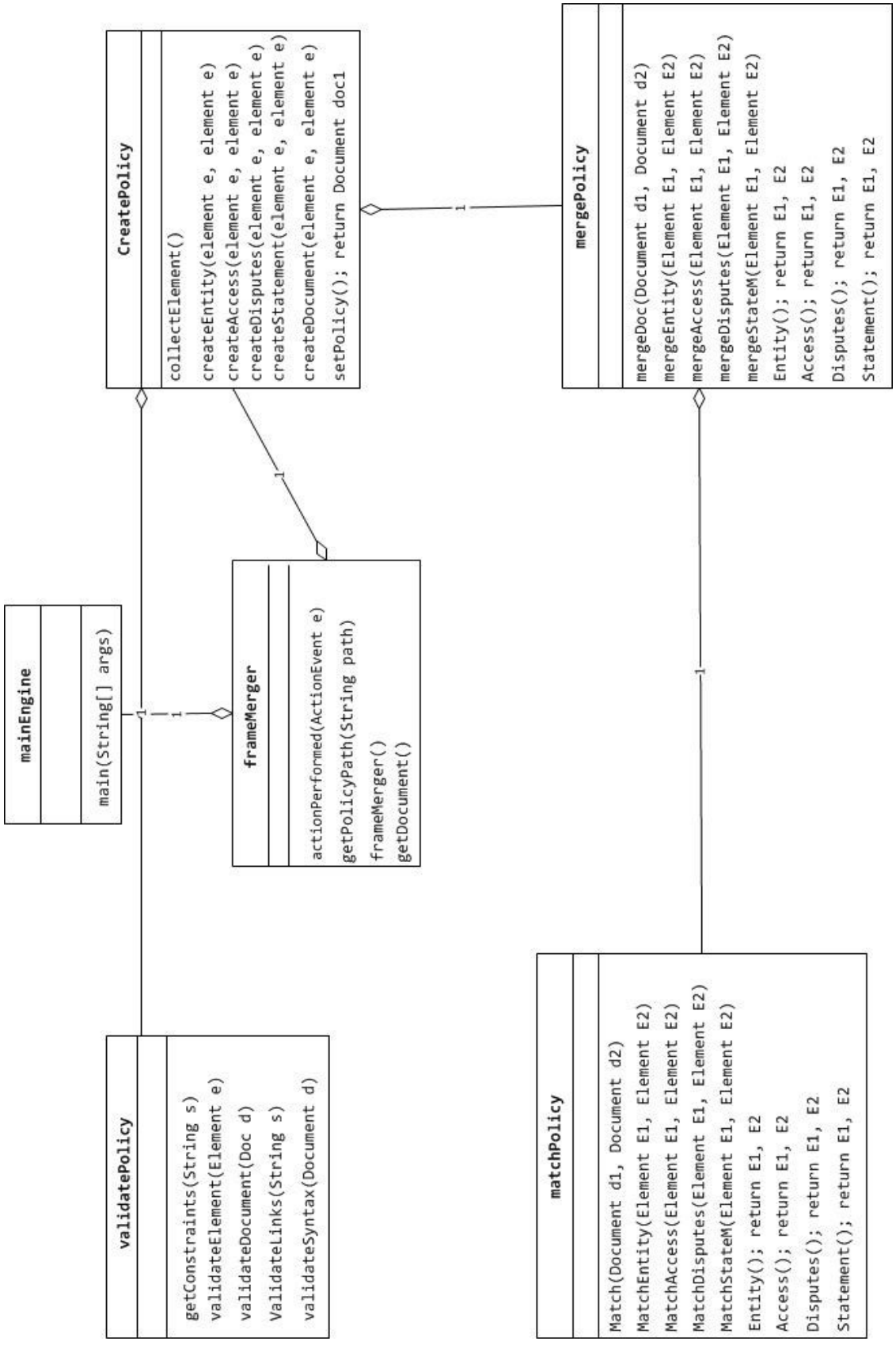


Figure 50: Class Diagram showing composition relationships

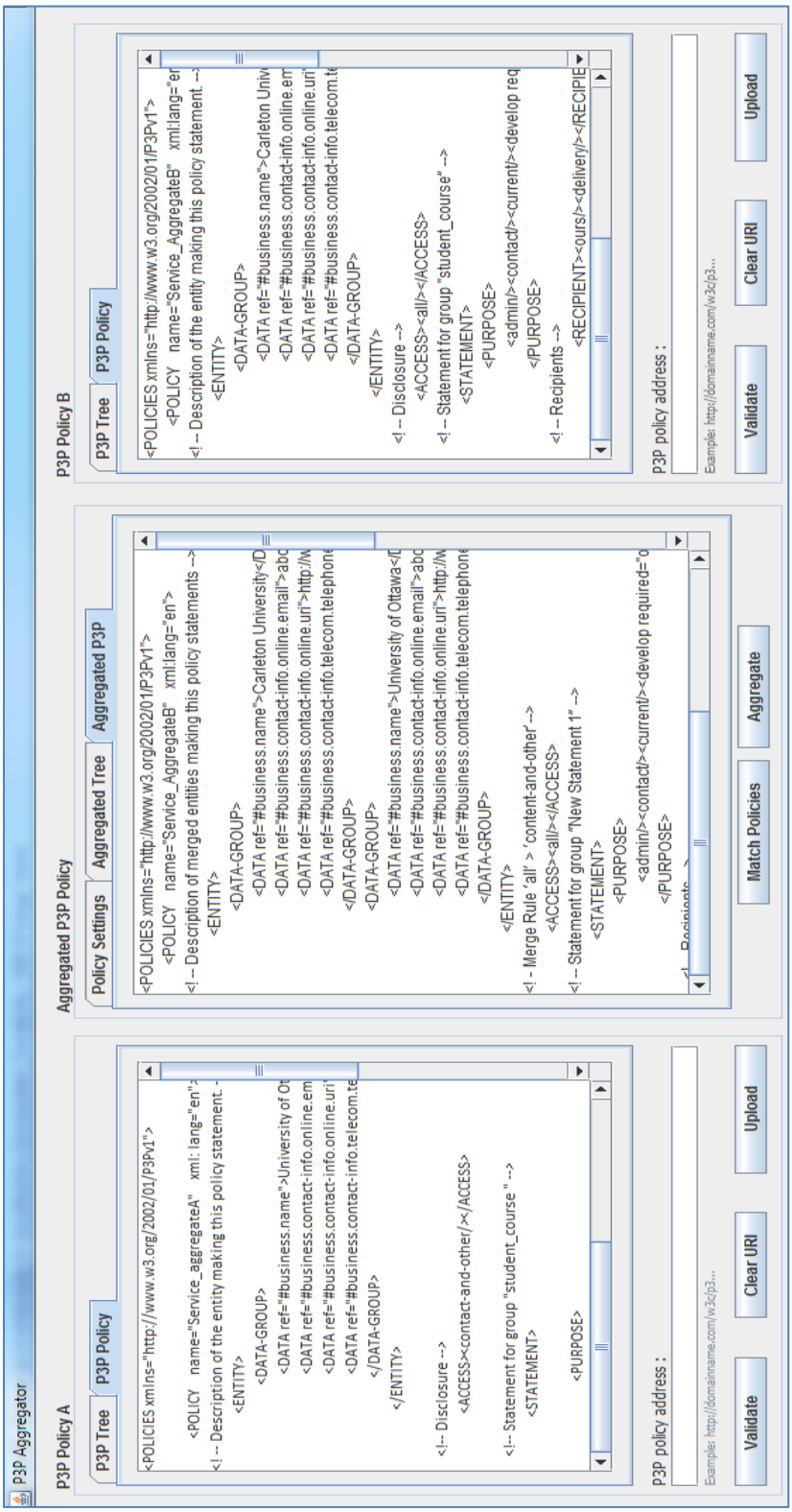


Figure 51: Example of an Aggregate P3P policy

7.5 Differences Between Our Approach and the Approach of Dong et al.

Compared with Dong et al. [30], our merger concept has more advanced functions in the matching and merging algorithms, including validating P3P policies, checking for inconsistencies, using the three-way merging principle and matching P3P elements. The Validator function helps prevent syntactic errors and helps in producing a consistent P3P policy. We consider P3P policy consistency to be a more important factor in this work than other factors (e.g. Merging speed). The differences between the two aggregate policy approaches are outlined in the following subsection.

Dong et al.'s aggregate algorithm for merging P3P policies will not consistently aggregate policies in a composite environment or aggregation platform because the approach used is based only on the two-way merge principle. This merger principle only allows a syntactic merge. Therefore, this algorithm needs to completely compare P3P elements in both policies and the policy aggregator will have to resolve semantic inconsistencies manually each time a merge operation is performed. The aggregate policy approach we are considering in this thesis is in real-time, which reduces manual manipulations from the policy aggregator.

	Dong et al	**Our P3P Merger Approach**
Validating P3P Vocabulary	No Syntax validation	Adopted the W3C's P3P validator
Matching P3P Elements	Label Match	Text, Label, Relative, and Full Match
Relating P3P Data, Purpose, and Recipient	×	The P3P Data Key
Merge Principle	Two-way Merge principle	Three-way Merge
Constraints Or Merge Rules	×	✓

Table 11: Differences in P3P Merger Approach

7.6 Runtime

One of the main benefits of our merger approach is the consistency of the aggregate P3P policies. The other noticeable benefit is the speed in real time merges operation: because the P3P policy file has an average file size of 3 to 10 kilobyte (KB), a merge on two policy files can be executed in a very short time. When the merger program executes repeatedly, its efficiency becomes important. However, because of the huge processor power delivered in the cloud, our algorithm performs efficiently on typical P3P policy files. However our focus was on creating consistent aggregate policies rather than optimizing merge speed.

7.7 Example Scenario

We illustrate our approach by aggregating privacy policies of sample service providers. We will be considering two different service providers' privacy policies (Policy A and B), which are to be merged on an aggregation platform. Each privacy policy presented has different access granting rules for their users, where policies are specified using the P3P policy language.

Privacy Policy for Service A.

We present the privacy policy for service A first in natural language and then translated into a full P3P privacy policy.

Natural Language Policy:

*At **Service A** Inc, we care about the privacy policy of our customers. When browsing through our site, we will collect personal information on the family name, the phone number, and contact address. This information includes the number of times a web page is accessed, the browser used, the paths taken when moving through the web site. We will delete this information yearly.*

***Service A** uses cookies to store some necessary information about you on your computer. You will be able to access information that concerns your contact and other related data alone. Information collected will only be shared with delivery agents that share our privacy practices.*

P3P Policy for A:

```
<POLICY    name="Service_aggregateA"    xml:  lang="en">
          <ENTITY>
               <DATA-GROUP>
                 <DATA ref="#business.name">Service A, Inc</DATA>
                 <DATA ref="#business.contact-info.online.email">abcd@serviceA.ca</DATA>
                 <DATA ref="#business.contact-info.online.uri">http://www. serviceA.ca</DATA>
                 <DATA ref="#business.contact-info.telecom.telephone.number">123-456-8976</DATA>
                </DATA-GROUP>
           </ENTITY>
           <ACCESS><contact-and-other/></ACCESS>

          <STATEMENT>
                <PURPOSE>
                    <admin/><contact/><current/><develop required="opt-out"/><individual-analysis/>
                    <individual-decision/><telemarketing/>
               </PURPOSE>
               <RECIPIENT><ours/><delivery/></RECIPIENT>
              <RETENTION><legal-requirement/></RETENTION>
              <DATA-GROUP>
                          <DATA ref="#user.name.family",  DP = “admin, contact, develop, Individual-analysis,
individual decision”, DR = “ours, delivery”/>
                          <DATA ref="#user.info.phonenumber", DP = “current, admin, contact, develop,
Individual-analysis,  individual decision, telemarketing”, DR = “ours”/>
                           <DATA ref="#user.info.contact", DP = “current, admin, contact,”, DR = “ours”/>
               </DATA-GROUP>
          </STATEMENT>
        </POLICY>
```

Table 12: Privacy Policy for Service A

Privacy Policy for Service B.

We present the privacy policy for service B first in natural language and then translated into a full P3P privacy policy.

Natural Language Policy:

*At **Service B** Inc, we care about your privacy. When you browse through our site, we collect information on the efficiency and working of our website. This information includes the number of times a web page is accessed, the browser used, and paths taken when moving through the website. We purge this information from the public domain yearly.*

We also collect postal codes but will always prompt you to enter it with your permission, this information is aggregated with information collected from all visitors to our website for market analysis. This information is provided to third parties. Once prompted for your postal code, you will not be prompted again if your browser privacy preferences allow cookies.

Service A uses cookies to store whether you have entered your zip code or declined to do so. We access this information each time you visit our site so that you are only prompted the first time you visit our web site.

P3P Policy for B:

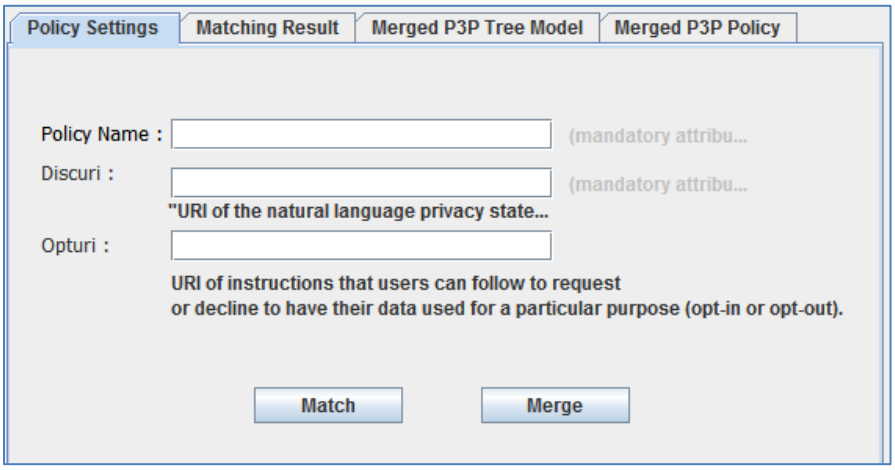
```
<POLICY    name="Service_AggregateB"    xml:lang="en">
          <ENTITY>
               <DATA-GROUP>
                 <DATA ref="#business.name">ServiceB</DATA>
                 <DATA ref="#business.contact-info.online.email">abcd@ ServiceB.ca</DATA>
                 <DATA ref="#business.contact-info.online.uri">http://www. ServiceB.ca</DATA>
                 <DATA ref="#business.contact-info.telecom.telephone.number">123-456-0000</DATA>
                </DATA-GROUP>
           </ENTITY>
          <ACCESS><all/></ACCESS>
          <STATEMENT>
             <PURPOSE>
                   <admin/><contact/><current/><develop required="opt-out"/><tailoring/><telemarketing/>
              </PURPOSE>
             <RECIPIENT><ours/><delivery/></RECIPIENT>
             <RETENTION><legal-requirement/></RETENTION>
              <DATA-GROUP>
                        <DATA ref="#user.name.family"  , DP = “admin, contact, develop, tailoring”, DR =
“ours, delivery”/>
                        <DATA ref="#user.info.phonenumber" , DP = “admin, develop, telemarketing”, DR =
“ours, delivery”/>
                        <DATA ref="#user.info.contact" , DP = “admin, contact, develop, tailoring”, DR =
“ours, delivery” />
                        <DATA ref="#user.gender", DP = “current, DR = “ours” />
              </DATA-GROUP>
            </STATEMENT>
          <STATEMENT>
               <PURPOSE>
                   <admin/><contact/><current/><develop required="opt-out"/><tailoring/><telemarketing/>
               </PURPOSE>
                     <RECIPIENT><ours/></RECIPIENT>
                      <RETENTION><legal-requirement/></RETENTION>
                   <DATA-GROUP>
                        <DATA ref="#user.postal_no", DP = “current, admin, contact, develop, tailoring”, DR
= “ours, delivery “/>
                        <DATA ref="#user.user_number" DP = “current, admin, develop, tailoring”, DR =
“ours “/>
                        <DATA ref="#user.browserinformation" DP = “current, admin, develop, tailoring”,
DR = “ours“/>
                    </DATA-GROUP>
            </STATEMENT>
        </POLICY>
</POLICIES>
```

Table 13: Privacy Policy B

7.7.1 Aggregate Result

Stage1 **Policy Definition**

The service aggregator is always responsible for creating a new Policy name, Dispute URI (discuri), and Opt-out or Opt-in URI information.



Stage2 **Entity Element**

After the validation process, the tool gathers the service providers' information and compares if they are the same elements with text, label or attributes. The match process detects similar elements to be merged.

```
<! -- Description of merged entities making this policy statements -->
        <ENTITY>
            <DATA-GROUP>
              <DATA ref="#business.name">Service A, Inc</DATA>
              <DATA ref="#business.contact-info.online.email">abcd@ServiceA.ca</DATA>
              <DATA ref="#business.contact-info.online.uri">http://www.serviceA.ca</DATA>
              <DATA ref="#business.contact-info.telecom.telephone.number">123-456-0000</DATA>
            </DATA-GROUP>

            <DATA-GROUP>
              <DATA ref="#business.name">ServiceB</DATA>
              <DATA ref="#business.contact-info.online.email">abcd@serviceB.ca</DATA>
              <DATA ref="#business.contact-info.online.uri">http://www. serviceB.ca</DATA>
              <DATA ref="#business.contact-info.telecom.telephone.number">123-456-8976</DATA>
             </DATA-GROUP>
        </ENTITY>
```

Stage3 **Access Element**

Access type for Service A = Contact-and-Other e.g. <ACCESS><contact-and-other/></ACCESS>

Access type for Service B = All e.g. <ACCESS><all/></ACCESS>

Conflict = Access Type. Apply AccessRule( ), aggregate result = <All/>

<! – Merge Rule 'all' > 'content-and-other'-->
<ACCESS><all/></ACCESS>

Aggregate Result for A and B

(Note that the execution time for this example was negligible). <POLICY name="Service_AggregateB" xml:lang="en">

```
<! -- Description of merged entities making this policy statements -->
        <ENTITY>
            <DATA-GROUP>
              <DATA ref="#business.name">Service A, Inc</DATA>
              <DATA ref="#business.contact-info.online.email">abcd@ServiceA.ca</DATA>
              <DATA ref="#business.contact-info.online.uri">http://www.serviceA.ca</DATA>
              <DATA ref="#business.contact-info.telecom.telephone.number">123-456-0000</DATA>
            </DATA-GROUP>

            <DATA-GROUP>
              <DATA ref="#business.name">ServiceB</DATA>
              <DATA ref="#business.contact-info.online.email">abcd@serviceB.ca</DATA>
              <DATA ref="#business.contact-info.online.uri">http://www. serviceB.ca</DATA>
              <DATA ref="#business.contact-info.telecom.telephone.number">123-456-8976</DATA>
             </DATA-GROUP>
        </ENTITY>
<! – Merge Rule 'all' > 'content-and-other'-->
        <ACCESS><all/></ACCESS>
<! -- Statement for group "New Statement 1" -->
        <STATEMENT>
            <PURPOSE>
              <admin/><contact/><current/><develop required="opt-out"/><tailoring/><telemarketing/>
            </PURPOSE>
  <! -- Recipients -->
              <RECIPIENT><ours/><delivery/></RECIPIENT>
  <! -- Retention -->
               <RETENTION><legal-requirement/></RETENTION>
  <! -- Base dataschema elements. -->
               <DATA-GROUP>
              <DATA ref="#user.name.family"  , DP = "admin, contact, develop, tailoring", DR = "ours,
delivery"/>
               <DATA ref="#user.info.phonenumber" , DP = "admin, develop, telemarketing", DR =
"ours, delivery"/>
               <DATA ref="#user.info.contact" , DP = "admin, contact, develop, tailoring", DR = "ours,
delivery" />
               <DATA ref="#user.gender", DP = "current, DR = "ours" />
  <! – Merge operation added gender to the statement -->
                  <DATA ref="#user.gender"/>
              </DATA-GROUP>
```

```
            </STATEMENT>
  <! -- Statement for group "library" -->
            <STATEMENT>
                <PURPOSE>
                    <admin/><contact/><current/><develop required="opt-out"/><tailoring/><telemarketing/>
                </PURPOSE>
  <! -- Recipients -->
                    <RECIPIENT><ours/></RECIPIENT>
                     <RETENTION><legal-requirement/></RETENTION>
  <! -- Base dataschema elements. -->
                     <DATA-GROUP>
                <DATA ref="#user.postal_no", DP = “current, admin, contact, develop, tailoring”, DR = “ours,
delivery “/>
                    <DATA ref="#user.user_number" DP = “current, admin, develop, tailoring”, DR = “ours “/>
                   <DATA ref="#user.browserinformation" DP = “current, admin, develop, tailoring”, DR =
“ours“/>
                        </DATA-GROUP>
              </STATEMENT>
  <! -- End of policy -->
          </POLICY>
```

Table 14: Aggregated Result for P3P A and B

7.8 Conclusion

In this chapter, we have presented the design choices for implementing the different parts and methods of our P3P policy aggregator tool. The overview of the prototype describes the features that are satisfied after the implementation. We presented an overview of the technologies used; Java Programming language, Dom4J, and XML. We discussed how the classes interact by presenting the UML of our implementation. We introduced the P3P merge tool, the graphical user interface, the program structure, and an example scenario of how the aggregate results are formed.

In the next chapter, we will conclude the thesis by reviewing the main contributions, the approach involved in our solution, the algorithms introduced, and the future work of the thesis.

Chapter 8

Conclusions and Future Work

8.1 Conclusion

In current privacy practices, privacy statements are often informal and imprecise, making it necessary to resort to human intervention in order to detect and resolve issues. The P3P standard has been developed to promote the automation of privacy policies. After giving a summary of P3P's main characteristics and solutions, we have given examples showing that the proper use of P3P and its enhancements can help the deployment and execution of privacy-sensitive services. The major objective of the thesis was to come up with a formal semantic approach and consistent rules to build a merger system on an aggregation platform, which is a potentially useful environment where the policy merger system can be used.

In this thesis, we have considered the use of the Platform for Privacy Preferences (P3P) language to encode privacy policies for services in a cloud-computing environment. This has led to several research contributions. We have implemented the concept of a three-way merge principle of two P3P policies. The several match cases considered in the thesis were identified in Chapter 6 and each of them was tested, to check the algorithm's functionality. For enhancing the P3P semantics, we have extended the syntax rules of the P3P language to include a relational link that ties together the purpose, the recipient, and the data elements in the P3P policy. We have demonstrated that the relation between the purpose element and the data element in a statement will eliminate some known inconsistencies in the P3P language. Lastly, we have shown how to formally verify and validate the merged P3P policy, by modeling and checking policies using an Alloy model checker.

8.2 Future Work

This section of the thesis provides a summary of the proposed directions suggested in [61]. In the future, we intend to consider some web privacy areas including a P3P policy solution for anti-phishing attacks, implementing P3P user agents for mobile applications, and locating service providers' local P3P policies as suggested by [61]. These concepts are explained below.

8.2.1 Policies and User agents for Mobile Apps

Smart phones provide mobility and flexibility to users; however, privacy declaration methods do not exist for Smartphone applications. Usually we trust our smart phones more than our laptops because we carry them with us most of the time. As a smart phone becomes a part of our everyday life, many opportunities for privacy leaks may arise, like giving away important and personal information whenever users connect to the internet or download new applications. Smartphone applications or mobile website visits often can track user's interactions or movements. Thus, some mobile users will want to know how the applications or mobile websites will use the information available to them, e.g., will they install tailoring cookies or tracking bugs on their smart phones? Will they allow third parties to install new applications? The third parties may not operate in the same way as the web site or application installed; they may have entirely different privacy policies, and the fact that they have users' personal information may increase the chances of privacy compromises.

For example, users can use smart phones to assess or store their health information (web connection to the clinic) or financial information (mobile banking applications) for easy and fast connection in cases of emergency. The challenge here will be that users who are conscious of the information stored on or accessed via their phones may not want those applications or websites to take or misuse the information. This leads to the following research questions:

- How to create a mobile-based privacy user agent that can communicate compact privacy policies of mobile web sites or applications to users.

- How to delegate automatic access control privileges to mobile applications and websites based on user defined privacy preferences.

8.2.2 Anti-Phishing Mechanism with the P3P Protocol

Fraudsters can create fake websites to lure users for collecting their data. This type of attack on users is called Phishing. Personal identity information such as username, passwords, and credit card details can be stolen by Phishing attacks from unsuspecting users by entities masquerading as trusted entities, such as PayPal sites. The main aim of phishers is to lead users to unsafe websites, and retrieve their personal data. Since the P3P policy on a legitimate website is freely available to the public, phishers can easily copy the P3P policies of the trusted websites and republish them on their fake websites. This can aid phishing attack schemes because P3P user agents that are trusted by the users will be misled by the P3P policy that appears to be associated with the fake site.

The P3P user agent will report a legitimate policy summary, perhaps with a (stolen) trust seal from a third party privacy seal organization. We suggest here a solution that is based on cryptographic techniques: the legitimate site should digitally sign its P3P policy using the private key that corresponds to the public key contained in a valid certificate for the site (possibly, but not necessarily, its SSL certificate). This signature can be done using the XML Signature format [13]. A P3P user agent, instead of simply downloading a P3P policy from a target site, would also engage in a challenge-response protocol with the site, submitting a one-time challenge (i.e., a random number that has not been used before) and receiving in return a digital signature on that challenge. If the user agent is able to verify this signature with the same public key that verifies the signature on the P3P policy, and if this public key is the one contained in the verified public key certificate issued to this site from a trusted Certification Authority, then the user can be certain that the downloaded P3P policy is actually associated with the website currently being accessed. A fake website would thus be unable to steal a P3P policy from a legitimate site to increase the probability of phishing success because the fake site does not know the real site's private key and so would be unable to construct a proper response for the user agent's one-time challenge.

8.2.3 Geographically Locating P3P Policies

Multinational companies are taking advantage of the web's capabilities (file sharing, web applications, and fast communication); they now globally distribute their products on the web for customers in different countries. However, policy guidelines for multinational companies are usually determined by the governments of the countries where the regional websites are operating. These government guidelines may affect the posted websites' privacy policies. More precisely, a branch of a multinational company in a particular country will only be bound by government guidelines or policies in that country. Thus, there is a need for Geolocating web users to know what policies should apply to their present location on the internet. The mechanism should allow multinational websites to accurately segment their global audiences and help the users to determine where (in which country) their data will be stored or if the government of the country where the website is located will have access to their personal data. For example, Amazon is a well-known multinational website located in countries like the USA (*Amazon.com*) and Canada (*Amazon.ca*). The two websites may have two different privacy policies adapted to each government's laws and guidelines. In a scenario where a user wants to purchase a product on Amazon.ca, it may happen that because of product availability, the user gets redirected to Amazon.com. These policy differences can promote research topics and questions such as:

- How will the user know the web location where his/her personal information will be stored? Will the other websites share their personal information?
- What are the inconsistencies between the various policies involved, and what are the risks associated with them?
- Most multinational companies do not bother to change their P3P policies, even if the privacy policies are modified according to the government guidelines. (E.g. Amazon.com/w3c/p3p.xml is available but Amazon.ca/w3c/p3p.xml is not). So redirected users may not get new summaries from their P3P user agents.

If a multinational company publishes P3P policies in a single domain, there should be a tool to determine the P3P file that will apply to the user based on its location.

References

[1] C Adams. *A Classification for Privacy Techniques*, in University of Ottawa Law & Technology Journal,Ottawa, Canada: University of Ottawa, 2006, volume 3, number 1, pp. 35-52.

[2] C Adams and K Barbreri. *Privacy Enforcement in E-Services Environments*, in Privacy Protection for E-Services. Hershey, USA: Idea Group Publishing, 2006, pp. 172-202.

[3] A Aggarwal, M Bawa, P Ganesan, H Garcia-Molina, K Kenthapadi, N Mishra, R Motwani, U Srivastava, D Thomas, J Widom, and Y Xu. *Vision paper: enabling privacy for the paranoids*, in In VLDB '04: Proceedings of the Thirtieth international conference on Very large data bases.: VLDB Endowment, 2004, pp. 708-719.

[4] R Agrawal, J Kiernan, R Srikant, and Y Xu. *An XPath-based Preference Language for P3P*, in Proceedings of the 12th international conference on World Wide Web. Budapest, Hungary: ACM, 2003, pp. 629-639.

[5] R Agrawal, J Kiernan, R Srikant, and Y Xu. *Hippocratic Databases*, in In 28th Int'l Conference on Very Large Databases. Hong Kong, China: VLDB Endowment, 2002, pp. 143-154.

[6] V K Anne, A Hors, P Hegaret, L Wood, G Nicol, J Robie, M Champion, S Byrne, K Anne, and G Arhey. *Document Object Model (DOM) Level 3 Core Specification*, Technical Report TR REC-DOM-3-Core-20040407, The World Wide Web Consortium (W3C), USA: 2004.

[7] A Antón, E Bertino, N Li, and T Yu. *A roadmap for comprehensive online privacy policy management*, in Communications of ACM, New York, USA, ACM: 2007, volume 50, issue 7, pp. 109-116.

[8] M Arcand, J Nantel, M Arles-Dufour, and A Vincent. *The Impact of reading a web site's Privacy statement on perceived control privacy and perceived trust*, in an Online Information Review, Canada: Emerald Group Publishing Limited, volume 31, number 5, 2007, pp. 661-681.

[9] J Argyrakis, S Gritzalis, and C Kioulafas. *Privacy Enhancing Technologies (PET): A Review*, in Electronic Government (Lecture Notes in Computer Science), Heidelberg, Berlin: Springer, 2003, volume 2739, pp.1079.

[10] F Arshad. *Privacy fox - A JavaScript Based P3P Agent for Mozilla Firefox*, School of Computer Science, Carnegie Mellon University, Pittsburgh,USA: [Online] http://privacyfox.mozdev.org/PaperFinal.pdf, 2004.

[11] P Ashley, S Hada, G Karjoth, and M Schunter. *E-P3P Privacy Policies and Privacy Authorization*, in Proceedings of the 2002 ACM workshop on privacy in the Electronic Society. Washington, USA: ACM, 2002, pp. 103-109.

[12] L Badger, D Bernstein, R Bohn, F Vaulx, M Hogan, J Mao, J Messina, Kevin, Mills, A Sokol, J Tong, F Whiteside, and D Leaf. *US Government Cloud Computing Technology Roadmap*, in NIST Cloud Computing Program Information Technology Laboratory. USA: NIST Cloud Computing Program Information Technology Laboratory, 2011, volume 1, pp. 1-32.

[13] M Bartel, and J Nantel. M Arles-Dufour, and A Vincent, *XML Signature Syntax and Processing (Second Edition)*, in *the World Wide Web Consortium, REC-xmldsig-core-20080610*, 2008.

[14] M Bennicke and Langendorfer, *Towards automatic negotiation of privacy contracts for Internet services*, in The 11th IEEE International Conference on Networks ICON2003. Frankfurt, Germany: IEEE, 2003, pp. 319-324.

[15] W Bernhand. *Structure-Oriented Merging of revisions of Software Documents*, in Proceedings of the 3rd international workshop on Software configuration management, SCm'91. New York, USA: ACM, 1991, pp. 68-79.

[16] D Binning. *Top five cloud computing security issues,* in Computer Weekly. United Kingdom: Computer Weekly, April 2009. [Online]. http://www.computerweekly.com/news/2240089111/ Top-five-cloud-computing-security-issues"

[17] C Bournez and C A Ardagna. *Policy Requirements and State of the Art*, in Privacy and Identity Management for Life. Berlin, Germany: Springer, 2011, pp. 295-312.

[18] J Buffenbarger. *Syntactic Software Merging*, in Selected papers from the ICSE SCM-4 and SCM-5 Workshops, on Software Configuration Management. London, England: Springer-Verlag, 1995, pp. 153-172.

[19] S Byers, L F Cranor, and D Kormann. *Automated analysis of P3P-enabled Web sites*, in Proceedings of the 5th international conference on Electronic commerce. Pittsburgh, Pennsylvania, USA: ACM, 2003, pp. 326 - 338.

[20] W Y Chang, H Abu-Amara, and J F Sanford. *Introduction to Enterprise Services and Cloud Resources*, in Transforming Enterprise Cloud Services. The Netherlands: Spinger, ISBN 978-90-481-9845-0, 2011, pp. 1- 42.

[21] L Cranor. *P3P: Making Privacy Policies more useful*, IEEE Security and Privacy Journal, IEEE,

2003, volume 1, issue 6, pp. 50-55.

[22] L Cranor, M Arjula, and P Guduru. *Use of a P3P User Agent by early adopters*, in Proceedings of 9th ACM Workshop on Privacy in the Electronic Society, Washington, DC, 2002.

[23] L Cranor, S Byers, and D Kormann. *An Analysis of P3P Deployment on Commercial, Government, and Children's Web Sites as of May 2003*, Technical Report prepared for the May 2003 Federal Trade commission workshop on technologies for Protecting Personal Information, USA, 2003.

[24] L Cranor; M Langheinrich, M Marchiori, M Presler, and J Reagle. *The Platform for Privacy Preferences 1.1(P3P1.1) specification*, The World Wide Web Consortium (W3C) Note NOTE-P3P11-20061113 [Online],http://www.w3.org/TR/P3P11/, 2006.

[25] L Cranor, S Egelman, S.Sheng, A.McDonald, and A.Chowdhury. *P3P Deployment on websites*, in Electron. Commercial. Rec. Application, Amsterdam, The Netherlands: Elsevier Science Publishers B. V, 2008, pp 274-293.

[26] L Cranor, M Langheinrich, M Marchiori, M Presler-Marshall, and J Reagle. *A P3P Preference Exchange Language 1.0(APPEL1.0) specification*, The World Wide Web Consortium (W3C) [Online]. http://www.w3.org/TR/P3P-preferences/, 2000.

[27] L Cranor and J Reidenberg, *Can User agents Accurately Represent Privacy Notices ?*, in the Telecommunications Policy Research Conference, TPRC2002, Alexandria,Virginia, USA: MIT Press, 2002, pp. 1-22.

[28] N Deepa and R Sahiyaseeian. *The Cloud and the Changing Shape of Education - Eaas (Education as a Service)*, in International Journal of Computer Application. Foundation of Computer Science, New York: IJCA Journal, 2012, volume 42, issue 5, pp. 04 - 08.

[29] T Dillon, C Wu, and Chang.E. *Cloud computing issues and challenges*, in 24th IEEE International Conference on Advanced Information Networking and Applications (AINA). Perth, Australia: IEEE, 2010, pp. 27-33.

[30] L Dong, Y Mu, W Susilo, P wang, and J Yan. *A Privacy Policy Framework for service Aggregation with P3P*, in The Sixth International Conference on Internet and Web Applications and Services ICIW 2011, St. Maarten, The Nertherlands: Think Mind Journals, 2011, pp. 171-177.

[31] X Fu, *Conformance verification of privacy policies*, in Proceedings of the 7th international

conference on Web services and formal methods, WS-FM'10. Hoboken, NJ, USA: Springer-Verlag, 2011, pp. 86-100.

[32] F Gens. *IT Cloud Service Forecast – 2008- 2012,: A Key Driver of New Growth*. IDC User Survery. [Online]. HYPERLINK "http://blogs.idc.com/ie/?p=224", 2008.

[33] J Glasgow, G Macewen, and P Panangaden. *A logic for reasoning about security*, in ACM Transaction. Computer Systems. New York, USA: ACM, 2003, vol. 10, pp. 226 - 264.

[34] H Graham and B Tevfik. *Automated Verification of Access Control Policies*, Techinical Report TR 2004-22, Department of Computer Science, University of California, Santa Barbara, USA: 2004.

[35] H Graham and B Tevfik. *Automated verification of access control policies using a SAT solver*, in International Journal on Software Tools for Technology Transfer (STTT), Special issue on selected papers from workshop on Web Quality, Verification and Validation, Berlin, Heidelberg: Springer-Verlag, volume 10, number 6, 2008, pp. 473-534.

[36] J Halpern and V Weissman. *Using first-order logic to reason about policies*, in Proceedings of 16th IEEE Computer Security Foundations Workshop, Ithaca, New York, USA: IEEE Computer Society Press, 2003, pp. 187 - 201.

[37] D.D He and Y Jian. *Security Policy Specification and Integration in Business Collaboration*, in IEEE International Conference on Services Computing, SCC 2007. Salt Lake City, USA: IEEE, 2007, pp. 20-27.

[38] G Hogben. *Suggestions for Long term changes to P3P*, in W3C Workshop on Long term Future of P3P and Enterprise Privacy Language. Schlesswig-Kiel, Germany: W3C Technology and Society Domain, 2003.

[39] D Hu. *Privacy Enforcement Architecure for an E-Business Architecure*, Masters Thesis,University of Ottawa, Ottawa, Canada, 2010.

[40] D Jackson. *Automating first-order relational logic*, in Proceedings of the 8th ACM SIGSOFT international symposium on Foundations of software engineering: twenty-first century applications, SIGSOFT '00/FSE-8. San Diego, California, USA: ACM, 2000, pp. 130-139.

[41] B Jiang, D Zhang, C Yang, G Yue, and F Yu. *A Privacy Policy of P3P Based Relational Database*, in Proceedings of the First International Multi-Symposiums on Computer and Computational Sciences - IMSCCS'06, Washington, DC, USA: IEEE Computer Society, ISBN

0-7695-2581-4, 2006, pp 510-515.

[42] G Karjoth, M Schunter, E V Herreweghen, and M Waidner. *Amending P3P for Clearer Privacy Promises*, in Proceedings of the 14th International Workshop on Database and Expert Systems Applications. Washington, DC, USA: IEEE Computer Society, 2003, pp. 445 - 450.

[43] A Khurat, D Gollhann, and J Abendorth. *A Formal P3P Semantics for Composite Services*, in Proceedings of the 7th VLDB conference on Secure data management, Springer-Verlag. Singapore: ACM, 2010, pp. 113 -131.

[44] A Kobsa. *Privacy Enhanced Personlaization*, in Communications of ACM Magazine, New York, USA: ACM, volume 50, number 8, 2007, pp. 24-33.

[45] J Kolter and G Pernul, *Generating user understandable Privacy Preferences*, in Proceedings of the 4th International Conference on Availability, Reliability and Security ARES'09, Fukuoka, Japan: IEEE Computer Society, 2009, pp. 299-306.

[46] P Kumaraguru, L F Cranor, J Lobo, and B Calo. *A survey of Privacy policy Languages*, in Workshop on Usable Privacy and Security (USM '07). Pittsburgh, USA: IEEE, 2007.

[47] Cylab Usable Privacy Security Lab. Carnegie Mellon University, Computer Science Department. [Online]. http://www.Privacyfinder.org, 2012.

[48] T Lindholm. *A Three-Way Merge for XML documents*, in Proceedings of the 2004 ACM symposium on Document engineering DocEng'04, Milwaukee, Wisconsin, USA: ACM, 2004, pp.1-10.

[49] J Liu, Y Huang, D Li, F Wu, and B Li. *A web-based aggregated platform for user-contributed interactive media broadcasting*, in Proceedings of the 15th international conference on Multimedia, MULTIMEDIA '07. Augsburg, Germany: ACM, 2007, pp. 541-544.

[50] P Luuk. *Change Detection in XML trees a survey*, in 3rd Twente Student Conference on Information Technology, Faculty of Electrical Engineering, University of Twente, Enschede, The Nertherlands: 2004.

[51] D Maarten, L Bert, and J Wouter. *Federated Authorization for SaaS applications*, in Doctoral Symposium of ESSoS 12. Eindhoven, The Netherlands: CEUR, 2012, pp. 43-48.

[52] M Mahdi and L Logrippo. *Access control policies: Modeling and validation*, in Proceedings of the 5th NOTERE Conference. Gatineau, Canada: IEEE, 2005, pp. 85-91.

[53] C Mar, *Privacy Bird for chrome*, in Privacy Policy , Law, and Technology Project. Pittsburgh,

USA: Electrical and Computer Engineering, Carnegie Mellon University, 2010.

[54] A McDonald, R Reeder, P Kelly, and L Cranor. *A Comparative Study of Online policies and Formats*, in Proceedings of PETS '09 Proceeding of the 9th International Symposium on Privacy Enchaning Technologies, Seattle, USA: Springer-Verlag, 2009, pp 37-55.

[55] P Mell and T Grance. *The NIST Definition of Cloud Computing*, in Recommendations of the National Institute of Standard and Technology, National Institute of Standards and Technology (NIST), Department of Commerce, USA: volume 15, 2009.

[56] T Mens. *A State-of-the-Art Survey on Software Merging*, in IEEE Transaction, Software Engineering, Piscataway, New Jersey, USA: IEEE Press, 2002, Volume 28, Issue Number 5, pp. 449-462.

[57] B Michael, D Markus, and S Rainer. *An algebra for composing enterprise privacy policies,* in Computer Security – ESORICS, Lecture Notes in Computer Science. Berlin: Springer, 2004, pp. 33-52.

[58] G E Nigusse and B Decker. *Capabilities and Limitations of P3P*, in CW Reports Leuven, Department of Computer Science, K.U.Leuven: volume CW539, 2009.

[59] L Ninghui, T Yu , and A Anton. *A semantics-based approach to privacy languages*, Techincal Report TR 2003-28, CERIAS,2003, pp.1-22.

[60] J Nitin and L Bing. *A Generalized Tree Matching Algorithm Considering Nested Lists for Web Data Extraction*, in Proceedings of the Tenth SIAM International Conference on Data Mining, Columbus, Ohio: Society for Industrial and Applied Mathematics, 2010, pp 930-941.

[61] M Olurin, C Adams, and L Loggripo. *Platform for Privacy Preferences: Description, Current Status, and Future Directions*, in In Proceedings of tenth Annual Conference on Privacy, Security and Trust. PST'12. Institut MINES-TELECOM, Paris, France: IEEE, 2012, pp 217-220.

[62] N Papanikolaou, S Creese, and M Goldsmith. *Refinement checking for Privacy policies*, in Journal of Science Computer Program, Amsterdam, The Nertherlands: Elservier North-Holland, Inc, 2012, volume 77, number 10 – 11, pp. 1198-1209.

[63] A Paul, H Satoshi, K Gunter, P Calvin, and S Matthias. *Enterprise Privacy Authorization Language (EPAL 1.1), in EPAL 1.1 Specification*, International Business Machines (IBM) Corporation, USA: W3C, http://www.zurich.ibm.com/security/enterprise-privacy/epal/, 2003.

[64] B Peter, D Susan, and F Wenfei. *Keys for XML*, in Proceedings of the 10th international conference on World Wide Web. Hong Kong: ACM, 2001, pp. 201-210.

[65] K B Philipp. *UML Model Merging in the Context of a Model Evolution Engine*, in Master's Thesis, Institute of Computer Science, Research Group Quality Engineering, Innsbruck University, Innsbruck, Austria: 2010.

[66] P Resnick, and J Miller. *PICs: Internet Access Controls Without Censorship*, in Communications of the ACM. New York, USA: ACM, 1996, volume 39, pp. 87-93.

[67] C, Dos S Rodrigo, and H Carmem. *A Semantical Change Detection Algorithm for XML*, in Proceeding of Nineteenth International Conference on Software Engineering and Knowledge Engineering (SEKE'07). Boston, USA: ACM, 2007.

[68] J Rueda, S Gómez, and A Chimeno. *The Service Aggregator Use Case Scenario*, in Service Level Agreements for Cloud Computing. New York: Springer, 2011, pp. 329-342.

[69] A Salminen and F Tompa, *Why Use XML?*, in Communicating with XML , USA: Springer US, 2011, Print ISBN 978-1-4614-0992-2, pp. 69-91.

[70] F Satoh and T Tokuda. *Security Policy Composition for Composite Services*, in Proceedings of the 2008 Eighth International Conference on Web Engineering ICWE '08. Washington, DC, USA: IEEE Computer Society, 2008, pp. 86-97.

[71] M Schunter, P Ashley, S Hada, G Karjoth, and P Calvin. *Enterprise Privacy Authorization Language (EPAL 1.2)*, IBM Research Report RZ 3485 (#93951). Zurich: The World Wide Web Consortium (W3C), [Online] http://www.zurich.ibm.com/security/enterprise-privacy/epal/ Specification/, March 2003

[72] S Steve, B Uwe, O Kopp, L Frank, and U Tobias. *Cloud Data Patterns for Confidentiality*, in Proceedings of the 2nd International Conference on Cloud Computing and Service Science, CLOSER 2012. Porto, Portugal: SciTePress, 2012, pp. 387-394.

[73] B Suntisrivaraporn and A Khurat. *Formalizing and reasoning with P3P policies using a semantic web ontology*, in Proceedings of the 5th international conference on Multidisciplinary Trends in Artificial Intelligence. Berlin, Heidelberg: Springer-Verlag, 2011, pp. 87-99.

[74] R Thibadeau. *A Critique of P3P: Privacy on the web*, Carnegie Mellon University, Pittsburgh, USA: [Online]."http//dollar.ecom.cmu.edu/P3Pcritique/", 2000.

[75] M Toahchoodee and I Ray. *Validation of policy integration using alloy*, in Proceedings of the

Second international conference on Distributed Computing and Internet Technology, ICDCIT'05. Bhubaneswar, India: Springer-Verlag, 2005, pp. 420-431.

[76] Michael C Tschantz and J M Wing. *Formal Methods for Privacy*, in Proceedings of the 2nd World Congress on Formal Methods, FM '09. Eindhoven, Netherlands: Springer-Verlag, 2009, pp. 1-15.

[77] Y Wang, D.J DeWitt, and J Y Cai, *X-Diff: an effective change detection algorithm for XML documents*, in Proceedings of 19th International Conference on Data Engineering. Bangalore, India: IEEE, 2003, pp. 519-530.

[78] P Wang, Y Mu, W Susilo, and J Yan. *Privacy Perserving Protocol for Service Aggregation in Cloud Computing*, Journal of Software: Practice and Experience, New York, USA: John Wiley & Sons, Inc, 2012 volume 42, number 4, pp. 467 - 483.

[79] H, J Wang, and H, J Choi. *Security Protocol Design by Composition,* Technical Report TR-657:1476-2986, Cambridge University, UK: 2006.

[80] A F Westin. *Privacy and Freedom*, in Washington and Lee Law Review, USA: WLLR, 1968, volume 25, issue 1, article 20, p. 487.

[81] J Yang and Z Chen. *Cloud Computing Research and Security Issues*, in International Conference on Computational Intelligence and Software Engineering (CiSE). Wuhan, China: IEEE, 2010, pp. 1 -3.

[82] T Yu, N Li, and A Anton. *A formal semantics for P3P*, in Proceedings of the 2004 workshop on Secure web service. Fairfax, Virginia: ACM, 2004, pp. 1-8.