Analyzing Causes of Privacy Mismatches in Service Oriented Architecture

Master Thesis
September 2010
(within European Master in Informatics, EuMI)

Chair of IT-Security
RWTH Aachen University

School of Informatics
The University of Edinburgh

carried out at
European Microsoft Innovation Center, EMIC

by
Sharif Tanvir Rahman

Industry Supervisors:

Dr.-Ing. Laurent Bussard
Microsoft

Dr.-Ing. Ulrich Pinsdorf
Microsoft

University Supervisors:

Prof. Dr.-Ing. Ulrike Meyer
RWTH Aachen, Germany

Dr. Massimo Felici
The University of Edinburgh, UK

This thesis has been submitted in partial fulfillment of the requirements for the degree of Master of Science at the RWTH Aachen University and The University of Edinburgh. September, 2010.

Signature of author _____
Sharif Tanvir Rahman

Signature of university supervisor _____
Prof. Dr. -Ing. Ulrike Meyer

Signature of university supervisor _____
Dr. Massimo Felici

I hereby declare that, this submission is my own work and that, to the best of my knowledge and belief, it contains no material previously published or written by another person nor material which to a substantial extent has been accepted for the award of any other degree or diploma of a university or other institute of higher learning, except where due acknowledgment has been made in the text.

Signature of author: _____

Abstract

Internet users want controlled disclosure of their private data. They are concerned about what personal information they may reveal inadvertently while accessing websites. Intelligent systems can alleviate user's concern by assessing website's data practices automatically, assuming machine readable privacy policies. In case of mismatch with user expectations, these systems can also help both parties reviewing their privacy statements by providing useful information.

In the context of the collaborative research project PrimeLife (Privacy and Identity Management for Europe in Life), IBM, SAP, ULD, W3C and European Microsoft Innovation Center (EMIC) are working on new languages to define privacy policies. Specifying logic-based languages is important to enable reasoning on mismatches, i.e. understanding why service's privacy policy does not match user's privacy preferences.

This master thesis, done with EMIC, uses domain specific language to specify privacy and focuses on mechanisms to analyze mismatches and to propose modifications for getting a match, at a higher abstraction level, e.g. DSL. In case of mismatch, this guidance permits the user judging the required amendments and make the right choice thereby, i.e. reject service's policy or modify her preference accordingly. Another concern of this work is separating different aspects of a privacy management system and link them effectively as required.

The proposed approach is validated by developing a proof-of-concept prototype implementation with Microsoft's textual DSL tool, *MGrammar* and an existing formal language, *Formula*.

Contents

List of Figures

1

Introduction

1.1 Motivation

Privacy is a complicated yet valuable concept having various aspects [40]. In the context of using information and communication technologies, privacy issues may occur easily because of massive collection, combination, dissemination and permanent storage that these technologies facilitate [19], and because of users' possibilities for making or breaking the reputation of others. Individuals using these technologies leave digital traces, sometimes without knowing they do so and sometimes in full awareness, and thus leak personal information when they go online. For example, search engines register the keywords we use to find information, but also the links we click on and the paths we take while navigating in the online realm; Credit and debit card details are stored with each of our purchases.

1.1.1 Need for intelligent tools

Given the established fact that internet society is concerned with protecting personal information [19], it is surprising as well as worrying that "the average citizen or consumer" has "a serious lack of awareness" with respect to the technical possibilities for gathering, storing and processing her private data [42]. Moreover, the extent to which privacy of internet users is protected is poor and we witness *mistrusted relationship* between internet businesses and internet users [36]. One way of addressing the problem of building trustworthy relationship is designing intelligent tools that enables data practices of the service transparent to user. The key aspect is, user knows beforehand what she permits is really what is going to happen to her data.

In privacy management context, users define their *preferences* regarding the disclosure of their *personal data* whereas in the counterpart *policy* service specifies how and what data it processes [1]. If personal data are requested by a service provider, we

[1]For a definition of 'User Preference', 'Personal Data' and 'Service Policy', see Appendix A

need a *matching* system which may stay in either side and performs the comparison between both side's privacy document and inform the user in case of a mismatch.

Support for retrieving and analyzing privacy policy. However, determining whether the website's privacy policy is in compliance is cumbersome and disliked by users [20]. Users either accept the policy without reading or stays away from online being worried if her personal information is not protected while sharing. More powerful online privacy tools can retrieve both party's policies and analyze thereby. This way, personal information which is being revealed, when the user is going online, remains transparent to her and she is in complete control of where that information might end-up.

1.1.2 Tools for Whom?

Data owner wants the control. While vast amount of issues exist related to privacy and the surrounding management [40], the primary stakeholder to consider while analyzing privacy issues needs to be the *data owner*, i.e the user. Respecting a user-control in privacy management system means is that, to put simply, information that individuals want to keep private, or only wish to share with a limited area, should not spread further than they had originally intended. This implies keeping the control in customer's side so that they can make informed decisions about the release of personal data.

Concerns exist from organizations as well. Privacy concerns also exist from company's point of view. The growing privacy concerns of customers increases pressure to companies, so that their personal information is protected from both internal and external threats. Organizations if expose user's private data, either accidentally or maliciously, need to bear additional expenses. This requires a change in the way organizations deals of privacy. The new way includes placing well-understood and comprehensive sets of security and privacy policies, educate their staffs on these policies, enforce them, and then audit their enforcement to ensure compliance [14].

These processes are currently difficult for organizations to implement successfully and requires long-term goals. One starting point needs to be empowering organization's end-users with the support of working on the same domain they understand.

1.1.3 Tools need to intend end-users

Much of the existing technology (mentioned in section 3.3) is designed for use by experts. End users like customer or organizational users are not security experts and its difficult for them to use these tools correctly. This usability issue can make the situation further complicated as these mechanisms can be used incorrectly [43], which may lead to a worse scenario than not using them at all. This requirement of designing of easy-to-use systems for end-users is emphasized even in early literatures [4].

Need for using Natural Language. Important technical projects, such as P3P, XACML, EPAL, PPL, address privacy by providing a standard computer-readable format for privacy policies. But many consumers prefer privacy policies in

a standard, easy-to-read format. This contrast suggests supporting human-readable privacy policy, i.e. a policy written in a spoken language, for instance, English, that is intended for people, rather than computers, to read.

The organizational policy officers are experts in policy and legislation and need not to have technical background. Tools they use are simple documents, emails or spreadsheets. Neither a customer, i.e. internet user of organization's website, is likely to feel comfortable with complex presentations of privacy documents. Rather it seems both parties are most comfortable expressing their statements in a precise language that is close to natural language, e.g. like plain spoken English. This would enable them go through the privacy policies without requiring a technical background. This would also eliminate the understanding gap between the service and customer. We believe allowing privacy document dealers writing their specifications *naturally* would lead one step ahead in creating *usable* tools dealing privacy matching and enforcement.

Enforcement requires translation. While we are motivated using a user-friendly technical language as close as possible to plain natural language for specifying the privacy statements, automated enforcement of user's preference requires it to be specified in machine readable structured information. This dilemma can be addressed by supporting *transformation* of the natural language as needed by the enforcement engine.

There is extensive research about enforcing privacy policy as agreed by the service, e.g. enforcing 'sticky' security policies throughout a distributed application [17], development of schemas such as XACML, EPAL [7] or recent PrimeLife Policy Language(PPL) [8], common specification and enforcement of obligations in the context of multiple trust domain [6]. Given that there is large body of work in enforcement specifications, it is necessary to support translation of the highly usable natural language specifications in existing enforceable schemas and reusing legacy enforcement engines thereby.

Besides enforcement, supporting translation would bridge some other missing links with existing technologies, if natural language is introduced for specifying privacy. For example, posting privacy practices in natural language has also been found hard to read and understand for people [28]. This implies keeping provision for user interfaces to develop based on a structured translation of the natural-like language. Furthermore, automated agents might fail to digest the language as it contains mix-up of detailed, unstructured information (from a machine-reading point of view). These dilemma requires keeping support of translation between policy languages (e.g. from a human-readable one to a machine-readable one).

1.1.4 User needs guided assistance

For a ordinary internet user, defining and adapting privacy preferences, in a way that they protect their privacy properly, is a complex and error-prone task which usually requires some expertise about the domain of basic legal privacy concepts and principles. As security and privacy protection are often secondary goals for ordinary computer users [27], it is indeed not realistic to assume that users will spend enough time and effort to adapt themselves to on privacy configurations [25].

Hence, with the support of Domain Specific Language (DSL) in writing or viewing privacy documents, another major challenge is supporting user with proper guidance in the case of a mismatch.

Adapting privacy document for a match is a complex and error-prone task which usually requires some expertise about basic legal privacy concepts and principles. In the non-electronic world no equivalent task exists [25], which means without experience, user's intuition can not easily comprehend a convergence to match. Without assistance, most users are very likely not to succeed writing their preferences in a way that they are compliant with the service without several trials or even being totally unable to write. To make the situation worse, the user could accidentally define or choose privacy preferences, which are not as privacy-friendly as the users would like them to be [25]. This possibility is supported by the the study that online user's privacy attitude diverge from behavior, i.e. privacy statements seem to have no or few impact on behavior [12]. This implies that user needs *useful* assistance from writing to reviewing her specifications.

Ackerman et al. summarized this as a customer-driven design implication that users would likely to benefit from systems that *help in reaching agreement* [4]. To make a choice, users need to be informed about mismatches of a site's policy with their preferences. This ensures *transparency* between two parties. Secondly, providing users an option that allows them to easily preview or edit the specifications and agree to or reject the transfer of personal information. These are more *noticibility* issues (i.e. notices from service provider's side about their information practices and customers deciding on that) well addressed even in early guiding principles [1]. However, the necessity of suggesting modification to the user to get a match with website's data practices, which remains a major user worry, is not well-addressed to the best of our knowledge. This way, user would get a set of suggestions instead of just being informed of a mismatch.

1.1.5 Usability enhancement needs supporting information

For enabling users to make well-informed decisions, there is a need for user interfaces (UIs) informing them in particular about the privacy policies of their communication partners, matching information, detail description in case of mismatch, and last but not least relevant suggestions that can lead to a possible match.

User interfaces, where the user can pick one preference DSL, e.g. graphical or textual, in order to see and react to mismatches, would be a breakthrough in Privacy UI design. But this depends whether we can propagate the mismatches back to the high level domain specific language, in the form of suggestions of modifications. This issue is closely relevant and should be considered before proposing user friendly DSL to use in specifying preferences.

However, user interface design is not a central issue for this thesis. Psychological or behavior studies of user are not performed either. Rather the focus is to provide details of mismatches that could offer extending research in this direction, i.e. building user friendly interfaces with high visibility of the reasons of mismatch. Thus the result of this work lays foundation for UI research. In fact, UI researcher in PrimeLife took this up and investigating how to visualize policy mismatches to the user [25]. Further user interaction related works are detailed in section 3.1

1.2 Objective

The above context is further structured as requirements in section 2. Based on the situation that users are still uncomfortable with present tools to make well-informed decision in writing their privacy preferences and organizations facing complicacies implementing their privacy policies, we limit objectives for this works to provide

1. Support of Domain Specific Language in specifying privacy documents.
2. An approach for using the DSL for automated enforcement.
3. Supporting transformation to other representation of the privacy document, as needed.
4. Propose modifications in privacy documents in order to get a match.
5. Highlight mismatch reasons across different privacy languages.

Research on these objectives would lead us one step ahead in finding better ways so that end-users are comfortable in creating and managing their privacy policies.

1.3 Structure of the Thesis

The organization of this document is as follows.

Section 2 (“Requirements and Scope”) provides an example scenario and gathers requirements to clearly position the scope and intention of this work in the world of private data management. Section 3 (“Related Work”) reviews the background literature from relevant perspectives. This is followed by details of the design concepts of our approach in section 4 (“Design”). The technologies behind and relevant technical details are in section 5 (“Implementation”). Finally, this document concludes with a brief summary of key contributions and mentioning future research opportunities we see on top of this work.

2

Requirements and Scope

In this chapter, we first introduce a simple user scenario to describe the context of this work. We present our reduced working context in section 2.2 which gathers relevant requirements within our scope. First we describe steps required to improve user-experience from the privacy document perspective. Then we focus on the key necessities when the user is concerned about matching the policies. Next we collect the requirements for a flexible system architecture.

2.1 Example User Scenario

Let's consider a user Alice, who is seeking a travel booking service. She searches for some suitable services in the web and tries to access one. Alice would share personal information (e.g. her contact details, credit card information) with the service. Moreover, this travel service in turn talks to a hotel booking as well as a car rental service and would disclose part of the information (e.g. her contact phone) it got from Alice to fulfill the booking.

Alice is concerned about what information she might reveal while making the booking. She also needs a clear picture where those data might end up. To make it clear, she wants complete control of her privacy.

Alice has a document that describes her privacy preferences in a formal way. The document describes how Alice as a *data provider* wants her data to be treated once she disclosed it to a *data consumer*. Likewise, each of the three service providers in the scenario has a document expressing their privacy policy. It describes how personal data of a user would be handled by the service provider.

Figure 2.1[1] presents a big picture of privacy scenario at multilayer service composition, which includes the scope of previous related work at EMIC as well as this thesis work. Here the data subject (or the end user or client) wants control disclosure of

[1] taken from presentation shown at IEEE Policy 2010 [15]

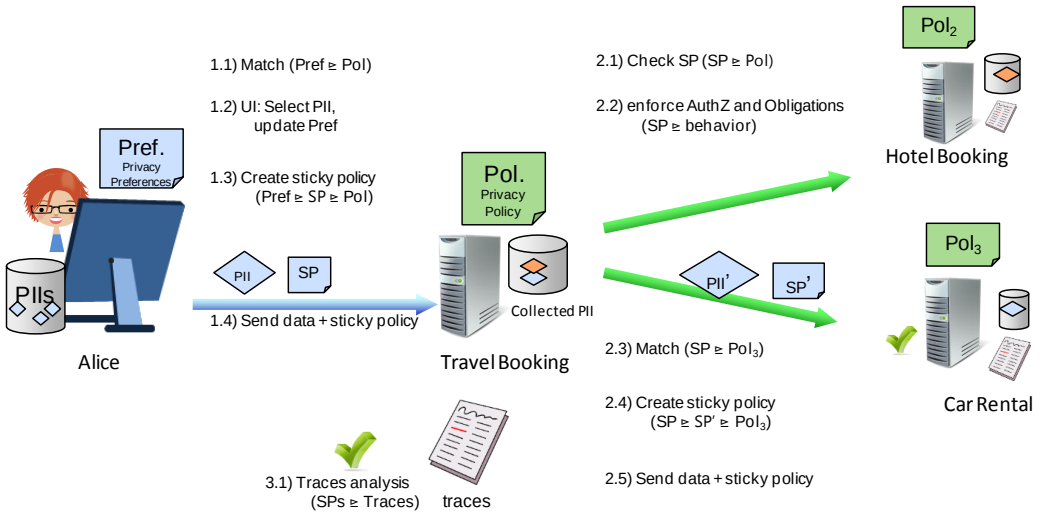

Figure 2.1 End-to-end scenario dealing privacy in service compositions

her private data to data controller (service). This data controller discloses user's data to two other services, again with agreements complying user's preference.

To further describe the end to end scenario of figure 2.1, the privacy policy, *Pol* is sent by the data consumer (travel booking service) to the data subject (user, Alice) to match against her privacy preference, *Pref*. The subject, if finds *Pol* compliant, sends her *Personally Identifiable Information (PII)* along with the sticky policy, *SP*. The sticky policy is the mutual agreement between the user and the service provider. It contains all the rights and obligations the service provider agreed to with respect to the information the user just sent. The data controller stores the agreement, *SP*, for this specific set of *PII*.

The data controller's responsibility is to enforce all obligations it agrees with various users. Here, the travel booking service checks *SP* whether it is indeed the agreements he made with Alice and if found compliant the service enforces it's promises. If this travel booking service shares parts of the data, *PII'*, with a third party (Car rental), it has to verify that the sticky policy of the data, *SP*, is equally or more permissive than privacy policy, *Pol3* exposed by this third party. In essence, this is the same matching step as Alice did with the travel booking service, only that the travel booking service is now in Alice's position of providing data and car rental service is the data consumer. If this matching is successful, the travel booking creates sticky policy with their mutual agreement, *SP'* and sends it along with *PII'*.

All services may log their enforcement information (all their details of dealing with user's data) and let these traces audit by another party to prove whether their commitments were actually kept. This proof of compliance is a business advantage for that service achieving customer's trust.

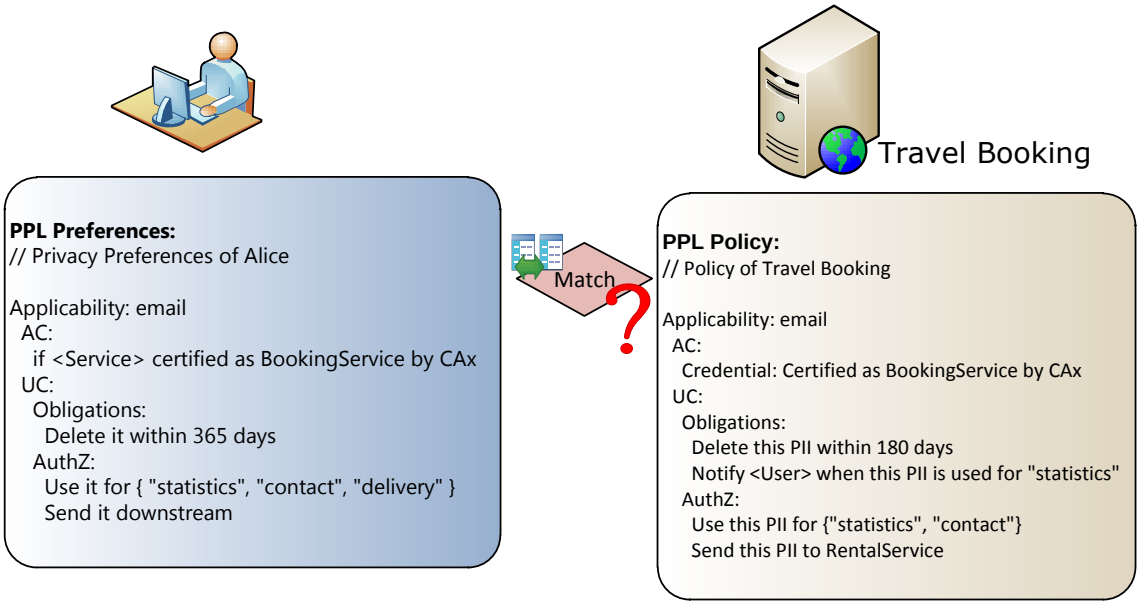

Figure 2.2 Matching privacy documents

2.2 Thesis Scope and Requirements

Whereas previous section describes an end to end scenario of the problem, here we limit our discussion to what is intended to solve in the scope of this thesis and collect the requirements in our problem domain.

Whereas the broader picture covers several important aspects of privacy, e.g., enforcing the agreement the service made to user or verifying whether the promise actually maintained by any *auditor*, the scope of this thesis looks at the scenario from a matching perspective and the focus is on proposing useful suggestion in case of a mismatch.

We gather the requirements from following point of views:

2.2.1 Privacy Document Perspective

It is obligatory to mention details about the nature of personal information that is collected, for what purpose the information will be used, mentioning who would have access to the data, and for how long the information would be retained by the site. However, this section does not intend to provide a check-list of the topics that should be contained inside a privacy document, rather the discussion is reduced only to usability-experience with the document, from both user and agent-developer point of view.

- First of all, a privacy document needs to be written in a simple, easy-to-read language. Moreover, the presentation of the document should be user-friendly.
- Using a natural-like language to specify/write the privacy document. Using controlled phrases from a spoken-language, for example in plain English,

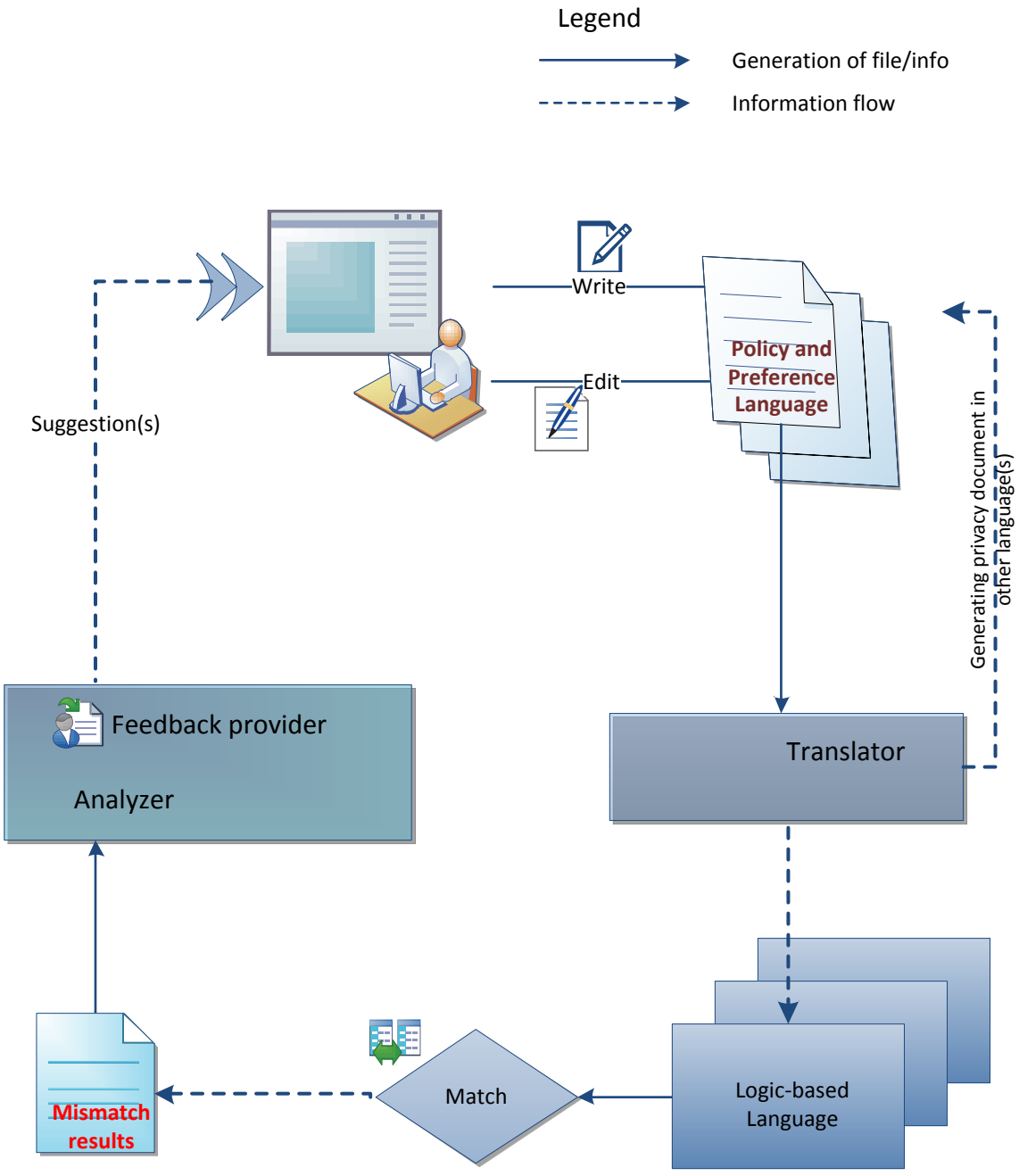

Figure 2.3 Requirement scenario

would make the awkward task of writing far comfortable. This way the user can customize her preference *naturally*, instead of using the configuration tools included by user-agents. However, analyzing natural language to understand policy rules is a difficult task [16]. Neither it is desirable from a privacy document writer to be completely ignorant about domain knowledge and not using *controlled vocabulary* from privacy domain. Moreover, few guidelines can keep the language linguistically simple. This lays the basis for using a precise, controlled natural-like technical language, e.g. domain-specific language (DSL) for privacy documents.

- Visualizing the privacy document naturally. This keeps the policies prominently clear and human-readable. Better visualization is helpful to understand what the other party specifies and what needs to change to get a match. The policy writers may still write their documents in other technical, e.g. any XML-like languages, but they also need a better visualization model, e.g. in DSL. This raises another requirement of supporting translation to DSL from other languages. Having same expressiveness between supporting languages is a constraint here.

- The policy language needs to be either logic-based that allows formal matching or should support being translated to a more expressive logic-based language. This is to support automatic agents to read the policy, understand, extract rights and obligations as well as reason there-on.

- Having same policy language on both user's and service's side is another requirement. However, translation between languages overcomes this constraint if they are with same expressiveness.

- Privacy specification architecture should regard downstream data sharing right from the start. Because privacy is usually not something between two parties, but real world scenarios involve sharing over multiple hops, across different trust domains and involve individual policies for each party. For example, in the scenario in section 2.1, the travel booking service *downstream*s personal information to hotel booking and car rental service.

- Need for a privacy document editor which allows users as well as the service provider to write and configure their specifications in a user-friendly way.

To draw a summary; Communication, between the privacy domain experts, customers, organization policy writers and solution architects, needs to be in an ubiquitous language. We suggest DSL to be the backbone language used by all parties, for example, instead of a technical specification like *<delete> <start> Now </start> <within> 30d </within> </delete>* using a natural-like data retention statement, *I want to delete within 30 days.*

2.2.2 Privacy Compliance Perspective

A user needs good reasons to trust the service provider's way of processing her information. She can be sure about the compliance after matching service's policy against her preference setting.

Checking and generating this *assurance information* about the trustworthiness requires automatic reasoning on the *formal* privacy document. As mentioned in section 2.2.1, automatic reasoning can be supported in a high level user-friendly language as well by translating the document in a more-expressive formal language and reasoning on the latter one.

Secondly, user should get the ability to review and if feels appropriate, correct her preferences. This review is closely related with the reasons for mismatch, that's why, *matching information* should contain as detail as possible instead of having only a *yes/no* message. This additional detail is to provide users the ability to review preferences with the support of meaningful suggestions. This feedback with suggestion can be formulated by analyzing the mismatch information if produced from an extended reasoner.

Thirdly, the user should be able to review the mismatches at the DSL abstraction level rather than the translated formal language statements. This transparency requires hiding the translation gap while preparing suggestions for the user-end from the lower level debugging information.

Lastly, if possible, decoupling the mismatch information, i.e. from the UI coordinates of the privacy document to translated lower level languages' metadata. This would help compliance checking in one side, say in service, and transferring the resulting information to the counterpart client. Systems, which allows user to specify the privacy in multiple language, need to support this feature, e.g. writing the preference in DSL and getting the suggestion in a XML-like representation of the DSL.

2.2.3 Reusing components: Privacy-system architect perspective

From a system developer point of view, it seems a good idea to decouple the privacy components and wire the dependencies as system requires. It should be interesting to think, whether we can move towards a well-designed and standardized architecture that would support integrating existing components to build customized privacy management solution.

UI separation is realized in previous section as we wanted to decouple mismatch information if possible. Another aspect of separation is *pluggable reasoner support*. This means replacing the mismatch analyzer as needed but using other legacy components, e.g. enforcement engine or top-layer privacy languages. *Provision of multiple privacy language* is another possible separation. This means allowing user to write in a new language but using the same underlying privacy management system. This requires supporting translation to the more expressive legacy language.

We address all these requirements, gathered from different perspectives in section 2.2.1, 2.2.2 and 2.2.3, as the scope of this thesis, illustrated in figure 2.3. To acknowledge, we ourselves have not conducted any survey on client, organization or system architect experience but the present state of research (see section 3) serves good reasons for supporting these requirements.

3

Related Work

Given the growing awareness by society of identity theft and other misuse of personal information, it is not surprising that privacy remains a very active area of research for current and emerging technology design [5]. However, the world of privacy has various aspects to care about and we will briefly delineate some which are related to our approach.

3.1 User Interaction Aspects

3.1.1 User experience: Privacy point of view

There exists a stream of research to connect privacy concerns, companies' privacy policy disclosures and customer behavior [22, 13]. For instance, Earp and Baumer [22] studied consumers' behavior and online privacy and showed that the willingness by consumers to provide certain information online greatly depends on who is doing the asking. Bortiz and Won had a detail examination of the privacy policy statements of existing companies, relating them with user concerns. They revealed a gap between privacy policy's individuals value and what companies emphasize in their privacy policy statements [13]. The study reveals that customers' privacy concerns are not adequately addressed in companies' privacy policy disclosures. The need for assisting end-users to negotiate their practices can be inferred from this gap of understanding.

Privacy policies can be posted on websites or contained within contextual texts or sent to users in other form. This document often include long complicated legal statements, which are usually not understood or even not read by the end users [25]. Moreover, users experience discomfort determining whether the website's privacy policy is in compliance and they find it cumbersome [20]. Making privacy policies easily understandable and transparent is therefore an important challenge. The need

for simple user-friendly statements is thereby advised even in early guiding principles (e.g. P3P Guiding Principles Document [1])

Studies exist that investigated customer-driven design issues in privacy protocols and their user clients, e.g. [4]. Users like to benefit from systems that help in reaching agreement and then in exchanging data when the agreement is acceptable in addition with getting assistance in identifying situations where a site's privacy practices is counter to their interest [4].

3.1.2 Use of Natural Language in Privacy Documents

Karat et al. conducted research on the range of skills policy authors might possess. They found some policy authors having a legal and/or business background while others having some technical background [31]. In practice, these policies are being expressed in natural language text (common in organizations), in executable code (common for IT systems), or implicitly (common for individuals) [30]. Reeder et al. conducted a user study to identify common policy authoring errors [38] and among other suggestions, the survey urges the need for authoring the policy in natural language. IBM designed SPARCLE Policy Workbench so that policy authors can write policy rules in natural language using a rule guide. Using SPARCLE, the policy author creates policies using their choice of either guided natural language or structured entry. This system opened research issues in various dimensions for supporting controlled natural language in policy authoring [14, 31, 41, 30, 38].

3.1.3 Policy Presentation

Present privacy policy frameworks support the services demonstrating their policies to customers, e.g. publishing on websites or sending the policy in some structured format (like XACML). But we notice insufficient research on helping better *visualization* of the privacy documents while the necessity of seamless and non-distracting presentation instead of showing up an extremely complex information and decision space, is implied from early user-surveys [3]. Ackerman and Cranor also revealed that a matrix-style user interface for private information over each of P3P's ten dimensions would be overwhelming for most users [3].

Better visualization can benefit the data owner to understand the policies better whereas the same model can support service side policy officers better design their documents. On the other hand, improper abstraction of visualization may lead to incorrect use of the tools. Whitten and Tygar, while studying the use of email encryption technology, pointed out that security mechanisms are only effective when used correctly' and these mechanisms are often not used correctly due to usability issues [43].

Ghazinour et al. present a policy visualization model to assist understanding, analyzing the privacy statements. [26]. This also intended to support policy officers to have a better understanding of the designed policies in order to improve, debug and optimize them.

PrimeLife's work-package 4.3 on 'User Interfaces for Policy Display and Administration' addresses the challenge of making privacy policy easily understandable and

transparent [25]. This work package investigates user-friendly visualization of policy mismatches. They provide options of predefined 'standard' of privacy preferences, which user can choose or customize on the fly. If for example, a service requests more data than permitted by the user's current privacy settings, and the user agrees to it, the user gets the option of adapting her preferences as well as saving the new one as another template. This work package addresses challenges of how to make privacy polices more comprehensible and transparent to end users and how to simplify the process of privacy preference management for them. They are working on developing user interfaces, based on PrimeLife Policy Language PPL, that are informative, comprehensible while legally compliant, but also flexible to handle both simple and complex interactions involving data disclosures to several data controllers or for several purposes and possibly different retention periods. Another contribution of their package is designing a privacy preference editor that allows the user to specify, on an attribute level, what the conditions under which they would accept to disclose the data to a data controller. However, this is still in prototype version and finding a balance between functionality and ease-of-use would remain as a challenge as they add more functionality.

Another recent work on 'Nutrition label Label' for privacy [32], proposes presentation of information to be displayed in short privacy notices. It uses a visualization technique for displaying policies in a two-dimensional grid with types of information that are requested as rows and purposes as columns. Their study shows well-perception of this visualization from test users.

3.2 System Design Perspective

3.2.1 Translation of Privacy Document

IBM's SPARCLE Policy Management Workbench allows user to construct service side's policies using a natural language interface [14]. SPARCLE enables writing policies in natural language, parse the policies to identify policy elements and then generate a machine readable (XML) version of the policy. This is intended to support mapping of policies to the organization's configuration in future to provide audit and compliance tools to ensure that the policy implementation operates as intended.

SPARCLE uses a set of grammars which execute on a parser that are designed to identify the rule elements in privacy policy rules entered as natural language. These elements are then used to create policy visualizations and the XACML version of the policy. However, they moved to using constrained language rather than attempting to parse completely unconstrained natural language.

3.2.2 Debugging High Level Language

We allow DSL for specifying policy and we want to highlight the reason for mismatch at this high abstraction level. Similar motivation exists in debugging paradigm for high level languages. The mismatch reasons are as if the *software bugs* and

discovering the reasons for a mismatch in policies, is as if finding the bugs. This analogy leads us to look at researches on high level language debugging.

Debugging environments for high level languages have traditionally used interpreters, as a classic example, *Interlisp* provides excellent interpreter-based debugging facilities at the source language level [39]. On the other hand, good debugging facilities have been developed using a compiler, but at the machine language and not at the source language level. High level languages if base on an existing complier can support ideal debugging environment, e.g. stepping through the execution of the program, to print values selectively, and to warn of uninitialized variables [24].

Cai et al. realized a debugger for *Aldor* [16], which is a high level symbolic mathematical computation language. Aldor provides an optimizing compiler as well as an interpreter which translates source to an intermediate code, *FOAM*, using the front end of the compiler, and then this intermediate code is executed by a software interpreter. [16] uses this interpretive environment as the context for debugging.

With *interpreted* and *compiled* models of execution for modern high-level languages, a language may also be *translated* into a low-level programming language for which native code compilers are already widely available.

Mernik et al. provided an extensive summary of the current practice of DSL implementation [34]. They discussed various *patterns*, i.e. the most suitable implementation approach for an executable DSL. The *source-to-source transformation*, where the DSL is transformed (translated) into a base language source code is already existing for DSLs like *ADSL, AUI, MSF, SWUL, TVL*.A common approach for implementing DSLs is to create a pre-processor that translates DSL source into a general purpose language (GPL), such as C, Java or C#. A benefit of the preprocessor approach is the potential for reuse of the host GPL development infrastructure to generate executable code.

However, the preprocessing has a serious disadvantage when it comes to the issue of debugging . Difficulty arises when debugging the DSL or any high level language is necessary after translation is done [46], as debugging requires knowledge of both the domain and the target GPL. This results in a conflict of abstraction levels. In such case, the programmer must understand the translated code in the GPL, rather than the higher-level description contained in the DSL, meaning mapping between these two level of sources need to be carefully generated and intelligently. Otherwise, the end-user would not be able to debug with higher level domain-specific concepts and notations, but instead in terms of GPL concepts.

Wu, Gray and Mernik used preprocessing(translating) approach for executing DSL and suggested using eclipse's[1] debugging framework as an interface to support debugging capabilities (e.g. error reporting and debugging are at the level of the generated GPL code) for DSL environments[46]. They used *ANTLR*[2] specification to support automated debugger generation and thus shows the possibility of existing debugging support. Wu have also research that unites the descriptive power provided by the Eclipse debugging perspective and the JUnit testing engine [3], in conjunction with

[1] Open Source IDE, mostly provided in Java

[2] A language tool for constructing recognizers, compilers, and translators

[3] JUnit is a framework to write repeatable tests. It is an instance of the xUnit architecture for unit testing frameworks

the *invasive modification capabilities of a mature program transformation system* [45].

If the translator approach is advocated, *Microsoft Oslo* comes up with a set of future modeling technologies for DSLs [44]. This includes *MGrammar* i.e. instructions on how to parse files for the language and a standard methodology to translate the DSL files to annotated *MGraph*. These features provide an easy way for keeping mapping record for debugging information while translating the DSL.

3.3 User-controlled Privacy Platforms

3.3.1 Compliance Checker

Comparing user's preference with website's data practices is well-addressed in several approaches both in literature and practice. Platform for Privacy Preferences(P3P) is an early recommendation with the goal *to inform Web users of the data-collection practices of Web sites*[18]. P3P was adopted early by many service providers [20].A complimentary specification, conceived by W3C was APPEL which is *for describing collections of preferences regarding P3P policies between P3P agents* [33] and primarily intended as a transmission format. IBM's EPAL was intended for *exchanging privacy policy in a structured format between applications or enterprises* [9]. Simply put, APPEL with P3P is for matching policies whereas EPAL was to locally enforce what the service is obliged by P3P.

Another work, by Backes et al. [10], examined the comparison of enterprise privacy policies using a formal abstract syntax and semantics to express the policy contents. The authors provide formal definitions and rules under which refinement can occur, and incorporate them in an algorithm for checking refinement for privacy policies expressed in EPAL. S4P, a declarative language from Microsoft Research [35], specifies preferences and policies uniformly as assertions and queries written in SecPAL [11] extended with two modal verbs, *may* and *will*. Compliance checking between two parties' privacy documents is simple with this language as it only involves evaluating the queries against the assertions.

Up to now, we looked at matching from service driven perspective. But in some cases, negotiating policies would serve for a better matching model [21, 23]. Research on how efficient privacy negotiation techniques can lead to efficient contracts is found in Preibusch's work [37]. They modeled user's individual utility maximization in multi-dimensionality of privacy, taking into account reducing the negotiation space in a way that suits the given business scenario. Based on a formalization of the user's privacy revelation problem, they modeled the negotiation process as a Bayesian game where the service faces different types of users. Yee proposed an automatic privacy policy agreement checker that automatically determines if the user privacy preference agrees with the corresponding provider privacy policy [50]. The solution computerizes the privacy policies by expressing them in XML-based APPEL. The work kept provision for incorporating the solution in policy negotiation [49] by invoking it to check whether a new offer by either party results in agreement.

However, as our scope is only service-driven scenario, we do not make the solution unnecessarily complex dealing with negotiation techniques.

3.3.2 Related work at EMIC

Under the same context as this thesis work is, a prototype application was developed at EMIC (shown at IEEE Policy 2010 [15])on privacy policies in multi-layered service composition scenario, outlined in the figure 2.1.

This prototype facilitated automating data-sharing decision by matching user's preference and service's policy. The work further supported obligation enforcements, i.e. executing all the promises the service agreed with dealing the private data. Finally, audit mechanism had been integrated to verify whether the enforcement acted in compliance with agreed sticky policy. The audit is based on the generated execution traces.

The purpose for developing enforcement tools was to make the service complying agreed privacy. The enforcement engine executes all the obligations it agreed upon in the past and generates meaningful traces for all data related activities. The auditor verifies these trace messages to track any violation of service's commitment.

This previous work needed several modules to be developed to provide a unified logic-based solution for privacy management for the scenario in figure 2.1, e.g. 1. User interface development, 2. Parsing policy DSL, 3. Parsing rules, 4. Interpreting rules, 5. Translation to *Formula* [4], 6. Formula matching domain, 7. Designing a presentation layer for simulating enforcement scenario based on WWF [5], 8. Enforcement of authorizations, 9. Enforcement of obligations, 10. Parsing traces, which are generated while enforcing, to object model 11. Translation of Object model to Formula for the auditor to verify service's commitment, 12. Formula audit domain.

3.4 Summary

Other than the ongoing work of SPARCLE [14], none of the previous work we are aware of is based on natural language to specify the privacy document. Our approach also resembles this work in that they also separate the policy specification (in natural language) from the enforcement (translated to XACML). However, the work presented here significantly differs from their approach as we decouple all domain specific modules to support a pluggable debugger that provides matching suggestion at the natural language abstraction level. Moreover, the transformation and reasoning modules are pluggable as well, opening the opportunity of translating from one DSL to other (expressiveness is a concern though) or plugging other reasoner on-demand. A generic translator is also supplied for generating the model for system architect to analyze on; from the language, transformation and reasoning domain.

In terms of weaving debugging approach in domain specific language, related works [46], [47], [48] focus similar concern as us. They advocate generating Eclipse framework tools directly from grammar and take benefit from existing debuggers. However, their approach is applicable in those cases when an aspect weaver is available for the generated GPL, whereas we need to translate the DSL for logic based reasoners where we do not find aspect oriented design. We followed a complete different

[4] further details of Formula in section 5.1.3

[5] Windows Workflow Foundation

way by generating the debugging information from the low-level reasoners, i.e. mismatch causes in our case, while generating the result and (re)map the information in all previous and next abstraction levels accordingly. Moreover, we need to translate the DSL in multiple layers (translation architecture in section 4.2) for supporting the requirements (section 2) that introduces complicacies designing the mapping information and propagating the low-level debugging information back to the topmost abstraction level.

4

Design

To help fulfilling the requirements we are imposed (gathered in section 2), we have designed and implemented (section 5) a privacy system architecture from matching point of view. We followed a generative approach to separate the UI, matching, enforcement and debugging concerns. We generate intermediate structured representations from the policy DSLs. These are further translated to a reasoning model for analyzing the mismatches. We also support translating the policy DSL to other DSL, i.e. other structured representation. All these transformations are automatically generated as directed by the knowledge of each domain, i.e. policy language, transformation rules and reasoning domain. In this chapter, section 4.1 is intended to outline the design principles, followed by section 4.2 describing our approach to link different aspects and section 4.3 discusses how we can discover the mismatch reasons and highlight those in end-user abstraction level.

4.1 Domain-Driven Design

Let the metamodel drive the implementation. It is important that the metamodel drives the implementation of any DSL. What we mean is, the language structure, parsing grammar and transformation rules need to be specified a prior and the implementation of a matching engine should follow this metamodel. To reflect the DSL in implementation in a very literal way, we designed a domain-knowledge driven generic parser which translates the DSL in intermediate standard representation e.g. XAML [1]. This XAML is further translated to another DSL, i.e. *PPL* [2] and to a reasoning model in *Formula* [3], to feed them into next abstraction level modules, e.g. matching and enforcement engine.

Isolation of the domain. Policy DSLs evolve over time and many be defined differently by different organizations. This is true for reasoning knowledge as well in

[1] Microsoft's declarative Extensible Application Markup Language (XAML)
[2] PrimeLife Policy Language (PPL) is an extension of XACML
[3] for further details about Formula, see 5.1.3

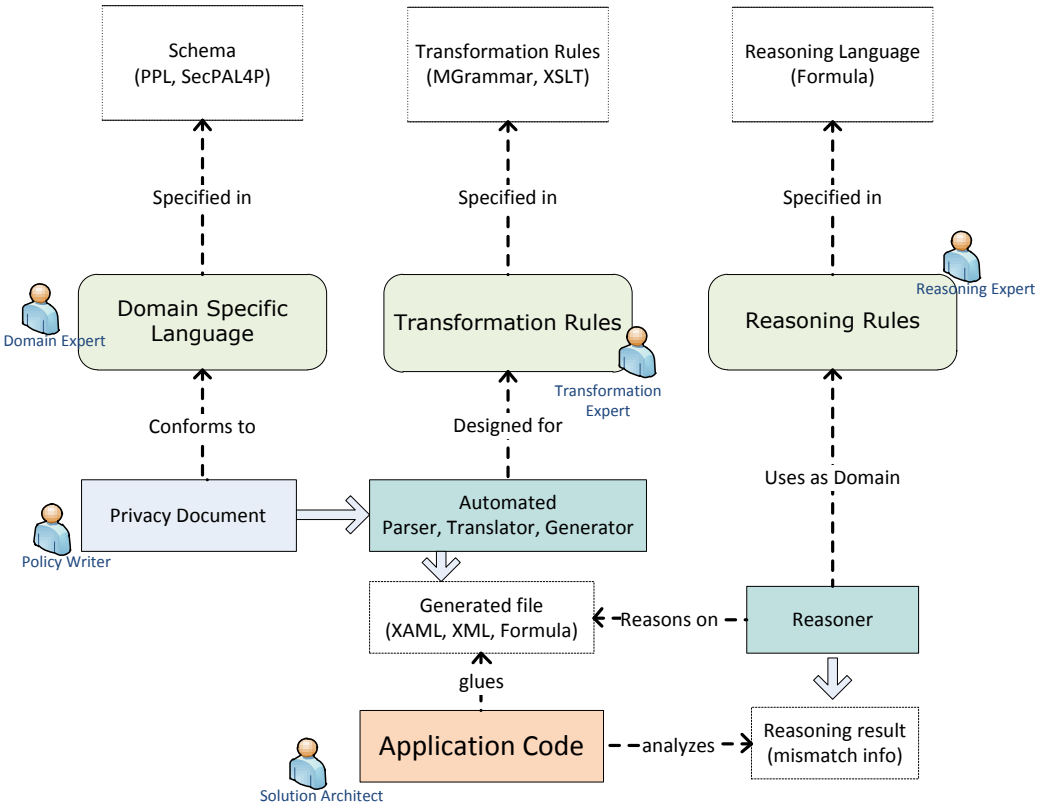

Figure 4.1 Domain-driven architecture: Metamodels drive the implementation

the case where someone needs to use other matching engine but keeping same upper layer. This implies the underlying infrastructure to be decoupled from the domain knowledge. We advocate for supporting intermediate representations to which the upper layer is translated and which would be used in implementation. This is how we maintain separation of abstraction. This also makes the architecture reusable; in the sense that, the same enforcement engine could be used for any different DSL or reasoned by any other matching engine.

The idea of decoupling different layers via intermediate representation, is popular for high-level languages, e.g., Common Language Infrastructure (CLI), in which applications written in multiple high-level languages can be executed in different system environments without the need to rewrite those applications to take into consideration the unique characteristics of those environments [2]. Another existing language is Microsoft's *XAML* (XML-based Extensible Application Markup Language) [4] which can be easily expressed in more traditional .NET language, e.g. C#. In our approach, we used XAML as the intermediate representation of policy DSL.

Linking Policy to Implementation. Another motivation discussed in section 1.1 is to link the privacy policy to enforceable information. This addresses the dilemma that in one perspective, an upper level user is comfortable with natural-like languages to understand and specify their privacy whereas the automated agents and enforcement engine are less error-prone if the policy is represented in a structured machine-readable format. We propose the linking by supporting automated translation of policy DSL to a structured format.

Figure 4.1 summarizes the domain-driven design principles discussed above. The high-level language, corresponding translation rules and the reasoning domain are designed by corresponding experts. A generic parser translates the upper level to intermediate formats according to the transformation specifications. The system architect gets the DSL as an enforceable structured format. This way policy DSL would be automatically and correctly reflected in the enforcement system. The matching engine also works on the same intermediate representation. The lower level reasoning model, upon which the reasoner works on, is generated from the intermediate representation as well. The reasoning domain, which directs the logical analysis, is also specified by an expert beforehand, in the format needed for this particular reasoner.

4.2 Reusing Components: Link by Translation

One motivation for this work is designing support for reusing legacy technologies and systems where possible. As discussed in section 2.2.1 we need a link between human and machine readable policy languages as users can specify their privacy in either format and switch to the other format for viewing or editing purpose. Translation to a machine-readable format is also necessary to automate matching and enforcement. The policy language can further be translated to a more-expressive reasoning model. This supports decoupling the matching engine.

[4] XAML Overview. http://msdn.microsoft.com/en-us/library/ms752059.aspx

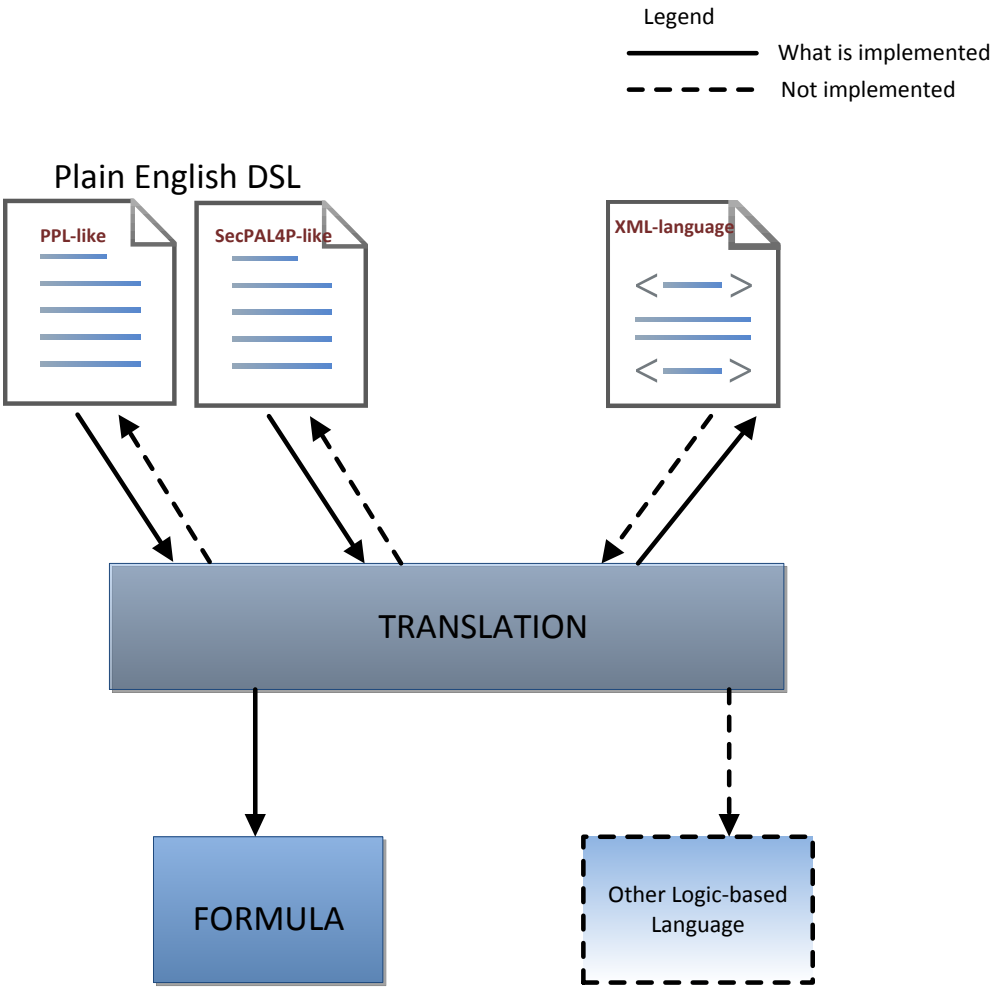

Figure 4.2 Towards (re)using existing components: link by Translation

Figure 4.2 gives an illustrative overview of to what extent we support translation. Here, two natural-like DSLs, i.e. PPL-like [25] and SecPAL4P-like [35], are translated to a structured machine-readable DSL. This may be required to be done in the opposite way as well, i.e. user writes in a XML-like language (e.g. PPL, XACML) and we need to translate it back to the natural-like DSL to give the user a better visualization. We do not implement this feature due to technological limitation but keep provision in the design in the way that graphical visualization models could be easily built on top of our translated structured DSL. Later, policy from both representations are further transformed to a logic-based language, in our implementation which is *Formula* (more details in section 5.1.3).

4.3 Debugging Perspective

A DSL source-level debugger is a critical tool to assist the user in discovering the reason for mismatch as precise as possible. In general, high level languages, if translated for execution, makes the debugging support difficult [46]. We address the problem with additional mapping processes while translating to next abstraction layer and interpreting this mapping against the mismatch information while generating the suggestions at topmost abstraction.

An illustrative overview of the debugging perspective is shown in Figure 4.3. The generic parser generates the intermediate representation as well as mapping information with the DSL source. This mapping process keeps track of source code segment, e.g. line and column number, with the corresponding segment in the intermediate file. Similarly, next level translation to reasoning model is also responsible for generation of another mapping file. This leads us to a consistent tracking of the DSL segments as we translate one abstraction layer to other and requires. This way a transformation domain expert simply writes the rules and additional mapping information is auto-generated requiring no interaction from the end-user.

The additional details of a mismatch, what we need to enhance usability, should come from the matching engine. If the reasoner behind the engine does not support provision of analyzable details we suggest to enhance the reasoning domain knowledge. For instance, the reasoning engine we use (i.e. *Formula* [5]) does not generate additional information that can be used to pinpoint the reasons. But what we can always do is to generate some detail level of states from the reasoner and analyze those with additional domain knowledge. This way, the separation of domain with the implementation, which is one of our design principle, is maintained. The key benefit is, we can plug another reasoner and use the same upper components except the reasoning domain to change. However, as we need to investigate the reasoner output for analyzing the mismatch reasons, plugging another reasoner implies knowing the output format of this reasoner in advance, so that the next layer analyzer component could be reused.

The mismatch reasons are discovered in the reasoner layer which has no clue about the existence of upper layers. We re-map the results back in the abstraction level user expects, i.e. DSL. User may also require step-by-step guidance, e.g. putting

[5] more details about Formula specification is in section 5.1.3

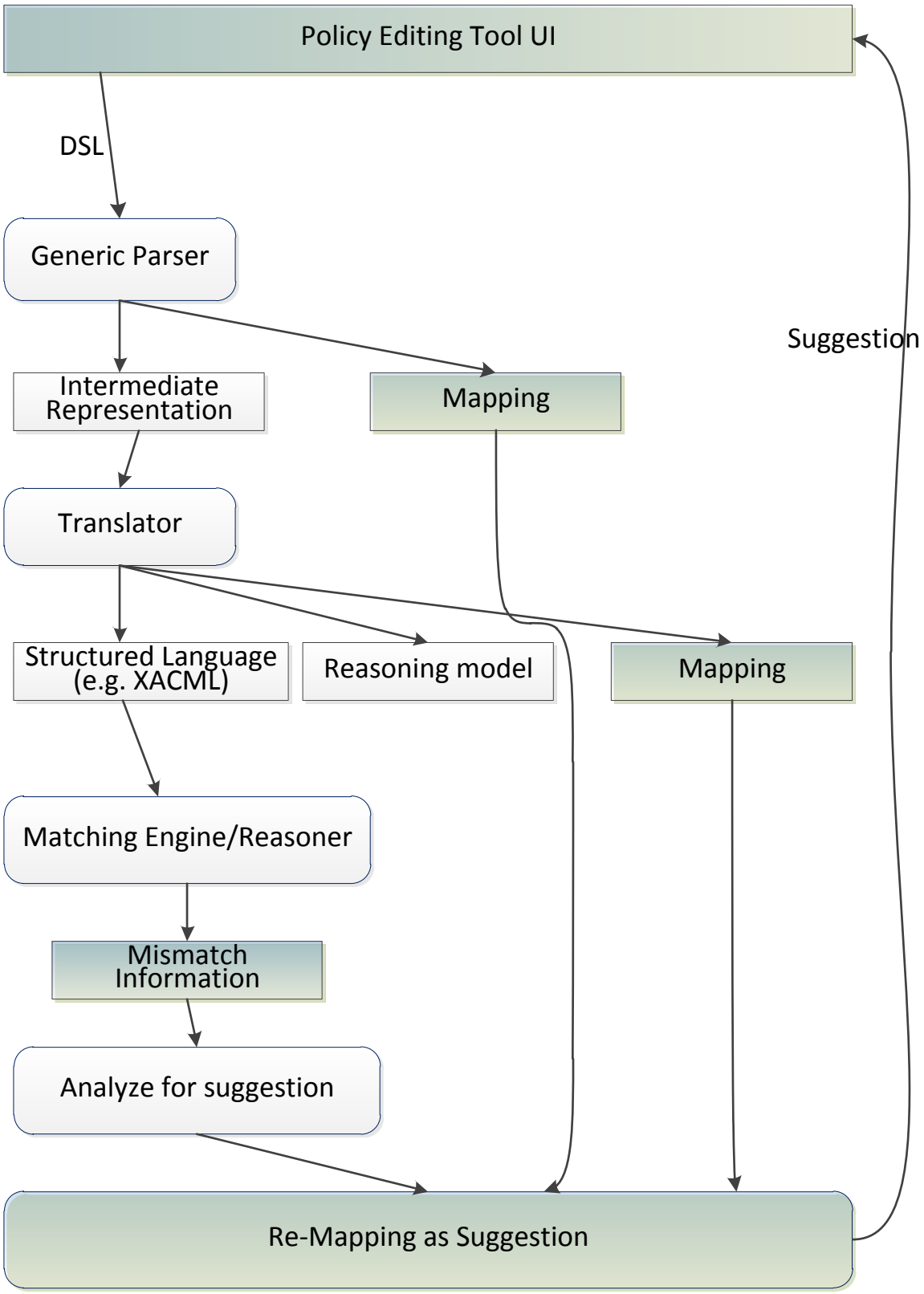

Figure 4.3 Debugging Aspect in a Policy Editing Tool

that suggestion first which requires least modification and moving to next as users directs. The re-mapping component takes user's command and maps the mismatch reasons back to the DSL abstraction level accordingly.

5

Implementation

We have gathered the requirements for a privacy management system from a matching (i.e. helping user and service achieving an agreement) point of view in section 2 and realized the design issues for such a system in section 4. This chapter describes the implementation of a proof-of-concept prototype. First we highlight the technologies behind (section 5.1), following a description of the developed components (section 5.2). A demonstration walk through the prototype is detailed in appendix B.

5.1 Technologies in Use

This section introduces the software tools and methodologies, employed to develop the architecture envisioned in previous chapter.

5.1.1 DSL Technology

There are many tools for building DSL, e.g. ANTLR [1], Boo [2]. We used Microsoft's SQL Server Modeling platform (former codename, *Oslo*) [3]. The toolkit ships with a GLR-based text parser [4], named *MGrammar*, that basically parse text and output an AST [5] node graph [44]. This supports us creating a general parser mechanism with .NET framework at the back and generating an in-memory syntax tree. The

[1] ANTLR (ANother Tool for Language Recognition) provides a framework for constructing recognizers, interpreters, compilers, and translators

[2] Boo is a flexible CLR language with an extendible compiler architecture, suitable for DSL

[3] Data Developer Center. http://msdn.microsoft.com/en-us/data/

[4] GLR parser (Generalized Left-to-right Rightmost derivation parser) is an extension of an LR parser algorithm to handle nondeterministic and ambiguous grammars

[5] AST (Abstract Syntax Tree) is a tree representation of the abstract syntactic structure of source code written in a programming language)

intermediate representation of the policy DSL is auto-generated by walking the AST, enabling us maintaining clear separation between the metamodel and implementation.

MGrammar Language enables information to be represented in a textual form that is tuned for both the problem domain and the target audience. The language provides simple constructs for describing the shape of a textual language; that shape includes the input syntax as well as the structure and contents of the underlying information. To that end, *MGrammar* acts as both a schema language that can validate that textual input conforms to a given language as well as a transformation language that projects textual input into data structures (i.e. *MGraph*) that are amenable to further processing or storage. The latter feature supports us further translating this structured output to another representation.

One missing feature with *MGrammar* is that there is no way to automatically go from the parse tree back into the textual form. However, it supports attaching annotations (called attributes) to production rules. These annotations can have structure (the same kind of structure as the parse tree). This enables attaching source level information, i.e line and column number, as we walk the parse tree. Another interesting feature of *MGrammar* is that it has modules. This opens up the possibility of reusing grammars for languages and fragments of languages. In practice, this feature helps maintaining consistency in multiple parsing grammars.

5.1.2 XML based technologies

For intermediate representation of the policy DSL we need a language which is structured enough to be further transformed and which supports transformation experts independently write their rules. XML languages are suitable for this purpose, if translation rules, if written in XSLT [6], would keep the domain knowledge not mixing with the implementation code. XSLT is designed specifically to transform XML to other XML, in our case to other policy language, e.g. PPL as well as XML to Text, what we need for translating to a reasoning model, Formula (discussed in next section). This way the system architect can plug another underlying reasoner and choose translation stylesheet accordingly for generating model for that platform.

As the need for an XML language for the intermediate representation is clear, we used Microsoft's declarative Extensible Application Markup Language (XAML) for this purpose [7].

5.1.3 Logical Analysis: Matching Engine

The policy language needs to be either formal or should be translated to a formal language to support automated reasoning. In our approach, a domain expert writes the XSL transformation sheets that support translating the intermediate language (i.e XAML) to a formal language. We use *Formula*(Formal Modeling Using Logic

[6] XSLT (Extensible Stylesheet Language Transformations) is a declarative transformation purpose XML-based language.

[7] Microsoft XAML Overview. http://msdn.microsoft.com/en-us/library/ms752059.aspx

Analysis)[29] for this purpose. It is a unified framework for specifying DSLs and model transformations. Formula is a LP language, meaning that the basic constructs of the language are precise logical statements. This precision makes it useful for analysis of and code generation from high-level specifications.

Formula specifications are separated into logical units called *domains*s. A domain encapsulates a set of data structures and constraints used for modeling some part of a software system. Declarations within a domain are scoped in a specification, which acts as a container for declarations. A *model* is simply a set of data instances built from the constructors of some domain. Intuitively, a domain represents a family of models by providing constructors and constraints.

In our approach, a reasoning expert writes the *Formula domain* and the transformation expert specifies the corresponding rules for generating the intermediate policy representation to *Formula model*. The model is simply a set of data instances built according to the constructors of the domain and is just another lower-level representation of the policy document. The model contains important matching information that must be extracted. This is accomplished by adding rules to the domain.

5.1.4 Policy Editor

Given the policy writers and end-users are assumed to have low technical skills (at least they need not to be technically sound), they are unlikely to express their own privacy directly in the XML-based PPL syntax. That's why, we offered a user-friendly interface where they can write their appropriate privacy requirements and fine tune if necessary. Similarly motivated works face a dilemma between simplicity and expressiveness regarding this issue [25]. We decided a trade-off by allowing user writing their policies in the DSL. At the same time, she gets a separate UI view where the automatically translated PPL representation is shown. The user, if possess technical background, can validate her statements in the PPL view whereas some more user-friendly view (e.g. graphical), if the user expects along with the natural-like DSL, could be generated on top of this machine readable PPL representation (or whatever structured language we target).

We thereby developed a rich text editor that allows users write their privacy as in a standard text editor. In case of a conflict between user's expectation and website's data practice, we highlight corresponding words (i.e. mismatch reasons) in the editor as a suggestion to edit. For presenting the XML-like PPL representation of the policy, a simple easy-to-view XML-viewer is developed on top of the structured policy. We acknowledge the need for a more user-friendly graphical policy viewer that would hide the XML in background and present the content only for a better understanding of the privacy statements. However, our support for automated translation in XML-like languages lays the foundation for any further development of smarter UI.

We used Microsoft's Windows Presentation Foundation (WPF) technology [8] for developing the Editor and Viewers.

[8] WPF is a graphical subsystem for rendering user interfaces in Windows-based applications

```
1. PPL Policy:
2. // Policy of Bookshop_Service

3. Applicability: email
4.      AC:
5.          Credential: Certified as Bookshop by BusinessAuthority
6.      UC:
7.          Obligations:
8.              Delete this PII within 180 days
9.              Notify <User> when this PII is used for "statistics"
10.         AuthZ:
11.             Use this PII for "statistics", "contact"
12.             Send this PII to Shipping_Service
```

Figure 5.1 Example Policy DSL

5.2 Relevant Technical Details

Our architecture (section 4) follows a generative approach where input files for different modules, which are not predefined by domain experts, are produced by our system. Domain experts define the *MGrammar* files for parsing corresponding DSL policy, *XSLT* files for transformation and *Formula Domain* with reasoning knowledge for matching. Our framework supports automated generation of intermediate representation of the policy DSL (which is fed to matching engine and could be used for enforcement-purpose systems), transformation to other structured policy DSL (here PPL, additional stylesheets can support others, e.g. XACML), translating to a low-level reasoning model (e.g. in our approach, Formula) for logical analysis. This is how we provide a practical technique of reusing legacy components. The debugging perspective needs mapping files for associating contents of different layers, which we also generate during the translation processes. An illustrative overview of our approach is shown in Figure 5.8.

5.2.1 Translation components

Before describing the details of the debugging procedure, we explain the translation components which provide the base of logical analysis and generating feedback for top DSL layer. We present a very simple policy DSL that would be used to illustrate the concepts (Figure 5.1). This is a policy document of a service provider saying how it will handle user's personal data. The document is in a higher-level language inspired by PPL. What the service states is that it collects an email address and would delete this in half a year. Moreover it would notify the user when her personal data is used for statistics. It also specifies for what purpose the PII would be used. It also mentions the possibility of sending the personal information to a third party.

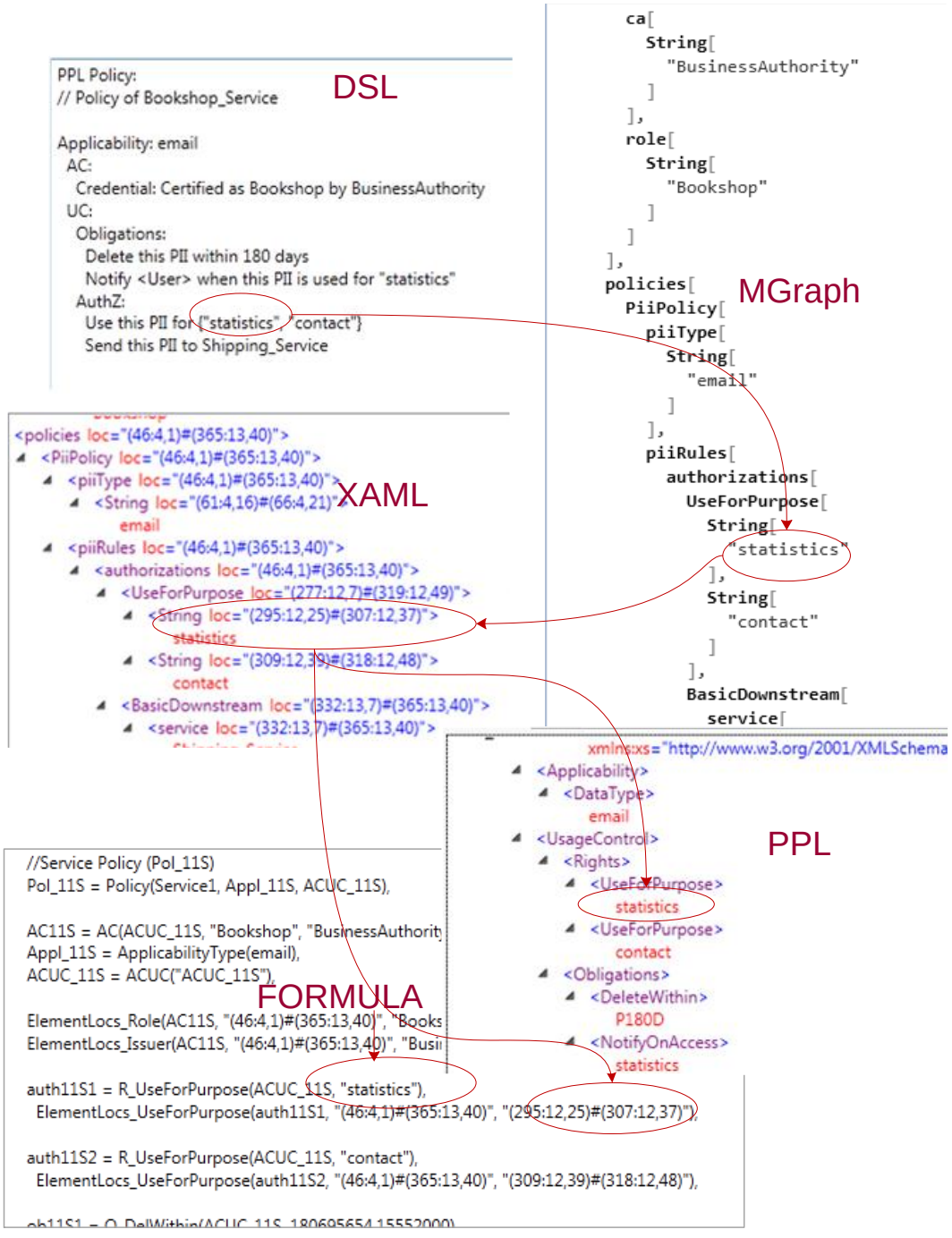

Figure 5.2 Translation in different representation: Keeping mapping information

```
<AC>
      <ca loc="(5,42)#(5,59)">BusinessAuthority</ca>
      <role loc="(5,30)#(5,38)">Bookshop</role>
</AC>
...

      <piiType loc="(3,16)#(3,21)">email</piiType>

...
<authorizations>
      <UseForPurpose>
            <String loc="(11,25)#(11,37)">statistics</String>
            <String loc="(11,39)#(11,48)">contact</String>
...
```

Figure 5.3 Intermediate XAML file, annotated with mapping information

Similarly, the user specifies her privacy in DSL. We parse each DSL to *MGraph* as specified by the *MGrammar*. Then the parser walks through the *MGraph* and generates the intermediate representation of the policy document, i.e. in XAML. This XAML is also annotated (a simplified fragment shown in Example 5.3) with the mapping information with source DSL. This XAML is further translated to a structured language, *PPL* as well as a lower-level reasoning model expressed in *Formula*. All these representations are automatically generated as directed by the pre-specified domain knowledge (expressed in XSLT, MGrammar) and we do not loose mapping information while translating. For example, the line and column number of "statistics" text in Policy DSL is rightly reflected in the XAML (compare example5.1 with example5.3). We maintain the consistency of mapping information in all the representations (figure 5.2 gives an illustration).

Figure 5.5 further explains the above-mentioned procedure step by step. Switching between different translation views are shown with a walkthrough of the prototype in Figure B.3.

5.2.2 Matching and Suggestion

The reasoner module requires knowledge for matching the preference and policy documents. A domain expert incorporates this in *Formula domain*. The analyzable model is generated from the translation module (discussed in previous section). We reason on the model file based on the domain knowledge and get a match result (*yes/no*). However, we want more precise mismatch reason(s) to feedback user as suggestion. Although the our formula engine provides more than a boolean response, but we need additional details about the mismatch. Therefore, we extended the reasoning domain incorporating additional *rule*s to generate mismatch reasons, tied with mapping information, at a meaningful granularity.

```
...
//source location in DSL
ElementLocs_Preference: (pref:Preference, LocUser:String, ...

// preferences that apply to parent also apply to children
Preference(user, ApplicabilityType(childType), acuc):-
 Preference(user, ApplicabilityType(parentType), acuc),
 DataTypeIsType(childType, parentType).
...
CompatibleACUC(acuc_pref, acuc_pol):-
 Preference(_, _, acuc_pref), Policy(_, _, acuc_pol),
 fail IncompatibleACUC(acuc_pref, acuc_pol).

CompatiblePref(user, appl, acuc_pref, acuc_pol):-
 Preference(user,appl,acuc_pref),CompatibleACUC(acuc_pref,acuc_pol).
...
qWrongSend:? Send(user, data, service), DataIsType(data, dataType),
 Policy(service, ApplicabilityType(dataType), acuc_pol),
 fail CompatiblePref(user,ApplicabilityType(dataType),acuc_pref,...
...
qSendOK:? !qSendWithoutPref & !qSendWithoutPol & !qWrongSend.

conforms:? qSendOK & !qWrongPref & !qWrongPolicy.
```

Figure 5.4 Formula Domain knowledge

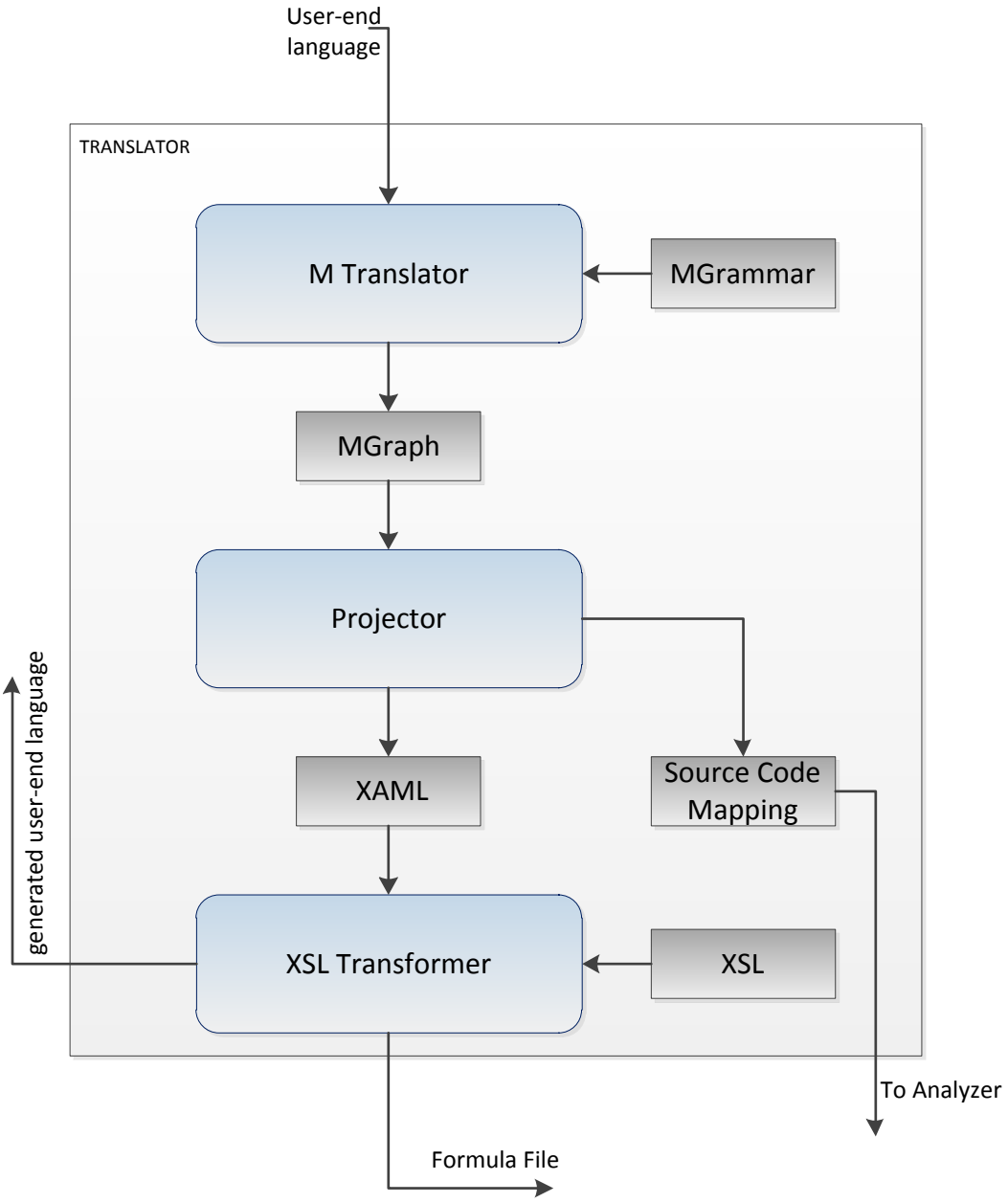

Figure 5.5 Translation module

Figure 5.4 provides an impression of how the domain knowledge is represented in *Formula*. We define *constructor*s specific to our domain, provide *rule*s to infer new knowledge and *queries* to check *conformity* of a *model*. We automatically generate the basic definitions for the model from the XAML (figure 5.9. The data definitions include the privacy policy specification *formally* and contain the mapping information as well. We need this mapping details to re-map the mismatch values back to upper layer DSL.

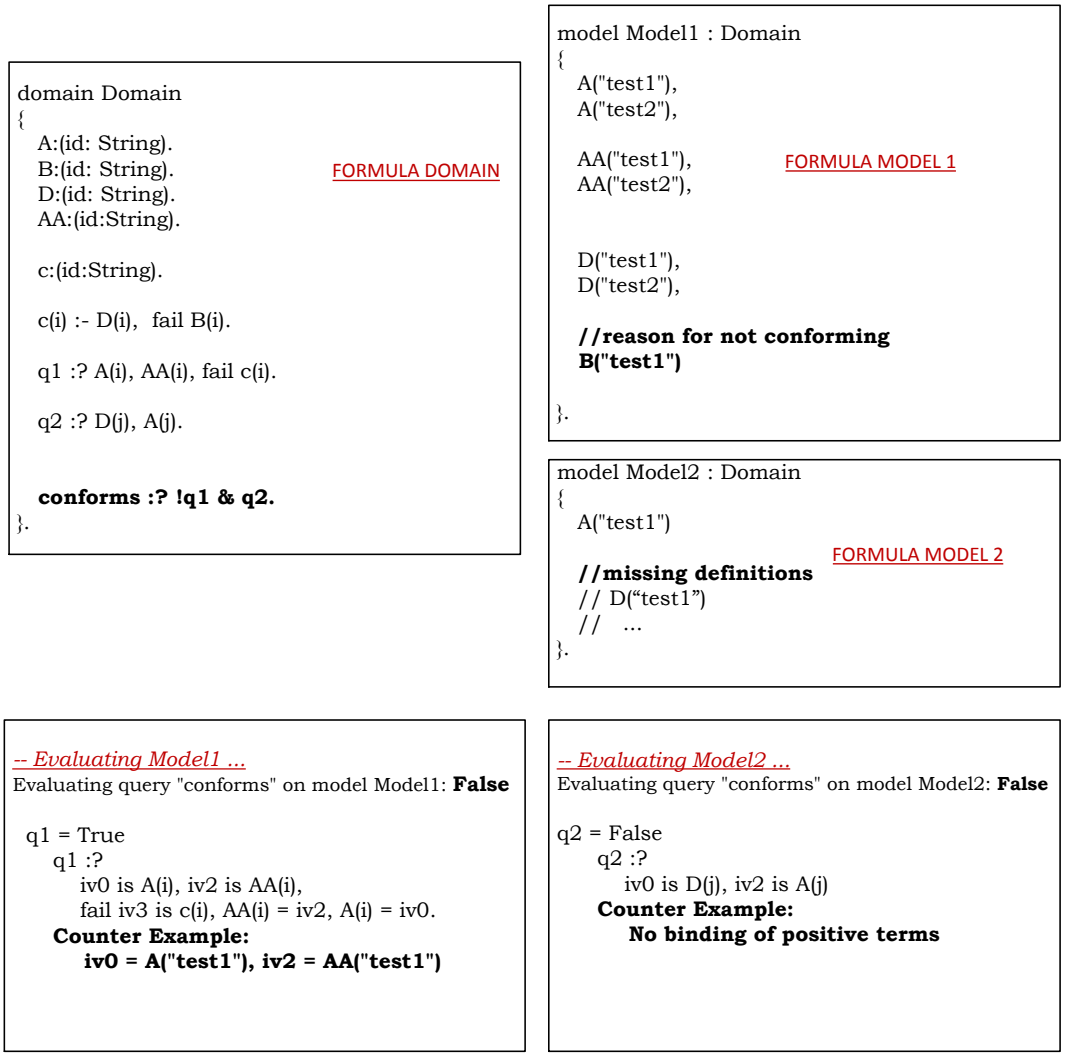

Figure 5.6 Insufficient details from Formula output

The *Formula engine* checks the conformity of the policy model with the domain. However, in case of a non-conformity (i.e. mismatch of privacy documents) the engine can not provide enough besides to extract exact mismatch reason(s). To what extent the formula engine provide additional details is shown in Figure 5.6. Both *Model1* and *Model2* are non-conforming with *Domain*. The *Domain* insists (*!q1* in conformity) not to have *B* where there exists a *D* with same id and to have *D* existing (*q2* in conformity) when there is a *A* with same id. The evaluation result of *Model1* does not directly lead to the fact that existence of *B('test1')* is the actual

reason for not conforming. The output is more vague while evaluating *Model2* that gives no idea that adding D would lead us towards a conformity.

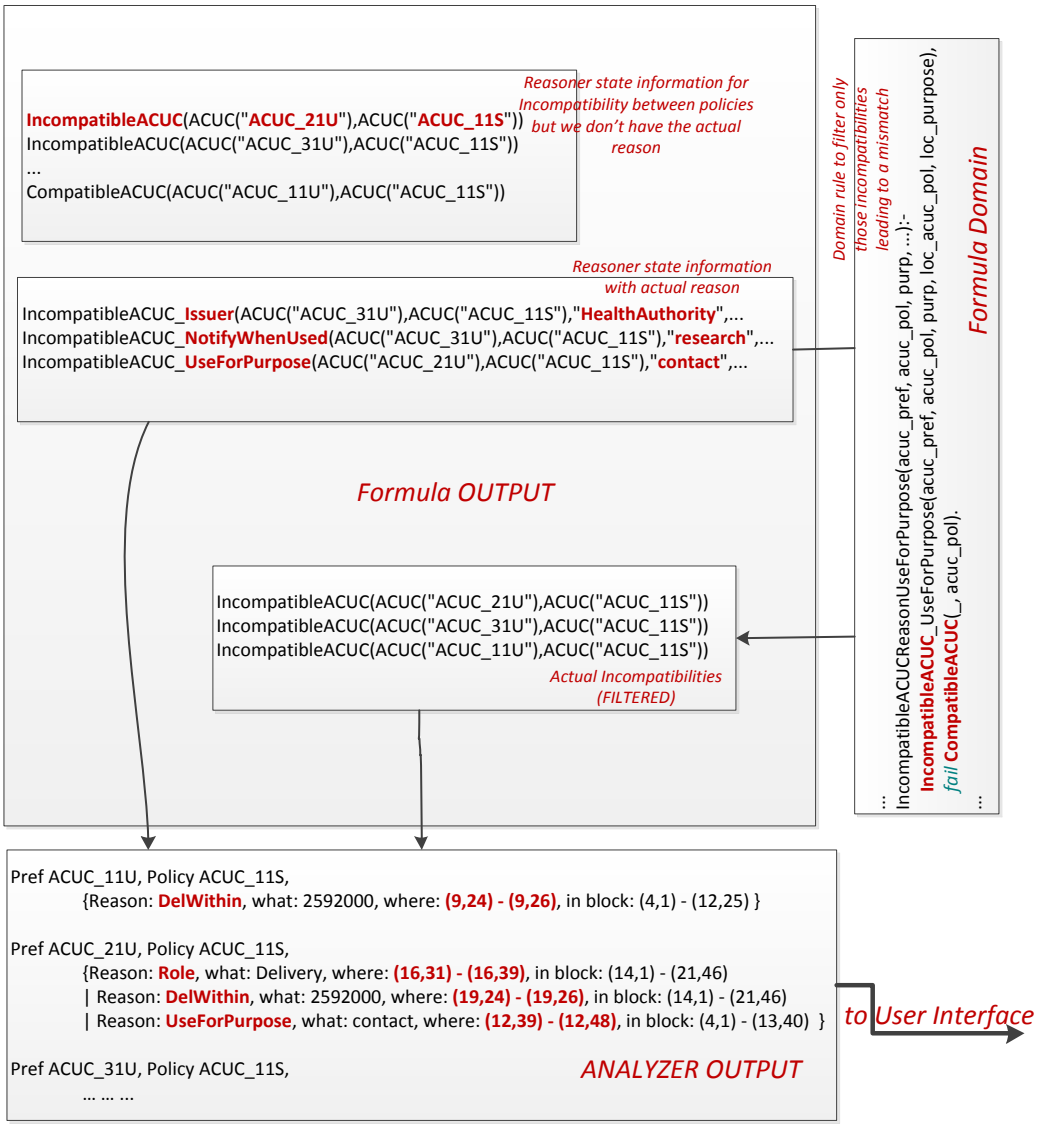

Figure 5.7 Additional knowledge to gather actual mismatch reasons

We address this problem by incorporating additional rules in the *domain* to generate more detail knowledge about non-conformity, i.e. mismatch. In case of a non-conformity i.e. mismatch, our tool extracts this additional knowledge, process it and finally associate the reasons with mapping information. Figure 5.7 further depicts the idea. Although knowing an incompatibility between policy segments are enough to check conformity, we incorporate additional rules in the domain to generate more *state*s that contain details about the reason behind. We further filter these extra states against actual incompatibilities. The *analyzer* extracts these additional state information, process those and generate a report (e.g. conforming a schema) with actual mismatch reasons. This report also contains the information for re-mapping these low-level outputs back to upper-layer DSL. This report is forwarded to the UI

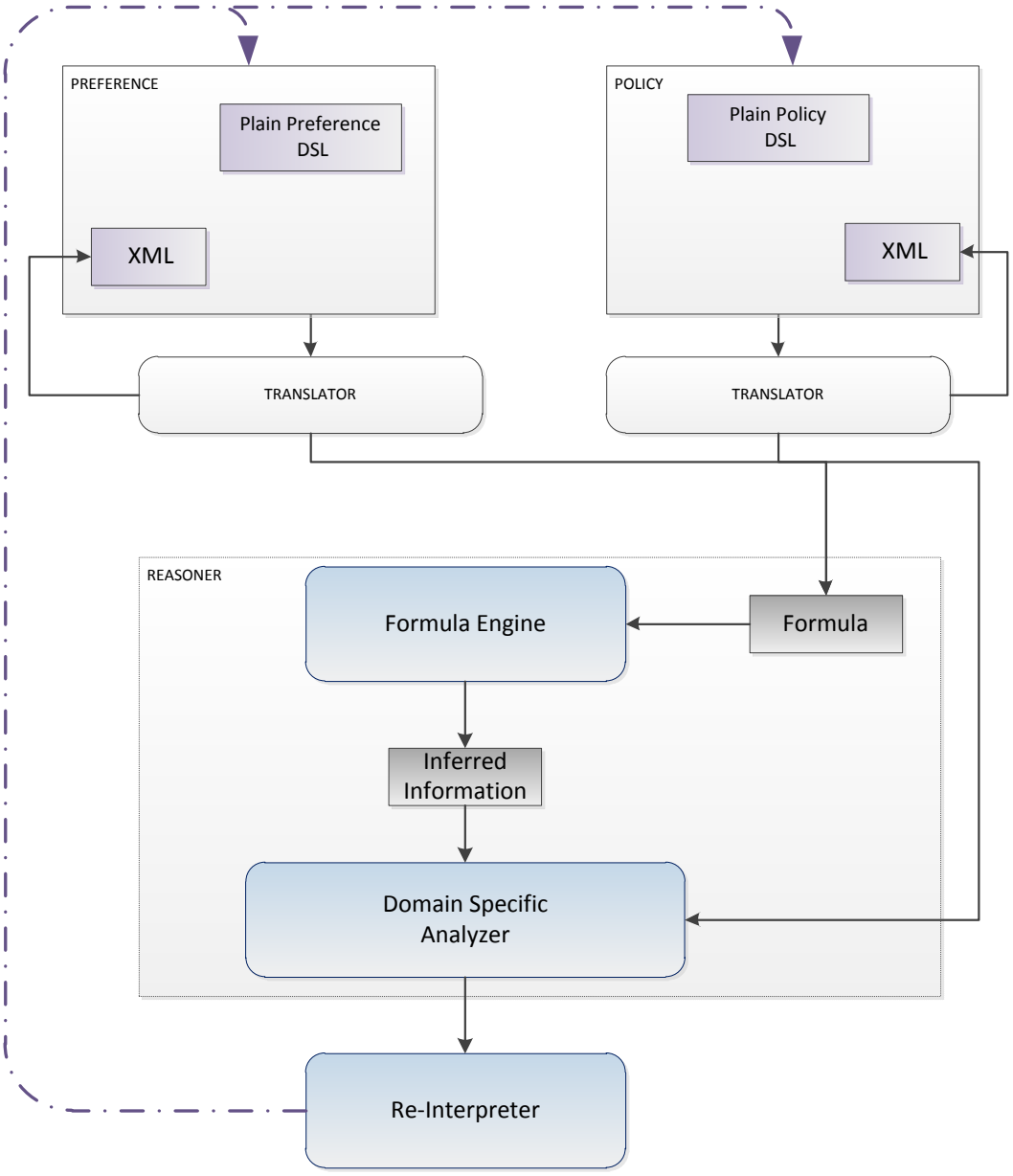

Figure 5.8 Implementation blocks: step by step

```
//User Preferences (Pref_11U)
Pref_11U = Preference(Alice, Appl_11U, ACUC_11U),
AC11U = AC(ACUC_11U, "Bookshop", "BusinessAuthority"),
Appl_11U = ApplicabilityType(email),
auth11U1 = R_UseForPurpose(ACUC_11U, "statistics"),
...
ElementLocs_Role(AC11U, "(6,31)#(6,39)"),
ElementLocs_Issuer(AC11U, "(6,43)#(6,60)"),
ElementLocs_DelWithin(ob11U1, "(9,24)#(9,26)"),
...

//Service Policy (Pol_11S)
Pol_11S = Policy(Service1, Appl_11S, ACUC_11S),
...
```

Figure 5.9 Formula Model: Automatically generated

component. This structured report could also be sent to the service side so that it gets informed about a client-side mismatch scenario.

Figure 5.8 further illustrates how different modules in our architecture communicate and serve the debugging perspective.

Thus in case of a mismatch, this tool automatically guide (by highlighting that corresponding text) the policy writer to a *set* of mismatch reason(s) which, if edited, would lead to a match. We generate all possible sets. However, how user wants the mismatch combinations to be presented is more a UI concern and we do not address this issue deeply. What we do, is showing the set with minimal cardinality (i.e. least number of mismatches) as we find the documents mismatched and wait for user's command for showing other suggestions.

6

Conclusion and Future Work

The work accomplished in this thesis is done in the context of PrimeLife project [1]. Research related to this work addresses a long-term goal of empowering end-users with usable and effective tools. Introducing user-friendly DSL for privacy document, automated comparison of policy statements to each other and in case of mismatch, precisely guiding the end-user with conflicts are some important steps towards this goal.

As the internet continues to evolve, new privacy enabling technologies need to come up to make it a safer place for transaction and to share personal information. User needs to trust the web when providing sensitive data. Otherwise the growth in e-commerce would be slowing because of consumer unwillingness to supply information. Similar discomfort also exists from organization point of view. Procedures they follow are semi-automated and many of them can not verify whether the privacy policies they publish to user are in fact what they practice. This work is intended to provide usable tools for end-users in both sides, to create and manage their privacy policies. Suppose, all services are bound to write their privacy practices in near future. This would require their business customers specifying their expectations as well. Both parties need a convenient tool to define their practices. They need help finding conflicts early and edit the policies if required. Writing in a natural way and getting guided assistance for detecting mismatch are critical to them.

We gathered related concerns from various aspects and advocated a set of guidelines which existing approaches need to consider for better usability and effectiveness. We further developed a proof-of-concept prototype on proposed design. We allow both parties, i.e. user and service side policy writer, specify their privacy in DSL. In case of a conflict, we analyze our reasoning model and filter all possible conflicts at granular level, which if resolved, would lead to a match. We not only make a report of the mismatches, but also relate them with the end-user's abstraction level, e.g.

[1]The research leading to these results has received funding from the European Community's Seventh Framework Program (FP7/2007-2013) under grant agreement no. 216483 for the PrimeLife project.

highlighting in the policy editor. We also consider priority of suggestions and sort them as the policy writer commands.

We suggest transforming the policy DSL in an intermediate structured representation. This helps achieving other important usability supports. This representation can

1. Be transformed in other more-expressive policy language(s) (we translate in *PPL*) which may be used either to get a structured visualization of the document or any other graphical DSL could be built on top of it. This also enables reusing a legacy system which uses that structured language on top of it.

2. Be transformed to a formal reasoning model (we translate to *Formula*) that could be analyzed for granular mismatch reasons.

3. Link the DSL with the implementable policy. This way the organizations can make sure what they publish as plain DSL is indeed what they would enforce.

4. Contain the mapping information with source DSL. This association enables us attach the lower-level mismatch information back to higher level DSL. Thus the user gets feedback about the conflicts at the same abstraction level he writes the policy.

Our approach of transforming the policy DSL in intermediate representation, for linking various layers as well as supporting multiple DSLs and other reasoners, is not specific for privacy domain only, rather could be generalized for other problem domains. Moreover, we allow a high level DSL on top of *Formula* which is a contribution in the area of improving support for lower level DSL.

We translated the policy DSLs in a reasoning model. In case of a non-conformity (i.e. mismatch of privacy documents) our *Formula* engine does not provide enough besides to extract exact mismatch reason(s). We address this problem by incorporating additional rules in the *domain* to generate more detail knowledge about non-conformity, i.e. mismatch. In case of a mismatch, our tool extracts this additional knowledge. However, this detail information, if not re-mapped with upper level representation, would have no clue to effectively guide the end-user. Our approach, while translating to next lower level, does not loose mapping information. We process the mismatch details, associate the reasons with mapping values, and finally highlight the conflicts at the end-user abstraction.

We believe that the proposed approach, if taken forward, would open up possibilities of future work towards user-controlled privacy. From the overall perspective, there are open research problems which were beyond the scope of the thesis work, but we summarize some issues that future research and product groups may address before our work could contribute to a generally useful privacy technology.

***Downstreaming* Support:** An important aspect of *usage control* (how the data has to be treated by data collector after it is released) for privacy is *downstream usage* i.e. with whom and under which usage control restrictions data can be shared [15]. This is critical in the composed web service (so called *mash-ups*)

scenario where the primary data controller share user's data with other parties. This thesis does not support downstreaming which could be an important extension to address.

Model Finding: We worked for discovering the conflict(s) behind a mismatch and highlighting accordingly. However, the policy writer needs guidance for a solution as well. We highlight conflicts in both side so that the user can get an idea with what values the conflicts need to be solved, but this is not sufficient in many cases, e.g. for missing values where the counterpart value is not specified or where the other party's policy is not visible. Better guidance need to rely on *model finding* for generating a solution (i.e. suggestion to get a match). The reasoner we used, *Formula*, enables model finding by executing partial models [2] [29]. By symbolically executing such models *Formula* can find valid bindings for these variables. Using this support, our approach can be extended to automatically search for alternative solutions and provide further guidance thereby.

Multiple *Facet*s: Users may augment their preferences for other *facet*s, e.g. for Service Level Agreement (SLA) [3]. Considering different facets is critical for an end-to-end privacy management system. For instance, the user may specify her privacy as well as the service quality preference and the system needs find a matching solution considering multiple dimensions. Our work can be extended to address such scenario as *Formula* supports *separation of concern*.

UI Experience Enhancement: User Interface is what finally going to enable the client properly visualizing the matching scenario so that she can make well-informed decision about sharing her personal data. However, UI designers need enough information from the reasoner to enhance the usability. One of the intension behind this thesis is to generate as detail information as possible behind a matching scenario. This way, our work also lays a foundation for further UI research.

A demonstration of the prototype implementation is given in appendix B.

[2] A model is called partial if it contains unbound but maybe restricted variables.
[3] SLA is a common way for establishing business contracts between two parties

Acknowledgements

In delivering this thesis, I am fortunate enough to have been assisted by many individuals whom I wish to acknowledge here. First and foremost, I am heartily thankful to Prof. Dr. Ulrike Meyer and Dr. Massimo Felici who kindly accepted me as an external master thesis student.

I owe my sincere gratitude to my ever smiling supervisor in Microsoft, Dr. Laurent Bussard. He has been consistently supportive and caring about this work from very beginning to very end. His research expertise and valuable ideas helped me overcome various challenges for this work. Alongside, I express my deepest gratitude to my other supervisor in Microsoft, Dr. Ulrich Pinsdorf, for pointing me towards relevant literature, reviewing preliminary drafts and above all, always placing encouraging remarks. This work would not reach to this extent without their support.

I received valuable feedback from the PrimeLife project partners. I am also thankful to all my colleagues at European Microsoft Innovation Center who provided valuable opinions on the direction of this work.

A

Glossary

User: An individual (or group of individuals acting as a single entity) on whose behalf a service is accessed and for which personal data exists.

Service Provider: The person or organization that offers information, products, or services from a Web site, collects information, and is responsible for the representations made in a practice statement.

Data Controller: The Data Controller means the entity which alone or jointly with others determines the purposes and means of the processing of personal data. The processing of personal data may be carried out by a Data Processor acting on behalf of the Data Controller. In most scenarios, the Data Controller is the Service Provider.

Data Subject: The Data Subject is the person whose personal data are collected, held or processed by the Data Controller. In most scenarios, the Data Subject is the User.

User Preference: A set of rules that determines what action(s) a user agent will take or allow when involved in an interaction or negotiation with a service. Users' preferences should reflect their attitudes towards the use and disclosure of their personal information.

User Agent: A program that acts on a Data Subject's behalf. The agent may act on preferences (rules) for a broad range of purposes, such as content filtering, trust decisions, or privacy.

Personal Data: Data privacy issues can arise in response to information from a wide range of sources, such as: Health-care records, criminal justice investigations and proceedings, financial institutions and transactions, biological traits, geographic records and ethnicity.

Personally Identifiable Information (PII): Personally Identifiable Information (PII) refers to information that can be used to uniquely identify, contact, or

locate a single person or can be used with other sources to uniquely identify a single individual. PII is a subset of Personal Data.

Service Policy: Data practices of a service. The main content of a privacy policy includes which information the server stores, use of the collected information, how long information is stored, whether and how the user can access the stored information.

Downstream Data Controller: Downstream usage (a PrimeLife Terminology [15]) means the situation when a Data Controller wants to make a Data Subject's personal data available to third parties, so-called downstream Data Controllers. For example, a travel service mash-up may want to forward the Data Subject's driver license information to a car rental service to book a car, or a hospital may want to make patients' medical records available to its researchers.

Sticky privacy policy: An agreement between Data Subject and Data Controller on the handling of personal data collected from the Data Subject. This is also a PrimeLife Terminology [25]. Sticky policies (as well as privacy preferences and privacy policies) defines how data can be handled. Different aspects are defined:

Authorizations

- *Usage:* what the Data Controller can do with collected data (e.g. use them for a specific purpose).
- *Downstream sharing:* under which conditions data can be shared with another Data Controller.

Obligations what the Data Controller must do.

PrimeLife Policy Language (PPL): PPL is an extension of XACML. For more details, see [25].

B

Demonstration Walkthrough

In this section, we are going to walk through our prototype. We begin by loading a default project which includes the policy DSLs, *MGrammar* to parse those DSLs, XSLT documents to translate to target language and the *Formula Domain* file.

B.1 Selecting Active Settings

The system, as described in section 4 allows creating different settings (according to which domain we choose for our system). Figure B.1 shows the menu to load a combination of different settings or a *project*.

In addition to specifying domain knowledge (i.e. DSL parsing grammar, Transformation rules and Reasoning domain), the project also contains predefined templates of preference and policy. However, the user can edit the DSL or even start writing from scratch in the policy editor.

B.2 Translation

We provide transformation of the DSL to an intermediate representation, i.e. XAML. This XAML is further translated to PPL and to lower-level Formula Model. MGrammar supports translating the DSL to structured MGraph (see figure B.2). This MGraph is then automatically transformed to XAML. With the help of XSLT, we further translate the XAML to PPL and Formula (for an illustrative view, see figure B.3).

Figure B.1 Selecting Active Settings for a customized system

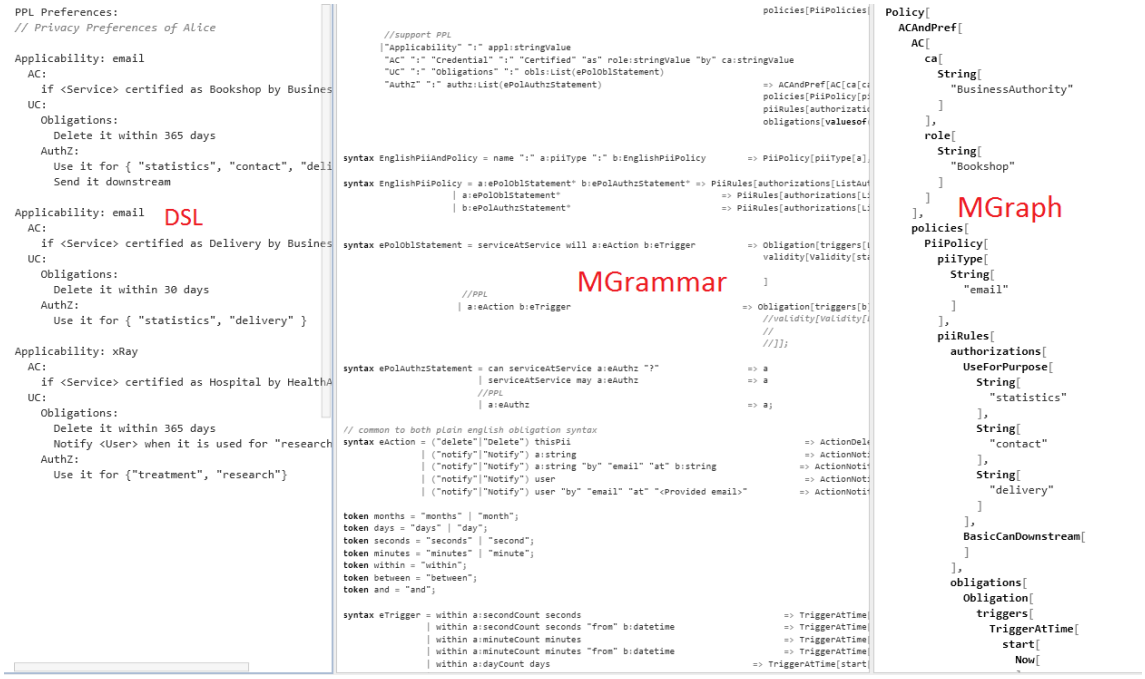

Figure B.2 Parsing DSL: Intellipad view

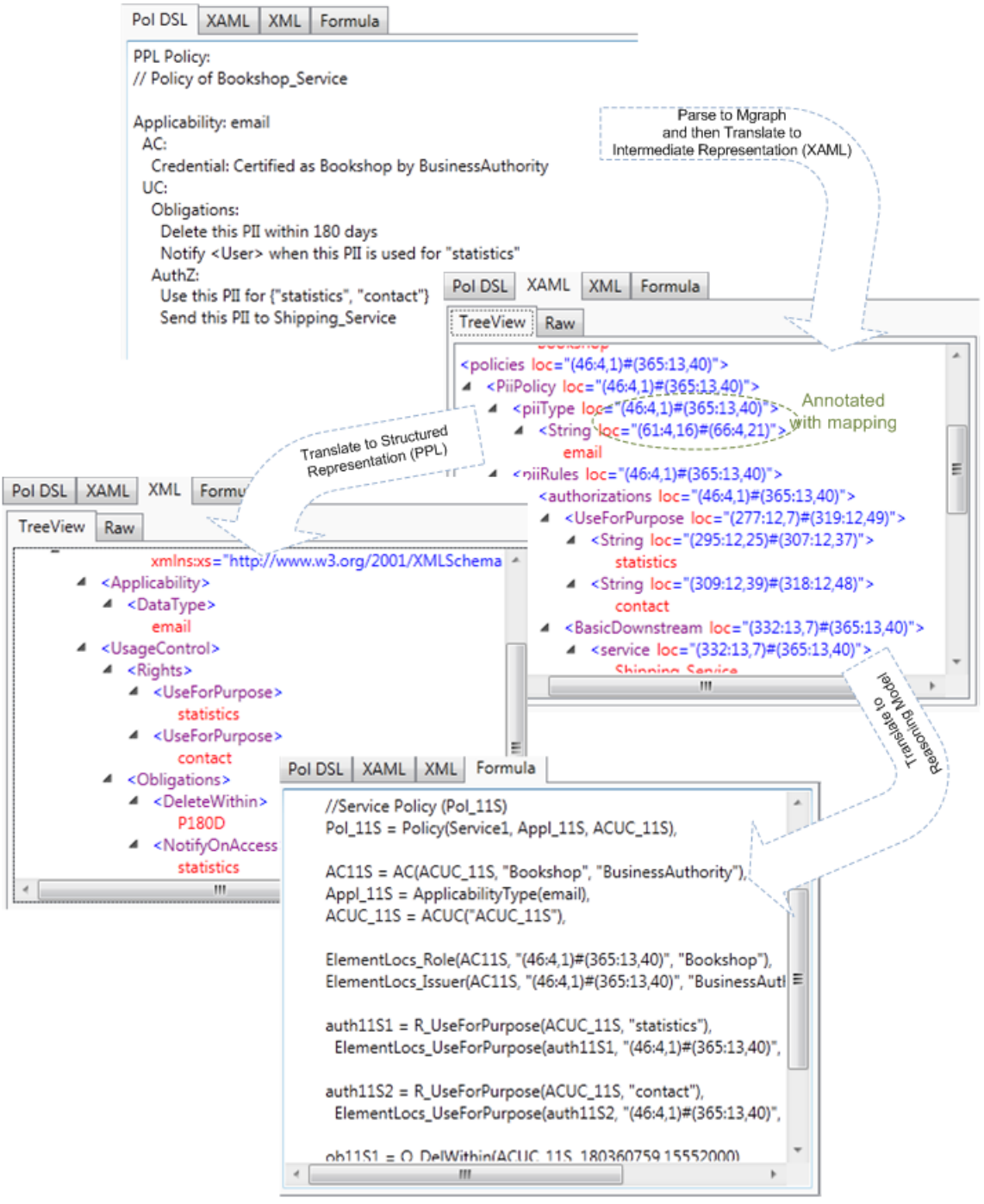

Figure B.3 Translation of Policy DSL to other formats

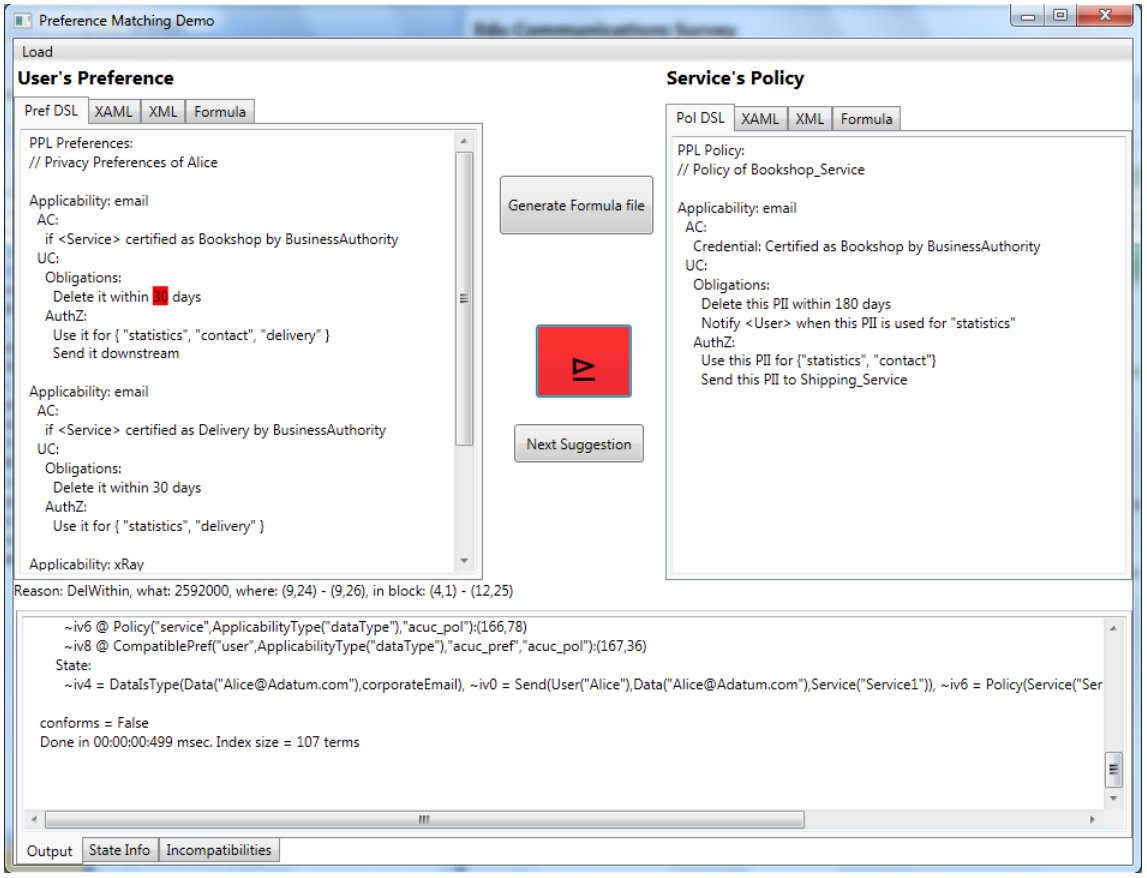

Figure B.4 Highlight mismatch reason

B.3 Matching and Suggestion

If user's preference does not match with service's policy, this tool automatically guides us to a suggestion (highlighting a set of mismatch reasons) which if taken, would lead to a match. The user may look for another suggestion (i.e. another set of mismatch reasons to edit) and does it pressing 'Next Suggestion' button.

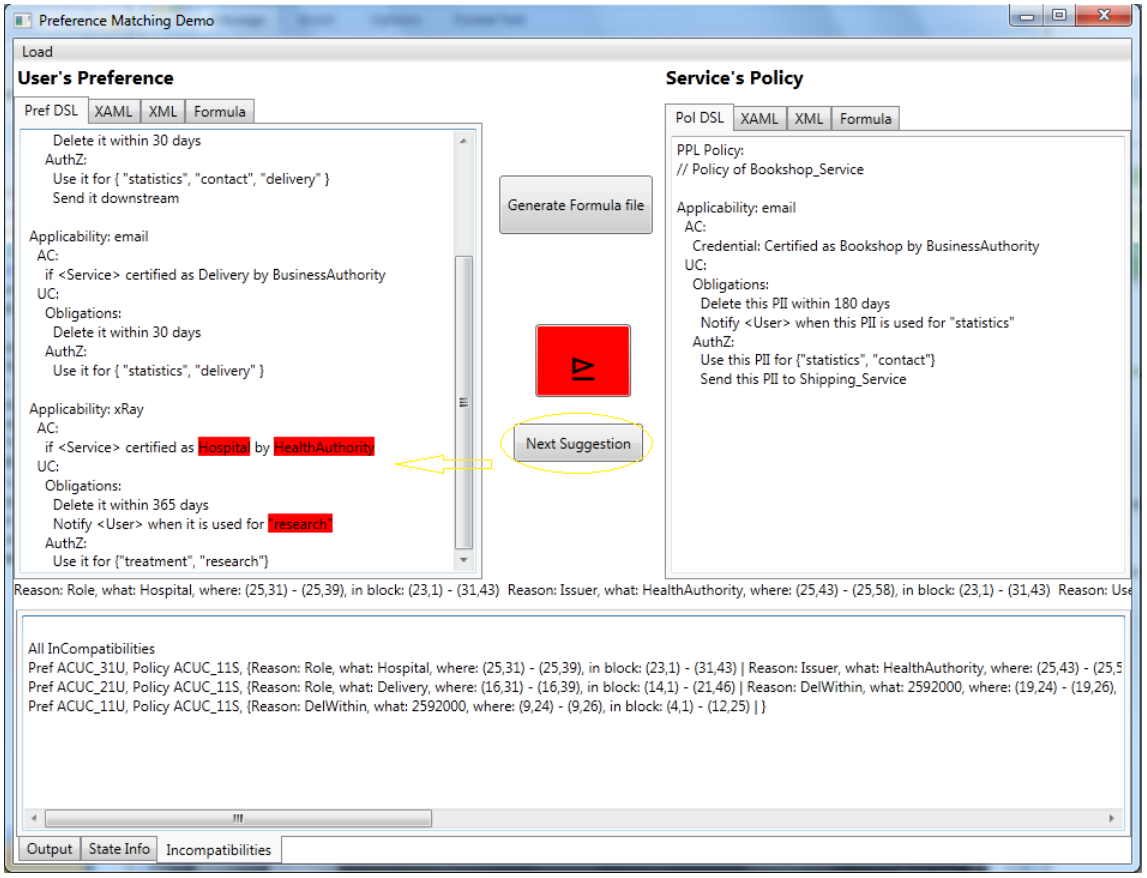

Figure B.5 Highlight mismatch: further suggestion

References

[1] P3P guiding principles. W3C NOTE, 21 July 1998. Online: http://www.w3.org/TR/NOTE-P3P10-principles.

[2] Common Language Infrastructure (CLI). Standard ECMA-335. 4th edition, June 2006.

[3] Mark S. Ackerman and Lorrie Cranor. Privacy critics: UI components to safeguard users' privacy. In *CHI '99: CHI '99 extended abstracts on Human factors in computing systems*, pages 258–259, New York, NY, USA, 1999. ACM.

[4] Mark S. Ackerman, Lorrie Faith Cranor, and Joseph Reagle. Privacy in e-commerce: examining user scenarios and privacy preferences. In *EC '99: Proceedings of the 1st ACM conference on Electronic commerce*, pages 1–8, New York, NY, USA, 1999. ACM.

[5] Mark S. Ackerman and Scott D. Mainwaring. *Privacy Issues and Human-Computer Interaction*. O'Reilly Press, Cambridge, MA, 2005.

[6] Muhammad Ali, Laurent Bussard, and Ulrich Pinsdorf. Obligation Language and Framework to Enable Privacy-Aware SOA. In *DPM/SETOP*, pages 18–32, 2009.

[7] Anne H. Anderson. A comparison of two privacy policy languages: EPAL and XACML. In *SWS '06: Proceedings of the 3rd ACM workshop on Secure web services*, pages 53–60, New York, NY, USA, 2006. ACM.

[8] Claudio A. Ardagna, Laurent Bussard, Sabrina De Capitani Di, Gregory Neven, Stefano Paraboschi, Eros Pedrini, Stefan Preiss, Dave Raggett, Pierangela Samarati, Slim Trabelsi, and Mario Verdicchio. PrimeLife Policy Language.

[9] Paul Ashley, Satoshi Hada, Günter Karjoth, Calvin Powers, and Matthias Schunter. Enterprise Privacy Authorization Language (EPAL 1.2). Technical report, IBM Research Report.

[10] Michael Backes, Günter Karjoth, Walid Bagga, and Matthias Schunter. Efficient comparison of enterprise privacy policies. In *SAC '04: Proceedings of the 2004 ACM symposium on Applied computing*, pages 375–382, New York, NY, USA, 2004. ACM.

[11] Moritz Becker, Cedric Fournet, and Andrew Gordon. Design and Semantics of a Decentralized Authorization Language. In *CSF '07: Proceedings of the 20th IEEE Computer Security Foundations Symposium*, pages 3–15, Washington, DC, USA, 2007. IEEE Computer Society.

[12] Bettina Berendt, Oliver Günther, and Sarah Spiekermann. Privacy in e-commerce: stated preferences vs. actual behavior. *Commun. ACM*, 48(4):101–106, 2005.

[13] E. Boritz and Won Gyun No. A Gap in Perceived Importance of Privacy Policies between Individuals and Companies. pages 181 –192, aug. 2009.

[14] Carolyn A. Brodie, Clare-Marie Karat, and John Karat. An empirical study of natural language parsing of privacy policy rules using the SPARCLE policy workbench. In *SOUPS '06: Proceedings of the second symposium on Usable privacy and security*, pages 8–19, New York, NY, USA, 2006. ACM.

[15] Laurent Bussard, Gregory Neven, and Franz-Stefan Preiss. Downstream Usage Control. In *IEEE POLICY 2010*, July 2010.

[16] J. Cai, M.Moreno Maza, S.M. Watt, and M. Dunstan. Debugging A High-Level Language via a Unified Interpreter and Compiler Runtime Enviroment. In *Proceedings of EACA, Santander*, pages 119–124. Universidad de Cantabria, July 2004.

[17] David W. Chadwick and Stijn F. Lievens. Enforcing "sticky" security policies throughout a distributed application. In *Proceedings of the 2008 workshop on Middleware security table of contents*, pages 1–6, New York, NY, USA, 2008. ACM.

[18] Lorrie Cranor, Marc Langheinrich, Massimo Marchiori, and Joseph Reagle. The Platform for Privacy Preferences 1.0 (P3P1.0) Specification. Technical report, W3C Recommendation, 16 April 2002.

[19] Lorrie Faith Cranor. Putting it together: Internet privacy: a public concern. *netWorker*, 2(3):13–18, 1998.

[20] Lorrie Faith Cranor, Manjula Arjula, and Praveen Guduru. Use of a P3P user agent by early adopters. In *WPES '02: Proceedings of the 2002 ACM workshop on Privacy in the Electronic Society*, pages 1–10, New York, NY, USA, 2002. ACM.

[21] Lorrie Faith Cranor and Paul Resnick. Protocols for automated negotiations with buyer anonymity and seller reputations. *Netnomics*, 2(1):1–23, 2000.

[22] Julia B. Earp and David Baumer. Innovative web use to learn about consumer behavior and online privacy. *Commun. ACM*, 46(4):81–83, 2003.

[23] K El-Khatib. A privacy negotiation protocol for web services. pages 85–92, Halifax, Nova Scotia, Canada.

[24] Jeanne Ferrante. High level language debugging with a compiler. In *SIGSOFT '83: Proceedings of the symposium on High-level debugging*, pages 123–129, New York, NY, USA, 1983. ACM.

[25] Simone Fischer-Hübner and Harald Zwingelberg. UI prototypes: Policy administration and presentation – version 2. PrimeLife Deliverable D4.3.2, PrimeLife Consortium, June 29 2010.

[26] K. Ghazinour, M. Majedi, and K. Barker. A Model for Privacy Policy Visualization. volume 2, pages 335 –340, jul. 2009.

[27] Almut Herzog. Usable Security Policies for Runtime Environments. Linköping Studies in Science and Technology. Dissertation No. 1075., May 2007.

[28] Donna L. Hoffman, Thomas P. Novak, and Marcos Peralta. Building consumer trust online. *Commun. ACM*, 42(4):80–85, 1999.

[29] Ethan K. Jackson, Wolfram Schulte, and Janos Sztipanovits. The Power of Rich Syntax for Model-based Development. MSR technical report, Microsoft Research, 2008.

[30] Maritza Johnson, John Karat, Clare-Marie Karat, and Keith Grueneberg. Optimizing a policy authoring framework for security and privacy policies. In *SOUPS '10: Proceedings of the Sixth Symposium on Usable Privacy and Security*, pages 1–9, New York, NY, USA, 2010. ACM.

[31] John Karat, Clare-Marie Karat, Carolyn Brodie, and Jinjuan Feng. Privacy in information technology: designing to enable privacy policy management in organizations. *Int. J. Hum.-Comput. Stud.*, 63(1-2):153–174, 2005.

[32] Patrick Gage Kelley, Joanna Bresee, Lorrie Faith Cranor, and Robert W. Reeder. A "nutrition label" for privacy. In *SOUPS '09: Proceedings of the 5th Symposium on Usable Privacy and Security*, pages 1–12, New York, NY, USA, 2009. ACM.

[33] Massimo Marchiori Lorrie Cranor, Marc Langheinrich. A P3P Preference Exchange Language 1.0 (APPEL1.0). Technical report, W3C Working Draft, 15 April 2002.

[34] Marjan Mernik, Jan Heering, and Anthony M. Sloane. When and how to develop domain-specific languages. *ACM Comput. Surv.*, 37(4):316–344, 2005.

[35] Alexander Malkis Moritz Y. Becker and Laurent Bussard. S4P: A Generic Language for Specifying Privacy Preferences and Policies. Technical report, Microsoft Research, April 2010.

[36] V. Patel and R. Juric. Internet users and online privacy: a study assessing whether internet users' privacy is adequately protected. pages 193 – 200 vol.1, jun. 2001.

[37] Sören Preibusch. Implementing Privacy Negotiations in E-Commerce. Discussion Papers of DIW Berlin 526, DIW Berlin, German Institute for Economic Research, 2005.

[38] Robert W. Reeder, Clare-Marie Karat, John Karat, and Carolyn Brodie. Usability challenges in security and privacy policy-authoring interfaces. In *INTERACT'07: Proceedings of the 11th IFIP TC 13 international conference on Human-computer interaction*, pages 141–155, Berlin, Heidelberg, 2007. Springer-Verlag.

[39] Warren Teitelman. The Interlisp Reference Manual. Technical report, revised 1978.

[40] Bibi van den Berg and Ronald Leenes. Privacy Enables Communities. PrimeLife Deliverable D1.2.1, PrimeLife Consortium, April 23 2010.

[41] Kami Vaniea, Clare-Marie Karat, Joshua B. Gross, John Karat, and Carolyn Brodie. Evaluating assistance of natural language policy authoring. In *SOUPS '08: Proceedings of the 4th symposium on Usable privacy and security*, pages 65–73, New York, NY, USA, 2008. ACM.

[42] Anton Vedder. Privacy, een conceptuele articulatie. *Filosofie & praktijk*, vol.30 (2009) nr.5 p.7-19, 2009.

[43] Alma Whitten and J. D. Tygar. Why johnny can't encrypt: a usability evaluation of PGP 5.0. In *SSYM'99: Proceedings of the 8th conference on USENIX Security Symposium*, pages 14–14, Berkeley, CA, USA, 1999. USENIX Association.

[44] Shawn Wildermuth. Textual Domain Specific Languages for Developers. Online, February 2009. Last Updated: February 2010 (for 2009 PDC CTP).

[45] Hui Wu. Grammar-driven generation of domain-specific language testing tools. In *OOPSLA '05: Companion to the 20th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications*, pages 210–211, New York, NY, USA, 2005. ACM.

[46] Hui Wu, Jeff Gray, and Marjan Mernik. Debugging Domain-Specific Languages in Eclipse.

[47] Hui Wu, Jeff Gray, and Marjan Mernik. Demonstration of a Domain-Specific Language Debugging Framework.

[48] Hui Wu, Jeff Gray, Suman Roychoudhury, and Marjan Mernik. Weaving a debugging aspect into domain-specific language grammars. In *SAC '05: Proceedings of the 2005 ACM symposium on Applied computing*, pages 1370–1374, New York, NY, USA, 2005. ACM.

[49] George Yee and Larry Korba. Bilateral E-services Negotiation Under Uncertainty. In *SAINT '03: Proceedings of the 2003 Symposium on Applications and the Internet*, page 352, Washington, DC, USA, 2003. IEEE Computer Society.

[50] G.O.M. Yee. An Automatic Privacy Policy Agreement Checker for E-services. pages 307 –315, March 2009.