Extended Privacy Definition Tool

Martin Kähmer and Maike Gilliot
{kaehmer, gilliot}@iig.uni-freiburg.de

Abstract: Eliciting non-functional security requirements within a company was one of the major aspects of the SIKOSA project[1]. Scenarios, such as that of METRO presented in this paper, show how besides a company's internal requirements, customers' preferences also play an important role. However, conflicts between specific customers' privacy policies and those of a company need to be detected and dealt with. To this end we present a policy language that is able to tackle this comparison problem and two tools: An editor tool allowing users to specify their policies in a user-friendly way and a monitoring tool to evaluate und enforce the policies at runtime.

1 Introduction

Personalized services are often viewed as the panacea of e-commerce [SSA06]. User profiles, such as click streams logging which sites users access, are used to generate a profile of the interests of the users. The decisive advantage of such services lies in the opportunity of entering a one-to-one relationship in order to achieve more effective customer loyalty. Service tailoring is thereby no longer limited to e-commerce. In Germany, the METRO-Group is developing the "Future Store", where shopping trolleys are fitted with personal shopping assistants, i.e. computers connected to the store's information system. Services, such as recommender systems, are personalized by means of customer cards [KA06]. However, personalization involves intensive collection and usage of personal-related data. If customers want to benefit from such services, enforcing privacy by minimizing data disclosure is no longer possible. Customers need means to control the usage of their data [PHB06].

In this paper, we present the **Ex**tended **P**rivacy **D**efinition **T**ool (ExPDT) as a means for companies to comply not only with regulatory and business requirements, but also with customers' privacy preferences. In Section 2, we classify our ExPDT solution consisting of a policy language and corresponding tools. The language itself is described in Section 3, and the editor tool and the monitor tool are presented in Section 4. The closing section gives an outlook on further work.

[1] SIKOSA: Sichere Kollaborative Softwareentwicklung und Anwendung [WKK+07], in collaboration with the Uni of Heidelberg, Uni of Hohenheim, and ETH Zürich, funded by MWK of the Land Baden-Württemberg

2 Tackling the privacy problem

Privacy and security for enterprise information systems is about ensuring that business processes are executed as expected and operations such as data accesses are in accordance with a prescribed or agreed on set of norms, such as laws, regulations, and decisions. Solutions to achieve this can be broken down to two main approaches according to the time of application. One approach is called retrospective reporting, where traditional audits usually done through manual checks based on comprehensive logs and reports of the last period of time are used to show policy conformance [Acc08].

The other, more recent approach is often called security by-design, exhibiting a more preventive focus. Non-functional privacy and security requirements are captured and subsequently propagated into the enterprise applications. We propose a model of different layers with respect to abstraction and potential for automation (cf. Figure 1). Since laws only describe what has to be done in general, these regulations have to be interpreted to obtain control objectives for the particular business domain of a company. Although formulated by experts, these control objectives are still in natural language. For IT systems, they need to be interpreted once more and mapped to the particular services, components, and employees of the company. Policies are a set of formalized rules specifying precisely for each unit what is allowed or mandatory and what is prohibited. Such policies serve as input for security monitors, enforcing them on the lower system level. A high degree of automation is only possible within the policy and the monitor layer, where ExPDT is situated, as this is the first level providing laws in a machine-readable format.

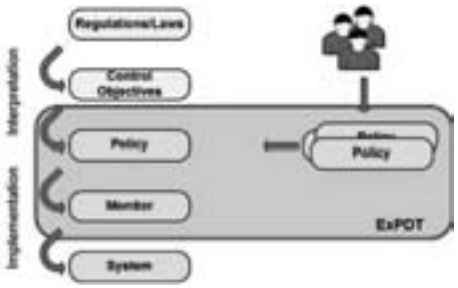

Figure 1: ExPDT within the layer model

2.1 Policy and monitor requirements

For a policy language, it is not merely essential to feature sufficient expressiveness based on a wide range of compassing modalities like permissions, prohibitions and orders as well as on context inclusion based on a fine grained vocabulary [HPSW06, BAKK05]. It is also necessary for a policy language to allow for modular specification and policy comparison so that every single requirement can be addressed and combined with a valid policy for deployment and, since requirements can stem from different regulations and privacy agreements, inconsistencies or contradictions are taken into consideration. Conflicting rules result in operational risk and should be detected and as far as possible solved. Not just regulatory objectives that could be realized in one central policy have to be enforced by a company. Its customers need to be able to control the collection and usage of their own personal data by formulating their own privacy and security preferences [Bun83, PHB06].

To enforce such expressive languages, monitors not only have to cope with conditions, i.e. "traditional" access control, but also control orders and obligations. As those types of rules are generally not enforceable, the monitor has to provide other means to control the fulfillment on system level.

2.2 Related Work

Tackling privacy by means of policies is not new. The World Wide Web Consortium (W3C) developed the Platform for Privacy Preferences (P3P) to express privacy policies in a machine-readable format and its counterpart, the P3P exchange language APPEL, to express customers' preferences [W3C06, CLM05]. Both lack conditions, obligations, and any kind of enforceability and are thus drastically reduced in their usage control capability. For internal policies, the eXtensible Access Control Markup Language XACML [Mos05] was designed by the OASIS consortium as open standard to specify expressive policies that can be interpreted and enforced by a security monitor. XACML even provides policy combination tools to support distributed policies, although it is not suitable for comparing policies, because the intersection of two general policies is not defined. The WS-Policy framework [WSP07] for web services provides a general purpose model and syntax to describe and communicate policies of web services, which consists of sets of different kinds of assertions, e.g. for security, privacy or reliability. Although allowing for optional assertions, this flexibility cannot be guided by sanctions or penalties. While XACML and WS-Policy can be used for privacy related policies, IBM's Enterprise Privacy Authorization Language (EPAL) is dedicated to this task [AHK$^+$03]. It accounts for the further usage of accessed data objects by supporting obligation elements in its policy rules and exhibits a fine grained vocabulary as well as monitor integration. The Novel Algebraic Privacy Specification (NAPS) framework enhances EPAL on a logical level to an algebra additionally allowing for modular specification of policies and adds a concept of sanctions to allow for flexible rule adherence [RS06]. However, common to all of these policy languages is the lack of adequate operators for comparing and analyzing policies.

3 ExPDT – Expressing Privacy Policies

The Extended Privacy Definition Tool (ExPDT) language allows users to specify declarative privacy and security policies over specific domain knowledge using OWL-DL, a computational complete and decidable subset of the OWL Web Ontology Language [MvH04] corresponding to Description Logic. The ExPDT language is used to describe permissions, prohibitions, and orders that have to be adhered to if certain contextual conditions are met or some obligations have to be fulfilled. As ExPDT is geared towards dynamic environments, it deals with incomplete context information and also includes sanctions that can be imposed. Based on the algebraic framework NAPS, it inherits semantics and combination operators allowing for a modular specification of policies. A distinguishing feature of the language is the difference operator for policy comparison. For deployment

of the language, an editor tool was developed. Additionally, a tool for interpreting and enforcing ExPDT policies is presented in Section 4.

For the presentation of the ExPDT language in the following chapters, we introduce three logical layers: the language layer, the domain layer and the instance layer. At the bottom, the language layer establishes the basis by providing fixed vocabulary for the specification of language itself, just like the grammar of a natural language. Based upon that, the domain layer fills up vocabulary by defining the instances of assets, actuators and environment. In contrast to the language layer, the specifications on the domain layer have to be consistent with the current scenario. Therefore, they are subject to occasional adaptations. A common understanding of privacy preferences is not possible, until language and domain are commonly defined. On the third layer, concrete policy instances both of the companies as well as of the customers can be defined, exchanged and agreed upon.

3.1 Language specification layer

A language is made from its syntax and semantics. Hereby, syntax is the definition of all words allowed to be used in the language as well as their set up, in particular the definition of a rule and its parts. The semantics describes the meaning behind the syntax. For a policy, the semantics is given by its evaluation function that provides the results for a particular policy query.

3.1.1 Syntax

Although the ExPDT language features a representation in OWL, its syntax is presented in a more space-saving way on the basis of the simplified OWL class diagram with the inheritance and selected properties of the OWL classes shown in Figure 2. Short examples of the actual OWL syntax are given in Sections 3.2 and 3.3 for domain and instance layer.

A *policy* consists of a prioritized list of rules and a default ruling in the case where no rule applies. A *rule* itself is comprised of one or more possibly negated guards constraining the scope of this rule from users, actions, data and purpose, a number of conditions and the ruling that subsequently delivers the decision of this rule. Hence, a generic rule has the following form: $[(\mathrm{user}, \mathrm{data}, \mathrm{action}, \mathrm{purpose}), \mathrm{conditions}, \mathrm{ruling}]$.

For intuitive specification of the scenario on the domain layer later on, the element instances of a *guard* are partially ordered in hierarchical structures allowing for grouping of instances and the formulation of policies rules applying to entire sub-hierarchies, e.g. to all users of a particular department or all the data belonging to contact information. Thereby, each of them has his own structure: customers, employees and services are combined in the *user* structure, sensitive data items are described in *data*, possible actions on these items in *action* and the possible intentions of actions in question are structured in *purpose*.

Regulations often depend on context information, e.g. permitting data access only if the customer is not under age or the legal guardian has given his consent. For the inclusion of such constraints, ExPDT reverts to a 3-valued, many-sorted condition logic. A condition

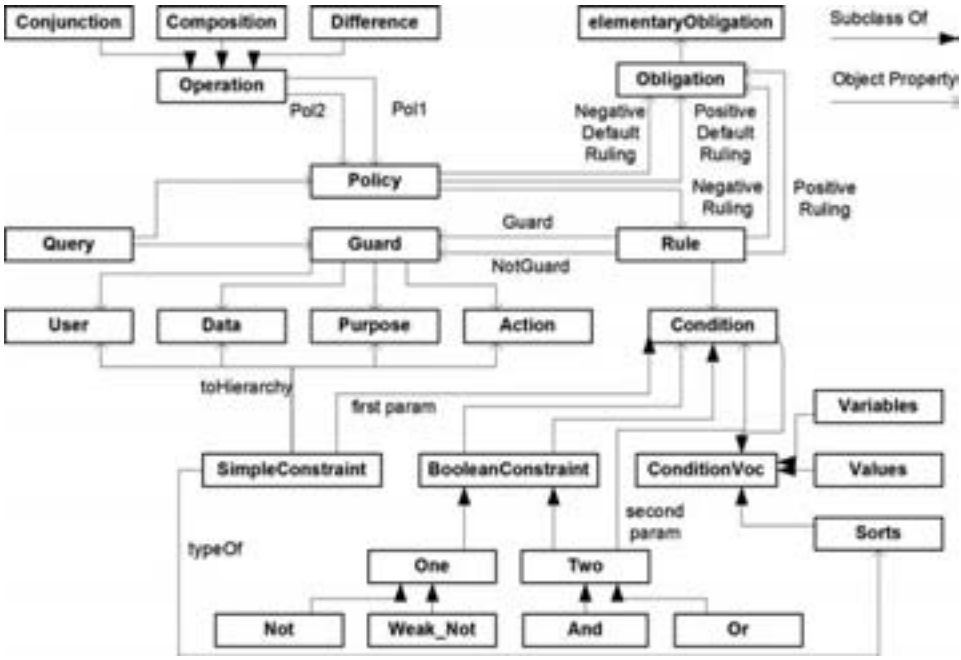

Figure 2: Syntax of ExPDT policy language as class diagram

is a formula of this condition logic defined over the condition *vocabulary* and its interpretation functions. A vocabulary consists of the final set of *sorts* (i.e. variable types) each with a final set of *variables*. The set of non-logical symbols of *simple constraints* includes relations, the set of logical symbols the operators *and*, *or*, *not*, *weak not*, 0, 1 and u as undefined. The single-valued operators not and weak not have only a first parameter, the others a second additional one. Formulas and terms of the condition logic are recursively defined as usual as in the predicate logic free of quantors. The semantics of a formula is given as in the 3-values Łukasiewicz L3 logic. The undefined value is advantageous to an environment of dynamic character, such as stores with continuously changing customers and modified or switched services, in that it supports the rule evaluation even with incomplete context information as will be shown in Section 3.1.2.

A policy rule not only regulates the actions on data items, but can impose *obligations*, such as "notify customer" or "delete data within one day". In contrast to many other policy languages, ExPDT does not consider obligations as pure black box instructions, but has an underlying obligation model of a half lattice above the power set of the *elementary obligations* $\tilde{O}$, subset as relation, conjunction as aggregation, with maximum element top $\top$ as the empty obligation and the minimal element bottom $\bot$ as the impossible obligation. Imposing the obligation $\top$ means that the action of the guard can be carried out without further undertaking, imposing $\bot$ that an action may not be carried out. Eliminations of contradicting elementary obligation combinations, such as "delete data within a week"

Modality	Obligations	Sanctions	Ruling
Permission			$(\top, \top)$
	o^+		$(o^+, \top)$
Prohibition			$(\bot, \top)$
		o^-	$(\bot, o^-)$
Order			$(\top, \bot)$
	o^+		$(o^+, \bot)$
		o^-	$(\top, o^-)$
	o^+	o^-	(o^+, o^-)
Error			$(\bot, \bot)$

Table 1: ExPDT codes modalities into ruling.

and "keep data for a year" at the same time, can be achieved by excluding from the lattice all those obligation sets containing problematic obligations.

The *ruling* of an ExPDT rule and the default ruling of the overall policy are specified by a tuple of obligation (postiveObligation, negativeObligation). Since ExPDT policies make statements about users performing an action on some data for a particular purpose, the policy query is accordingly also a tuple of user, action, data, and purpose, too.

3.1.2 Semantics

While the specification of policies, queries and rulings were part of the syntax, the evaluation function of a query resulting in a ruling for a given policy defines the semantics of the ExPDT policy language. Firstly, the semantics of a single ruling is explained, followed by the description of the evaluation function.

Ruling of a rule The required authorization and order rule modalities presented can both be expressed in the ExPDT language, as shown in Table 1. Actions can therefore not only be permitted but also forbidden. It is also possible to compulsorily regulate the execution of actions. In addition to the conditions obligations can also be imposed on the user. These are actions that have to be performed in future. The ExPDT language also allows the users a certain degree of freedom in rule compliance. While this always applies in the case of a permit – if an action is allowed, one does not necessarily have to use this right – users can decide for themselves whether they adhere to a prohibition or a command. If they do not, sanctions in the form of additional obligations can be specified in an ExPDT rule. If these sanctions correspond however to the impossible obligation , adherence becomes necessary for the users. The various rule modalities are mapped in ExPDT via the tuple of the ruling. Here are some examples:

- Permission: A retailer can access the customer number. Ruling: ($\top$, $\top$)
- Permission with obligation: A retailer can access the customer's shopping list but not secretly. Ruling: (notify, $\top$)

- Prohibition with sanction: A retailer may not access the shopping list, which is achieved by imposing the impossible obligation. Disregarding this prohibition, he must inform the customer and pay a fine as sanction. Ruling: ($\perp$, payFine)
- Compulsory command: The security administrator must in any event classify the data requested according to its sensitivity. The sanction according to the impossible obligation makes adherence to this order indispensable. Ruling: ($\top$, $\perp$)

Evaluation of a policy The semantics of the policy language is determined through the evaluation function $eval_\alpha(\mathcal{P}, q)$ for a query q regarding a particular policy $\mathcal{P}$ and current assignment α of the contextual condition variables. Roughly, the function searches through the list of policy rules until a rule is matched by the query. Matching means that all elements of the rule guard are either equal to the user, action, data, and purpose of the query or stand higher up in their corresponding hierarchy. Additionally, the condition of the rule must not evaluate to false using the current variable assignment. Thus, queries are not restricted to minimal elements of the guard hierarchies and allow for scenarios, where, for example, a basic policy company policy referring to departments is only composed with a department policy making concrete statements about individuals. Although the particular user Bob may not be mentioned in the company policy, its rules still apply to him. The complete evaluation algorithm works as follows:

1. Initialize the ruling r_p with ($\top$, $\top$) and preset evaluation status v to default.
2. Evaluate rules one by one according to their priority.
    a. If the rule's guard is matched by the query and its condition evaluates to 1, return the conjunction of the rule's ruling and hitherto accumulated r_p as policy ruling and an evaluation status v of final.
    b. If the query is matched by a rule's guard and its condition evaluates to u, add rule's ruling to r_p, set the status v to applicable and proceed with the next rule.
3. If the status v is applicable, than return r_p as ruling and that status.
4. If the status is still default, no rule has matched and the default ruling is returned together with the status v default.

The case of incomplete context information resulting in an undefined condition value for a rule is taken into account by accumulating the ruling of such a rule with a possibly previous found ruling, i.e. conjunct both the positive obligations and the negative obligations, and proceeding with the evaluation. Hence, it is ensured that the evaluated ruling is possibly too restrictive due to the additional obligations, but never too weak.

Combination of policies The extensive dragging along of the evaluation status v with its distinction of final, applicable or default ruling allows ExPDT the definition combination operators despite the stub-behavior of the policies. The stub-behavior corresponds to the

intention of the default ruling is to ensure a safe ruling until another rule matches, therefore the refinement of a default ruling with an applicable or final one should be possible in case of a policy combination. In ExPDT, two combination operators are defined: the conjunction $\mathcal{P}_1 \wedge \mathcal{P}_2$ thereby evaluates $\mathcal{P}_1$ and $\mathcal{P}_2$ with equal priority, while the composition $\mathcal{P}_1||\mathcal{P}_2$ gives $\mathcal{P}_1$ higher priority for the evaluation. For more detailed combination tables of the rulings, generating algorithms, and algebraic laws see [Rau04].

Comparison of policies In related literature, for the comparison of two guidelines one often finds the equivalence where both guidelines always supply the same results and the refinement which examines a guideline as to whether it is more restrictive or specific than another. In practice, these tests are however only suitable to a certain extent for users, if they can only determine with them whether their preferences are fulfilled by a service. How should the users behave, however, if the policy of a service does not correspond to that of their own, therefore being not equivalent or more restrictive? They will not reject a utilization of the service in each case. It can even be their wish, depending on the current situation, to lower their data protection demands in favor of the utilization.

In such situations, the users must be able to quickly survey and estimate how far the service guideline deviates from their own preferences or the service's previous ruling. In dynamic environments, in particular, where the users are faced with many different services and their respective individual policies, this task can no longer be manually accomplished. In order to alleviate the user's personal decision for or against service utilization, the difference operator for two guidelines is defined in the following. This operator reduces the regulation of the policy to become effective to those rules describing situations that are of interest to the users for their assessments, namely to those allowing additional actions or at least actions on weaker conditions or obligations and so supply more generous results.

Difference: Given two policies $\mathcal{P}_1$ and $\mathcal{P}_2$ over compatible vocabulary, the difference $\mathcal{P}_2 - \mathcal{P}_1$ is a mapping from $\mathcal{P} \in \mathcal{P}$ to a list of rules R that covers exactly those queries q and assignments α of conditional variables that result in a less restrictive ruling for $\mathcal{P}_2$, so $(r_i, v_i) = eval_\alpha(\mathcal{P}_i, q)$ for $i = 1, 2$ and $r_1 \not\leq r_2$. For these, the difference rule list results in the same decisions as $\mathcal{P}_2$.

This rule list describes the functional difference of both policies, so they are compared independent of their possible evaluation status; the stub behavior of the policies is not taken into consideration. This is particularly significant if policy $\mathcal{P}_1$ is to be replaced by a different policy $\mathcal{P}_2$, for example if a customer discards his own preferences $\mathcal{P}_1$ and releases his personal data under the service's policy $\mathcal{P}_2$. Then it is irrelevant whether an action is forbidden owing to the standard ruling of $\mathcal{P}_1$, but this standard ruling is refined with a permit. It is only important here that this action is subsequently permitted. However, if policies $\mathcal{P}_1$ and $\mathcal{P}_2$ are intended to be connected afterwards, the difference should consider the stub-behavior of $\mathcal{P}_1$. For instance, the $\mathcal{P}_1$ default ruling can be replaced by an arbitrary non-default ruling of $\mathcal{P}_2$, which would provide a more specific result without the need to get the users' attention – as long as they are aware of this stub-behavior. In fact, the former mentioned equivalence and refinement of two policies can be computed by means of the

difference: if $\mathcal{P}_2 - \mathcal{P}_1$ results in an empty list, $\mathcal{P}_2$ describe less restrictive situations and $\mathcal{P}_2$ is a functional refinement of $\mathcal{P}_1$. If the difference of switched policies results in an empty rule list as well, $\mathcal{P}_1$ and $\mathcal{P}_2$ are functionally equivalent.

An initial implementation can take place here by way of a brute force approach. The decisions for all possible enquiries and all possible allocations of the environment parameter must simply be calculated for both $\mathcal{P}_1$ and $\mathcal{P}_2$ policies. A rule with the scope of the query (i.e. corresponding guard) and the conditions and ruling of the rule appropriate in each case of $\mathcal{P}_2$ is included in the rule list $\mathcal{P}_2 - \mathcal{P}_1$, if the ruling r_1 has other or lesser obligations than r_2. This brute force approach tests all possible combinations of enquiries and parameter allocations so that its complexity grows exponentially with the vocabulary used.

Algorithm 1 difference(R_1, R_2): walking through the rule sets R_1 and R_2

```
 1: PolicyDIFF := ∅
 2: for i:= max(R2) downto min(R2) do
 3:    for j:= max(R1) downto min(R1) do
 4:       RuleDIFF := rulecomp(R1[j], R2[i])
 5:       if RuleDIFF ≠ ∅ then
 6:          for all rd in RuleDIFF do
 7:             if guardOf(rd) ∧ ¬guardOf(R1[j]) ≠ ∅ then
 8:                PolicyDIFF += differenz( (R1[j − 1] downto R1[max]), rd )
 9:             else
10:                PolicyDIFF += rd
11:             end if
12:          end for
13:          next i
14:       end if
15:    end for
16: end for
17: return PolicyDIFF
```

Therefore, a more efficient approach is presented in this paper. As outlined by Algorithm 1, the rules of both policies are looped through according to their priority, so that each rule of $\mathcal{P}_2$ is compared with all rules of $\mathcal{P}_1$. According to the guard logic, guards can contain disjunctions, conjunctions and negations of guards. Therefore, guards cannot be compared using their top-elements alone, but by the set of hierarchy elements described by them, defined by the closure. For an element h, the closure $\overline{h}$ consists of h itself and all elements lower in the hierarchy. The guard operators $\vee$, $\wedge$, and $\neg$ can be mapped accordingly $\cup$, $\cap$ and $\not\leq$. If such a comparison detects only equal or more restrictive situations with bigger scope or weaker conditions and obligations, the looping is continued with the remaining rules of $\mathcal{P}_1$. If there are, however, such situations and if they are not captured by a following $\mathcal{P}_1$-rule with lower priority (cf. recursive call), they are formally captured by a new rule that is appended to the difference result. Then, the looping is discontinued for the current $\mathcal{P}_2$ rule and starts with the next rule anew. If all $\mathcal{P}_2$ rules are examined, the construction of the difference terminates. However, before Algorithm 1 can start, the policies have to be preprocessed by upgrading the default rulings and normalizing $\mathcal{P}_2$.

For the construction of the functional difference, a distinction between normal rules and default ruling is not necessary. Hence, the standard rulings of both policies are upgraded to

Algorithm 2 DiffNorm(R): Normalization of rule list

$R^N := \emptyset$
for i:= $max(R)$ downto $min(R)$ {rules in order of priority} **do**
  (g, c, r) := split i {splitting of a rule in its parts}
  M := $\emptyset$
  for all $f \in \{0,1\}^{|l_g|}$ **do**
    g_f:=g
    c_f:=c
    for $n = 1$ to $|l_g|$ **do**
      if $f(n) = 0$ **then**
        $c_f := c_f \wedge l_c$
      end if
      if $f(n) = 1$ **then**
        $g_f := g_f \wedge l_g$
      end if
      $M := M \cup (g_f, c_f, r)$
    end for
  end for
  $R^N := R^N \cup M$
  $l_c := l_c \cup \neg c$
  $l_g := l_g \cup \neg g$
end for
return R^N

normal rules by appending rules to the list with the root elements of the guard hierarchies, without conditions, and with default ruling as rule ruling: (user_{root}, data_{root}, action_{root}, $\mathrm{purpose}_{root}$), 1, (default ruling). If there is more than one root for a hierarchy, append a rule for each of them. These rules match all possible queries by design, so that the default ruling is not triggered anymore and can be disregarded for the difference construction.

Since the evaluation function of ExPDT considers the policy rules as a prioritized list with dependencies, the rules of $\mathcal{P}_2$ cannot be individually compared; $\mathcal{P}_2$ has to be normalized first. Following Algorithm 2, for each rule all the situations matched by rules with higher priority are explicitly excluded from its guard and conditions. For $\mathcal{P}_1$, this normalization is not necessary because the recursive call already copes with the dependencies.

For the comparison of two rules in Algorithm 3, it has to be determined whether there is an overlap between the two scopes. If they do not overlap, the comparison stops. Otherwise, the difference between these two rules is examined by the following case differentiation:

- $r_1 \not\geqslant r_2$: The ruling of $rule_2$ is less restrictive or different. Independent of the conditions, $rule_2$ is appended to the rule difference and returned.
- Otherwise: The ruling r_2 is stricter or equal. Hence, up to two rules have to be appended to the rule difference:
  - For queries matching $guard_2$ but not $guard_1$, this stricter or other ruling is in any case new, so that a rule with these queries as guard, conditions of c_2, and ruling r_2 is appended.
  - For queries also matching $guard_2$ as $guard_1$ (i.e. the disjunction of their closures), the r_1 is less restrictive, but a less restrictive condition c_2 can neces-

Algorithm 3 rulecomp($rule_1, rule_2$): comparison of two rules

```
1:  (g1, c1, r1) := split rule1
2:  (g2, c2, r2) := split rule2
3:  RuleDIFF := ∅
4:  if g2 ∧ g1 ≠ ∅ then
5:    if r1 ⋡ r2 then
6:      RuleDIFF += (g2, c2, r2)
7:    else
8:      if g2 ∧ ¬g1 ≠ ∅ then
9:        RuleDIFF += ((g2 ∧ ¬g1), c2, r2)
10:     end if
11:     if c2 ↛ c1 then
12:       RuleDIFF += ((g2 ∧ g1), (c2 ∧ ¬c1), r2)
13:     end if
14:   end if
15: end if
16: return RuleDIFF
```

sitate another difference rule. This new rule should only describe the new situations, so that its condition is $c_2 \wedge \neg c_1$.

Example: $c_1 = (\leq 18)$, $c_2 = (\leq 18 \vee GuardianOK)$
$c_2 \wedge \neg c_1 = (\not\leq 18 \wedge GuardianOK)$

For the last case, it is essential to not only evaluate conditions but to also compare the ability to satisfy two given conditions. It must be determined whether one rule restricts its applicability with more strict or equivalent conditions than another rule or features a greater application space with additional context situations. This yields in the examination whether a condition c_1 satisfies another c_2, i.e. if c_1 is satisfied, then c_2 is also satisfied, or if c_1 results in the undefined state u, c_2 is also undefined or even satisfied. For independency of the current situation, this has to hold for all possible variable assignments. This examination is, however, NP-complete over the number of variables of the vocabulary considered, so that no efficient algorithm is to be expected for the general case. Nevertheless, in order to be able to compare the conditions, the examination is reduced to a satisfy relation similar to [BKBS04], which is at least correct, i.e. if two conditions c_1 and c_2 are contained in the relation, then the above-mentioned satisfaction holds true.

Satisfy relation: Given a conditional vocabulary $\mathfrak{Voc}$, the satisfy relation is the relation $\rightarrow_{\mathfrak{Voc}} \subseteq C(\mathfrak{Voc}) \times C(\mathfrak{Voc})$. The relation is correct if for all conditions $c_1, c_2 \in C(\mathfrak{Voc})$ for all possible assignments α holds (in infix notation):

$$c_1, c_2 \in \rightarrow_{\mathfrak{Voc}} :\Rightarrow \Big[eval_\alpha(c_1) = true \Rightarrow eval_\alpha(c_2) = true\Big] \vee \Big[eval_\alpha(c_1) = \emptyset \Rightarrow \big(eval_\alpha(c_2) = true \vee \emptyset\big)\Big]$$

If the opposite direction also holds, the satisfy relation is complete. A correct satisfy relation can often be constructed via the symbolic evaluation by all pairs of atomic formulas

```
<owl:Class rdf:about="DATA:shoppingList">
  <rdfs:subClassOf rdf:about="DATA"/>
</owl:Class>

<owl:Class rdf:about="DATA:Prescription">
  <rdfs:subClassOf
    rdf:resource="DATA:shoppingList"/>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty
        rdf:resource="COND:contains"/>
      <owl:someValuesFrom
        rdf:resource="DATA:drugs"/>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>

<Prescription rdf:ID="DATA:shopAtDrugstore"/>
```

Listing 1: Definition of guard elements

```
<owl:Class rdf:about="OBL:Delete">
  <rdfs:subClassOf
    rdf:resource="OBL:ELEOBLIG"/>
  <owl:disjointWith>
    <owl:Class rdf:about="OBL:keep"/>
  </owl:disjointWith>
</owl:Class>

<Obligation rdf:about="OBL:delete_Notify">
  <rdfs:subClassOf
    rdf:resource="OBL:keep"/>
  <rdfs:subClassOf
    rdf:resource="OBL:notify"/>
</Obligation>
```

Listing 2: Definition of obligations

with known semantics satisfy dependency, but also via the comparison on a pure syntactic level of their interpretation functions of the same sort. Such a correct satisfy relation is mostly adequate for practical application, even if it is not complete. For if two conditions are mutually dependent and this dependency is unknown to the users and therefore not included in the relation, then such conditions should be independently treated in order to meet the users' expectation. At worst, the differential result hereby increases by an additional rule, however without changing its evaluation.

3.2 Domain layer

Based upon the language layer, the vocabulary is filled up by defining concrete instances of assets, actuators, conditions, and obligations and by categorizing them accordingly on the domain layer. These specifications should always be consistent with the current scenario and, therefore, need to be adapted in case of environmental changes. ExPDT uses an ontology specified in OWL-DL, as it supports not only for the representing of domain specific knowledge, but also for the automated interpretation and reasoning.

Listing 1 gives an example of the data hierarchy specification. Here, a shopping list is a subclass of the root DATA. In turn, a medical prescription is a subclass of a shopping list, and its instances are dynamically assigned by a property restriction: all shopping lists that contain at least one drug to buy are considered as a prescription. Hence, customers can formulate rules for this particular kind of shopping list, taking care not to provide its contents to normal salespersons, but only to the druggist of the shop. Examples of obligation are given in Listing 2. The obligation "delete data after usage" is coded as an elementary obligation that, of course, is incompatible with the obligation to keep the data afterwards. Obligations used in rulings are either instances of single or multiple elementary obligations.

3.3 Policy layer

On the policy layer, instances of policy can be formulized by combining the building blocks of the domain layer stuck together with the elements of the language layer. Listing 3 shows a privacy policy containing only two rules. The first rule allows a druggist to access a particular prescription for t he purpose of giving further information, e.g. the compatibleness of some substances. The second rule allows all persons working in sales to read shopping lists, if the customer has consented, the data is deleted afterwards and the customer is notified about this event. Queries for all other situations are not matched by the rules, but by the restrictive default ruling of the policy that prohibits everything not already covered by the rules. As in the policy instance, the assignment of the conditional variables to the guard hierarchies – the customer has not given consent for actions involving his data – is part of the policy layer, but the actual variable values have to be provided by the monitor.

```
<POL:Policy rdf:ID="MyPolicy">
  <POL:PolicyHasRules>
    <rdf:Seq rdf:ID="MyPolicyRules">
      <rdf:li>
        <POL:Rule rdf:ID="Rule1">
          <POL:RuleHasUser rdf:resource="USER:druggist"/>
          <POL:RuleHasAction rdf:resource="ACTION:read"/>
          <POL:RuleHasData rdf:resource="DATA:shopAtDrugstore"/>
          <POL:RuleHasPurpose rdf:resource="PURP:information"/>
          <POL:RuleHasPosObligation rdf:resource="OBL:top"/>
          <POL:RuleHasNegObligation rdf:resource="OBL:top"/>
        </POL:Rule>
      </rdf:li>
      <rdf:li>
        <POL:Rule rdf:ID="Rule2">
          <POL:RuleHasUser rdf:resource="USER:sales"/>
          <POL:RuleHasAction rdf:resource="ACTION:read"/>
          <POL:RuleHasData rdf:resource="DATA:shoppingList"/>
          <POL:RuleHasPurpose rdf:resource="PURP:advertisment"/>
          <POL:RuleHasCondition rdf:resource="COND:hasConsented"/>
          <POL:RuleHasPosObligation rdf:resource="OBL:deleteAndNotify"/>
          <POL:RuleHasNegObligation rdf:resource="OBL:top"/>
        </POL:Rule>
      </rdf:li>
    </rdf:Seq>
  </POL:PolicyHasRules>
  <POL:PolicyHasPosDefaultRuling rdf:resource="OBL:bottom"/>
  <POL:PolicyHasNegDefaultRuling rdf:resource="OBL:top"/>
</POL:Policy>

<ASS:Data rdf:about="DATA:Customer">
  <ASS:gaveConsent rdf:resource="Consent:no"/>
</ASS:Data>
```

Listing 3: Example of policy instance with two rules

4 ExPDT – The tools

The previous section described the semantic and syntactic aspects of the policy language itself. In the following we focus on the tools making the language usable. We will first

Figure 3: Displaying policies and specifying operations using the ExPDT policy editor

present an editor tool as graphical user interface for users to define their privacy preferences, followed by the components of a monitor tool for the evaluation and the enforcement of policies on the system layer.

4.1 Editor tool

Basically, any generic OWL editor can be used as graphical user interface for ExPDT, as it enables the user to handle the various OWL-DL files, such as domain knowledge and policy instances, to visualize and to edit the contained classes and their properties. However, the limited usability of those editors prevents its deployment for laymen. Hence, we are developing a lightweight editor with reduced capabilities that hides the loquaciousness of OWL-DL. As seen in Figure 3, this editor not only allows for the editing of ExPDT policies, but can also be used for requesting policy operations, such as comparison or query evaluation, and subsequently displaying of the results returned by the monitor tool.

4.2 Monitor tool

An access control monitor prevents unacceptable behavior by running alongside an untrusted program and intercepting all actions that would lead to undesirable states [HMS06]. Access control rules without obligations and sanctions are always enforceable, as well as some obligations. Starting from the architecture of the ExPDT monitor in Figure 4, we present the main components of the monitor and their interplay. Whenever the Policy Enforcement Point receives a request, it is forwarded to the Policy Decision Point, which selects the “relevant” rules from the policy and evaluates it by dint of the Policy Information Point. The access decision is then returned to the Policy Enforcement Point, which grants or denies the access.

Policy Decision Point: The guard's elements user, action, data, and purpose are mapped to their corresponding hierarchies in order to determine the relevant policy rules for a given query. As the rules are ordered by priority, only the first matching rule is considered and evaluated. Further rules matching the guard only come into play, if the conditions statement returns the value "undefined". ExPDT handles an undefined condition values as true, but imposes on the access further obligations by accumulating the obligations of following matching rules.

Figure 4: Components of ExPDT monitor tool

Policy Information Point: The Policy Information Point assigns the values to the conditions. Conditions either relate to former accesses or events or to the system environment. Some conditions are static: Once the value of the condition "over 18" is true, it stays true. In contrast, the value of the condition "has consented" has to be evaluated at each request, as the user may have in between revoked his consent. To this end, a mapping from the language ontology onto real system events is required. For example, a message reporting a click on a button in the GUI has to be interpreted as "user has consented". The mapping of the received events onto the ontology is done mainly manually. Approaches to handle this translation automatically are presented for example in [GMP06].

Policy Enforcement Point: The enforcement component has to impose the access decision of the evaluation component onto the system. This is easy for policy rules without obligations or sanctions. They are enforceable, as an access control monitor is able to evaluate each request and to prohibit policy violating accesses. Enforcement of obligations and sanctions in general is not possible, as both prescribe actions a subject has to perform some time in the future. An online monitor has thus no possibility to enforce that the obliged action is accomplished within the given timeframe. However, the monitor is able to supervise the events and to report violations or fulfillment *after* the period allotted. An online monitor can evaluate the obligation "user has to be informed about data usage within 2 days" by checking after the two days if the user notification has been sent. However, obligations expressing a prohibition can be enforced if they can be transferred into preconditions. Examples are the Chinese Wall policy rules, where a user reading some confidential data about company A is in future not allowed to read confidential data of competing company B. This obligation can be expressed as precondition to further access requests and is thus enforceable.

For the evaluation of ExPDT policies, a prototype was implemented in Java. This tool provides for the functionality of a policy management point by offering the difference operator for policy comparison and the combination operators. (cf. Section 3.1.2)For acting as a policy decision point, it can evaluate policy requests given an assignment of environment variables. The prototype is based on the Jena OWL framework [Hew07] that not only allows for parsing the OWL files, but also provides an API to reason about the con-

tained information. As the internal reasoner of Jena is still quite limited for OWL-DL, the external Pellet reasoner [Cla08] is attached. It evaluates queries in description logic (DL) on the given knowledge base and relieves ExPDT of the computation of class inheritances and memberships, and the reconstruction of the full guard hierarchies.

5 Summary and conclusion and further steps

Due to technologies like RFID, personalized services are not longer limited to the e-commerce world. Motivated by an example of the retailer METRO, we show that enterprises can provide personalized services to their clients based on the personal data collected during their shop visits. Customers accept the release of personal data if they are still able to control the usage of their data. Thus, enterprises have - beside their internal security requirements - also to respect their customers' privacy preferences and demonstrate compliance to their policies. This requires an expressive policy language and a mechanism to prove compliance to the customer and regulatory bodies. This paper focuses on the policy language. We present the extended privacy definition tool ExPDT expressing privacy preferences for access and usage of personal data, focusing on the comparison and merging features of the language. We further describe the implementation of the ExPDT enforcement monitor describing which ExPDT rules can be enforced, and which cannot. The proposed tools contribute to the compliance by design approach.

For now, ExPDT enables differences between policies to be detected. How these differences can be resolved has not yet been considered. The next step, therefore, is a negotiation protocol. The goal is not a fully automated procedure, but a tool to assist the negotiation process step by step. Enforcement of obligations is a open research issue. We are currently investigating how obligations can be enforced by rewriting business process. A second approach to the enforcement of obligations uses heuristics in order to determine at runtime process executions that will probably lead to obligation violation.

References

[Acc08] Rafael Accorsi. Privacy Audits to Complement the Notion of Control for Identity Management. In de Leeuw, Fischer-Hübner, Tseng, and Borking, editors, *Policies and Research in Identity Management*, volume 261. Springer, 2008.

[AHK+03] Paul Ashley, Satoshi Hada, Günter Karjoth, Calvin Powers, and Matthias Schunter. Enterprise Privacy Authorization Language (EPAL 1.2). Submission to W3C, 2003.

[BAKK05] Travis D. Breaux, Annie I. Antón, Clare-Marie Karat, and John Karat. Enforceability vs. Accountability in Electronic Policies. Technical Report TR-2005-47, North Carolina State University Computer Science, 2005.

[BKBS04] Michael Backes, Günter Karjoth, Walid Bagga, and Matthias Schunter. Efficient comparison of enterprise privacy policies. In *Proc. of 2004 ACM Symposium on Applied Computing*, pages 375 – 382, 2004.

[Bun83] Bundesverfassungsgericht. Volkszählungsurteil. In *Entscheidungen des Bundesverfassungsgerichts*, volume 65. 1983. Urteil 15.12.83; Az.: 1 BvR 209/83; NJW 84, 419.

[Cla08] Clark & Parsia. Pellet OWL-Reasoner. `http://pellet.owldl.com`, 2008.

[CLM05] Lorrie Faith Cranor, Marc Langheinrich, and Massimo Marchiori. A P3P Preference Exchange Language 1.0 (APPEL). Technical report, W3C, 2005.

[GMP06] Christopher Giblin, Samuel Müller, and Birgit Pfitzmann. From Regulatory Policies to Event Monitoring Rules: Towards Model-Driven Compliance Automation. Technical report, IBM Research Zürich, 2006.

[Hew07] Hewlett-Packard Development Company. Jena: A Semantic Web Framework for Java. `http://jena.sourceforge.net/`, 2007.

[HMS06] Kevin W. Halmen, Greg Morrisett, and Fred B. Schneider. Computability Classes for Enforcement Mechanisms. *ACM Transactions on Programming Languages and Systems*, 28:1, 2006.

[HPSW06] Manuel Hilty, Alexander Pretschner, Christian Schaefer, and Thomas Walter. Enforcement for Usage Control - A System Model and an Obligation Language for Distributed Usage Control. Technical Report I-ST-18, DoCoMo Euro-Labs Internal, 2006.

[KA06] Martin Kähmer and Rafael Accorsi. Kundenkarten in hochdynamischen Systemen. In *Proc. of KiVS NETSEC 2007*, 2006.

[Mos05] Tim Moses. eXtensible Access Control Markup Language (XACML) Version 2.0. Technical report, OASIS, 2005.

[MvH04] Deborah L. McGuinness and Frank van Harmelen. OWL Web Ontology Language – Overview. W3C Recommendation `http://www.w3.org/TR/owl-features/`, 2004.

[PHB06] Alexander Pretschner, Manuel Hilty, and David Basin. Distributed Usage Control. *Communications of the ACM*, 49(9):39–44, September 2006.

[Rau04] Dominik Raub. Algebraische Spezifikation von Privacy Policies. Master's thesis, Universität Karlsruhe (TH), 2004.

[RS06] Dominik Raub and Rainer Steinwandt. An Algebra for Enterprise Privacy Policies Closed Under Composition and Conjunction. In *Proc. of Int. Conf. on Emerging Trends in Information and Communication Security (ETRICS)*, pages 132 – 146, 2006.

[SSA06] Stefan Sackmann, Jens Strücker, and Rafael Accorsi. Personalization in Privacy-Aware Highly Dynamic Systems. *Communications of the ACM*, 49(9):32–38, 2006.

[W3C06] W3C. Platform for Privacy Preferences (P3P) Project. `http://www.w3.org/P3P/`, 2006.

[WKK+07] Daniel Weiß, Jörn Kaack, Stefan Kirn, Maike Gilliot, Lutz Lowis, and Günter Müller at al. Die SIKOSA-Methodik: Unterstützung der industriellen Softwareproduktion durch methodische integrierte Softwareentwicklungsprozesse. *Wirtschaftsinformatik*, 49(3):188 – 198, 2007.

[WSP07] Web Services Policy 1.2 - Framework (WS-Policy). `http://www.w3.org/Submission/WS-Policy/`, 2007.