Science of Computer Programming 77 (2012) 1198–1209

Contents lists available at SciVerse ScienceDirect

Science of Computer Programming

journal homepage: www.elsevier.com/locate/scico

Refinement checking for privacy policies[☆]

Nikolaos Papanikolaou[a,*], Sadie Creese[b], Michael Goldsmith[b]

[a] *Cloud and Security Lab, HP Labs, Bristol, United Kingdom*

[b] *International Digital Laboratory, WMG, University of Warwick, United Kingdom*

ARTICLE INFO

Article history:
Available online 14 July 2011

Keywords:
Privacy policy
Policy refinement
CSP
Model checking

ABSTRACT

This paper presents a framework for analysis and comparison of privacy policies expressed in P3P (Platform for Privacy Preferences). In contrast to existing approaches to policy analysis, which focus on demonstrations of equality or equivalence of policies, our approach makes it possible to check for *refinement* between policies. We automatically generate a CSP model from a P3P policy, which represents the policy's intended semantics; using the FDR model checker, we then perform various tests (using process refinement) to determine (a) whether a policy is internally consistent, and (b) whether a given policy refines another by permitting similar data collection, processing and sharing practices. Our approach allows for the detection of subtle differences between practices prescribed by different privacy policies, the comparison of relative levels of privacy offered by different policies, and captures the semantics of policies intended in the original P3P standard. The systematic translation of policies to CSP provides a formal means of reasoning about websites' privacy policies, and therefore the practices of various enterprises with regards to personal data.

© 2011 Elsevier B.V. All rights reserved.

1. Introduction

In our digital age more and more of society's benefits and services are provided online; for access to these, we find ourselves constantly being asked for personal information. Meanwhile, enterprise information systems are amassing ever growing digital piles of data pertaining to individuals, data which is often regarded as having huge commercial value. Databases can be indexed, searched and sorted as dictated by enterprise needs, although data processing practices are always expected to comply with current law and data protection agreements. Unfortunately there seems to be a big gap between customers' expectation of privacy and the actual privacy provided through common business practice.

Website users are often unaware of and unconcerned by any privacy policy that may be in effect during an online transaction. Where such a policy is expressly provided, it often takes the form of a lengthy legal document which is presented to the user for him or her to accept or reject completely. Interestingly, there is no provision for the user to opt out of individual clauses of such a policy, and he or she is expected to grant 'blanket consent' for all uses of their data set out in the policy document. Some websites provide the ability to opt in or opt out of being contacted for marketing purposes, but the controls available are not very fine grained and only exist to satisfy the minimum requirements of the law (e.g. the Data Protection Act in the UK). There is an increasing body of work on developing better privacy controls for users; see for example the PRIME [1], PrimeLife [2] and EnCoRe [3] research projects.

☆ This work was carried out under the auspices of the EnCoRe project ("Ensuring Consent and Revocation"), sponsored by the UK Technology Strategy Board, the Engineering and Physical Sciences Research Council, and the Economic and Social Research Council.

* Corresponding address: Cloud and Security Lab, Hewlett Packard Laboratories, Long Down Avenue, Stoke Gifford, Bristol BS34 8QZ, United Kingdom.
E-mail addresses: nick.papanikolaou@hp.com, N.Papanikolaou@warwick.ac.uk (N. Papanikolaou), S.Creese@warwick.ac.uk (S. Creese), M.H.Goldsmith@warwick.ac.uk (M. Goldsmith).

0167-6423/$ – see front matter © 2011 Elsevier B.V. All rights reserved.
doi:10.1016/j.scico.2011.07.009

The Privacy Preferences Platform Project (P3P) [4] is focused on developing machine readable XML for expressing websites' privacy policies and users' privacy preferences; this is intended to enable the use of privacy-aware browsers and to allow websites to collect and process information they may require in a fashion that respects user privacy. The policy languages developed within P3P so far are lacking a formal semantics and hence are prone to inconsistency and ambiguity [5]. XACML [6] is a language for expressing role-based access control, which has been augmented with a "profile" for expressing privacy policies [7], but until recently lacked a formal semantics (the latest version, XACML 2.0, has a draft formal semantics). The lack of a widely accepted semantics for privacy policies is the main source of difficulty in policy comparison.

We are interested in the use of privacy policies as a means to protecting personal data in cyberspace. As we argue below, it may be desirable to compare two related policies, particularly when these policies are expressed using different languages. However, even when two policies are expressed in the same language, it is possible that they will differ syntactically but supposedly express the same intention. If mistakes are made in the comparison of privacy policies this could result in the wrong policy being implemented and exposure to risk.

Policy refinement [8] is a term used to refer to the process of synthesising lower level policies from policies expressing higher level concerns. An enterprise typically has to conform with many different sets of policies and, for any particular context, such as privacy, there will be a *policy hierarchy* that specifies the order and priority of policies which must be enforced. In such a hierarchy one will find policies which are all intended for essentially the same goal, but they differ in the details; a higher-level policy may stipulate, for instance, that adequate protections on customer data must be in place, while a lower-level policy would detail the encryption mechanisms and access controls used to protect such data. The lower-level policy is said to *refine* the higher-level one when it only permits data processing permitted by the higher-level policy. Our focus on refinement as opposed to equality of policies is justified by the need to check that one policy is a valid implementation of another, as might arise in the setting of a supply chain. Consider, for example, the situation where a user shares personal data with an organisation and in doing so stipulates a range of preferences on how that data should be handled. If the data is passed onto a third party it will be necessary to ensure that the policy adopted by the third party offers at least as much privacy as the original organisation.

Our intention in this paper is to connect the concept of policy refinement (particularly in the context of privacy) with the notion of refinement as used in process algebra, by developing a translation of P3P policies into CSP processes which can be directly compared using the FDR model checker[1] [9,10]. Using this translation, one can perform three types of analysis:

- Comparison of two given P3P policies as to the data collection, processing and sharing practices which they each prescribe.
- Comparison of a given P3P policy with an enterprise's higher-level privacy goals (and possibly with customer requirements).
- Consistency checking of a given P3P policy, to prevent ambiguities.

The translation of policies to CSP gives them a uniform representation which makes it convenient to reason about privacy requirements and dependencies between policy rules. With FDR it is then possible to automatically check that overall goals are being met; if they are not, FDR produces counterexamples which can be used to determine sources of errors and inconsistencies.

While we are aware of some existing work on direct comparison of P3P policies (see Section 1.1), we believe that our approach is particularly well-suited to the task of policy comparison, in that it permits a fine grained analysis: rather than just comparing the overall effects of two given policies, we compare the collection, processing and sharing practices of each policy for each data item individually. Furthermore, we take into account the intended semantics of the predefined constants in the P3P standard; most existing policy comparison approaches are syntactic, in that they do not make any effort to capture the dependencies and possible conflicts between the values allowed in policy rules.

It is worth noting here that, as of this writing, the latest (and final) version of P3P is 1.1 and, while this language is still in use for specifying website privacy policies, it has been somewhat superseded by the XACML standard. However, the wide availability of tools such as the IBM P3P Policy Editor [11] and the Privacy Finder search engine [12] make it easy to edit, manage and locate real-world privacy policies, and the language is still widely used. For these reasons, P3P policies are worthy of further study and analysis. Future work is likely to include extending the techniques presented in this paper to other policy languages.

1.1. Related work

Related work includes research on the formalisation and automatic processing of P3P and XACML policies. There is also purely theoretical work on mathematically describing access control and its properties, though there is a lack of formal work on the analysis of privacy.

[1] FDR is a product of Formal Systems (Europe) Ltd (http://www.fsel.com) and the University of Oxford (http://web.comlab.ox.ac.uk/projects/concurrency_tools).

May et al. [13] define 'policy relations' in an analogous way to bisimulation for concurrent processes; their high-level semantic framework is used for comparing P3P policy outcomes. Their approach is mathematically elegant as it avoids comparing specific actions permitted by a policy, and focuses only on the outcomes of applying it; however it corresponds to an equivalence check between policies, rather than a more fine-grained refinement check.

Yu et al. [14] have proposed a relational semantics for P3P, in an effort to give unambiguous meaning to syntactically different expressions of a single policy. Their ideas are complementary to the approach we propose, although they focus only on internal policy consistency. Known ambiguities and problems with P3P policies have been studied by Hogben [5], and these issues inspired both [14] and the present work.

Stamey and Rossi [15] applied Latent Semantic Analysis techniques to interpret and compare sets of legal privacy policies (expressed in natural language, of course). They investigate linguistic patterns arising in the texts of privacy policies, identifying high-frequency words and semantic similarities.

Bryans [16] builds on earlier work by Ryan [17] and develops a CSP model of XACML access control policies. The emphasis in this paper is on applications related to separation of duty constraints and workflow.

Fisler et al. [18] presents the Margrave tool for analysing and reasoning about XACML access control policies; Margrave translates policies into a decision diagram representation which allows a user to pose queries about the policy and its properties.

Zhang et al. [19] describes a tool and associated formalism, RW, for writing specifications of access control policies from which valid XACML can be generated.

Gunter et al. [20] defines a formal privacy system as a mathematical object, namely a labelled transition system with a set of predefined actions that arise in the context of location-aware services.

May et al. [21] extends [20] by incorporating notification and logging operations and by proposing a formal translation to Promela, the input language of the SPIN model checker.

Fournet et al. [22] uses process algebra to express a logic of authorisation, while Walker [23] develops a type system for reasoning about security policies; both approaches demonstrate how authorisation or access control can be formalised.

Becker et al. [24] have designed a declarative language, S4P (or SecPAL4P) for expressing privacy preferences and policies. They provide a formal semantics and proof rules for their language. They do not seem to define a direct mapping from existing formalisms such as P3P or XACML into their language, however.

1.2. *Outline of this paper*

There are five parts to this paper. Section 2 describes P3P policy structure, constants and intended semantics; this leads to a discussion of what it means for a given P3P policy to refine another. Section 3 discusses how we model a P3P policy in CSP, and the tool we have developed for performing the conversion automatically. Then in Sections 4 and 5 we discuss the process refinement checks that we can perform on a policy and on a pair of policies, respectively, using FDR. Finally, Section 6 concludes with a discussion of future work.

2. Understanding P3P policies and their semantics

First it is necessary to pin down exactly what we mean by the term 'P3P policy,' since the W3C Recommendation [4] does define a number of similar terms. A P3P-compatible website that collects and uses personal data will have associated with it a *policy reference file*, which is a listing of all the *policy files* that apply to the different pages in the site. P3P is an instantiation of XML, so both files are expressed in this format, using only the tag names and values defined in the Recommendation. A policy file may generally contain any number of actual policies. For the purposes of this paper we work only with policy files consisting of a single policy, and this assumption is also made by the tool described in Section 3. However, the technique can generally handle any number of policies within a policy file.

2.1. *Structure of an individual policy*

A policy defines a set of rules, or *statements,* that specify which processing and sharing practices are permitted for different types of data that the website may collect. Statements will refer to types of data explicitly referred to in the Recommendation, namely dynamic data, user data, business data and third party data. For each of these types of data the Recommendation defines a schema, and there are a number of predefined constants which are used in actual policies (see Section 2.2), although user-defined types are also allowed. For the needs of this paper we assume that policies only use predefined types. The predefined types are termed 'datagroups'.

A statement includes:

- a `<DATA-GROUP>` element,
- a `<PURPOSE>` element,
- a `<RECIPIENT>` element,
- a `<RETENTION>` element,
- (optionally) a `<CONSEQUENCE>` element,
- (optionally) other extensions.

```
<POLICY xml:lang="en">
    <STATEMENT>
    <PURPOSE><admin/>
             <current/>
             <develop/>
    </PURPOSE>
    <RECIPIENT><ours/></RECIPIENT>
    <RETENTION><stated-purpose/>
    </RETENTION>
    <DATA-GROUP>
    <DATA ref="#dynamic.clickstream"/>
    <DATA ref="#dynamic.http"/>
    </DATA-GROUP>
</STATEMENT>
</POLICY>
```

Fig. 1. Policy A

A statement is intended to express a policy rule that, it is permitted for the website to collect a datum belonging to the given datagroup for the specified purpose, for a period of time defined by <RETENTION>, and for distribution only to those entities defined in the <RECIPIENT> element. We ignore optional elements in this paper.

2.2. Predefined constants

The values that are permitted in the elements of a statement are the following (we do not include the data schema, which defines the possible values for datagroups, for lack of space):

- for <PURPOSE>:
  - *current, admin, develop, tailoring, pseudo-analysis, pseudo-decision, individual-analysis, individual-decision, contact, historical, telemarketing, other-purpose*
- for <RECIPIENT>:
  - *ours, delivery, same, other-recipient, unrelated, public*
- for <RETENTION>:
  - *no-retention, stated-purpose, legal-requirement, business-practices, indefinitely*

While an explanation of each value is given in the Recommendation, it should be noted that the purposes in particular are subject to some degree of interpretation, in that a website owner may choose quite loosely among these values. The different values for <RECIPIENT> describe the nature of an entity with whom data is being shared; the value *ours* applies when data is to be used exclusively by the owner of the website and by third parties who need it for a stated purpose only. The value *delivery* applies when data is to be shared with service providers whose practices are unknown and may differ from those of the site owner. Values *same* and *other-recipient* are used for data that may be shared with third parties whose data practices are either identical or different, respectively, to those of the site owner, and the other values are special cases.

For the values that describe the data retention practices, it is interesting to note that there is no numerical value (such as, for instance, a 30-day limit on the use of data). The values range from no retention at all to indefinite retention.

While the Recommendation does not define any relationships between these predefined values, we believe that the semantics of the language would be significantly less ambiguous if the permitted recipients and retention practices in a given policy depended on the purpose(s) for which data collection is taking place.

2.3. Examples of policies and refinement

In this section we will use two example P3P policies to explain the notion of policy refinement. As we will see, a purely syntactic comparison would treat these policies as being different from one another, esp. in terms of stated purpose; we show that there is a relationship of refinement between the two. The example should make clear why refinement is useful in this context, and more effective as a comparison tool than a notion of equality or equivalence of policies.

Fig. 1 shows a policy, Policy A, consisting of a single statement that applies to two different datagroups, #dynamic.clickstream and #dynamic.http. These data groups refer to the data available on a web site server log about the access requests it has received. In particular,

> "The clickstream element is expected to apply to practically all Web sites. It represents the combination of information typically found in Web server access logs: the IP address or hostname of the user's computer, the URI of the resource requested, the time the request was made, the HTTP method used in the request, the size of the response, and the HTTP status code in the response." [4]

```
<POLICY xml:lang="en">
    <STATEMENT>
    <PURPOSE><admin/>
    </PURPOSE>
    <RECIPIENT><ours/></RECIPIENT>
    <RETENTION><stated-purpose/>
    </RETENTION>
    <DATA-GROUP>
    <DATA ref="#dynamic.clickstream"/>
    <DATA ref="#dynamic.http"/>
    </DATA-GROUP>
</STATEMENT>
</POLICY>
```

Fig. 2. Policy B

Policy A requires that these datagroups be collected for administrative, current, and development purposes, for a period of time determined by the enterprise's stated purpose, and sharing is only permitted with third parties who will use it for a stated purpose (note the meaning of the constant `ours`, which does in fact allow limited sharing).

Policy B in Fig. 2 has a statement that applies to the very same datagroups. It may have been defined at a later stage to Policy A, when the enterprise completed deployment of a new version of its website. How does it compare to Policy A?

If these two policies were compared directly using existing approaches, such as policy relations [13], or by means of a syntactic comparison, they would be found to be substantially different. While the two policies are certainly not equal, since they differ in the number of purposes for which collection is being performed, there is clearly a similarity between the two, in that everything policy B permits is also permitted by policy A. We say that *Policy A is refined by Policy B* and write $A \sqsubseteq B$.

The approach to policy comparison advanced in the following sections attempts to take into consideration meanings and intentions of the predefined constants.

3. Modelling P3P policies in CSP

We have developed tool support[2] for translating a P3P policy into a CSP model that can be checked using FDR. In this section we present the structure of such models and discuss our design choices.

For definitions of the CSP notations, including among others external choice ($P \,\square\, Q$), indexed external choice ($\square_i P_i$), parallel composition ($P \parallel Q$), the predefined *CHAOS* process and more we refer the reader to the standard reference [9]. We assume familiarity with these notations and concepts.

Given a P3P policy file, which contains a single privacy policy, our translator extracts all the different statements, and generates definitions of CSP processes corresponding to each statement. The first (top-level) process corresponding to a P3P statement will be referred to as a *rule process*. To represent the policy as a whole, we will use a single process combining together all rule processes in parallel composition.

3.1. Modelling P3P statements

From the kth P3P statement in a given policy, only the `DATA-GROUP`, `PURPOSE`, `RECIPIENT` and `RETENTION` elements are extracted. The process $RULE_0^{k,dg}$ contains a *collect* event, which corresponds to the collection of an item in datagroup dg, for purposes $P = \{p_n\}$ $(n \geq 1)$ and retention t, with the possibility of sharing to recipients r, and is defined as follows:

$$\begin{aligned} RULE_0^{k,dg} &= collect.dg.p_1.r.t \rightarrow RULE_1^{k,dg} \\ &\quad \square\ collect.dg.p_2.r.t \rightarrow RULE_2^{k,dg} \\ &\quad \vdots \\ &\quad \square\ collect.dg.p_n.r.t \rightarrow RULE_n^{k,dg} \end{aligned}$$

For the specified purposes $p_i \in \{p_1, \ldots, p_n\}$, with $i \leq n$, in the P3P statement, we define $RULE_i^{k,dg}$ as a state in which processing of the data dg for purpose p_i, followed by further collection, further processing for purpose p_i, or sharing with recipient r (through $RULE_{n+1}^{k,dg}$) is permitted.

$$RULE_i^{k,dg} = process.dg.p_i.t \rightarrow \left(RULE_0^{k,dg} \,\square\, RULE_i^{k,dg} \,\square\, RULE_{n+1}^{k,dg} \right)$$

2 This is a Python program, which is available on request.

We define $RULE_{n+1}^{k,dg}$ as a state in which sharing with recipient r is possible, followed by either further collection, further processing, or further sharing:

$$RULE_{n+1}^{k,dg} = share.dg.r \rightarrow \left(RULE_0^{k,dg} \, \Box \left(\Box_i \, RULE_i^{k,dg} \right) \Box \, RULE_{n+1}^{k,dg} \right)$$

In order to handle the retention value specified in P3P statements, we include in our model for each statement a clock, or *counter*. A full description of a P3P statement is given by synchronising the top-level rule process with this counter. At time instant 0, the counter only permits collection events to be triggered. Thus we have the counter process:

$$COUNTINIT(d, P, r, t) = COUNT(val\_of(t), d, P, r, t)$$

The use of a counter is a way of addressing the fact that standard CSP does not allow for time constraints and specifications. If we had chosen to use the formalism of Timed CSP [25] this could have been avoided, but FDR only supports untimed CSP, so this would not be practical for our purposes.

We have taken the convention of assigning the following numerical values to retentions:

$$\begin{aligned} val\_of(no\_retention) &= 1 \\ val\_of(stated\_purpose) &= 2 \\ val\_of(legal\_requirement) &= 5 \\ val\_of(indefinitely) &= 100 \\ val\_of(business\_practices) &= 10 \end{aligned}$$

Note that our choice of numerical values for these constants is based on our interpretation of the P3P standard and has been validated through discussions with legal and regulatory experts. Should the intended interpretation of the constants change or require adjustment, this is easy to implement.

The *COUNT* process specifies when different events are permitted, with the intention that processing and sharing events should only be possible while a collected data item can be retained. Once the time for which a data item is available has lapsed, processing and sharing are no longer possible, and the item should be collected anew. There is a separate instance of *COUNT* for every combination of d, P, r, t. The process counts downwards to ensure a bounded state space in the model, which is necessary for model checking. This is expressed in the following process definition.

$$\begin{aligned} COUNT(cnt, d, P, r, t) = \; & cnt > 0 \; \& \left(\left(\Box_p \, collect.d.p.r.t \rightarrow COUNT(cnt, d, P, r, t) \right) \right. \\ & \qquad \Box \left(\Box_p \, process.d.p \rightarrow COUNT(cnt - 1, d, P, r, t) \right) \\ & \qquad \left. \Box \, share.d.r \rightarrow COUNT(cnt - 1, d, P, r, t) \right) \\ & \Box \; cnt = 0 \; \& \left(\Box_p \, collect.d.p.r.t \rightarrow COUNTINIT(d, P, r, t) \right) \end{aligned}$$

So, a P3P statement for datagroup dg, with purposes $P = \{p_i, i \leq n\}$, recipient r and retention t is fully defined in CSP by a process of the form:

$$RULE^{k,dg} = RULE_0^{k,dg} \, |[\forall p \in P \bullet collect.d.p.r.t, process.d.p, share.d.r]| \, COUNTINIT(d, P, r, t)$$

Example. The definitions given in the previous section are easily understood by way of an example. Suppose we wish to express the statement contained in Policy A (see Fig. 2) in CSP. The statement in Policy A applies to two datagroups, *dynamic_clickstream* and *dynamic_http*. The corresponding CSP code for *dynamic_clickstream* datagroup is shown below, where we have abbreviated $RULE^{1,dynamic\_clickstream}$ to $RULE^a$ for clarity.

$$\begin{aligned} RULE^a = \; & RULE_0^a \quad |[\forall p \in \{admin, current, develop\} \bullet \\ & \qquad collect.dynamic\_clickstream.p.ours.stated\_purpose, \\ & \qquad process.dynamic\_clickstream.p.ours.stated\_purpose, \\ & \qquad share.dynamic\_clickstream.ours]| \\ & COUNTINIT(dynamic\_clickstream, \{admin, current, develop\}, ours, stated\_purpose) \\ RULE_0^a = \; & collect.dynamic\_clickstream.admin.ours.stated\_purpose \rightarrow RULE_1^a \\ & \Box \; collect.dynamic\_clickstream.current.ours.stated\_purpose \rightarrow RULE_2^a \\ & \Box \; collect.dynamic\_clickstream.develop.ours.stated\_purpose \rightarrow RULE_3^a \end{aligned}$$

$$
\begin{aligned}
RULE_1^a &= process.dynamic\_clickstream.admin.ours.stated\_purpose\\
&\qquad \rightarrow (RULE_0^a \,\Box\, RULE_1^a \,\Box\, RULE_4^a)\\
RULE_2^a &= process.dynamic\_clickstream.current.ours.stated\_purpose\\
&\qquad \rightarrow (RULE_0^a \,\Box\, RULE_2^a \,\Box\, RULE_4^a)\\
RULE_3^a &= process.dynamic\_clickstream.develop.ours.stated\_purpose\\
&\qquad \rightarrow (RULE_0^a \,\Box\, RULE_3^a \,\Box\, RULE_4^a)\\
RULE_4^a &= share.dynamic\_clickstream.ours\\
&\qquad \rightarrow (RULE_0^a \,\Box\, RULE_1^a \,\Box\, RULE_2^a \,\Box\, RULE_3^a \,\Box\, RULE_4^a)
\end{aligned}
$$

3.2. Modelling a policy

A policy is defined by the totality of statements it contains, and therefore can be expressed as the parallel composition of all the $RULE^{k,dg}$ processes for all k and all dg:

$$POLICY = \Big|\Big|\Big|_{k,dg} RULE^{k,dg}$$

With this definition in hand we can perform analyses of the kind discussed in the Introduction.

The analyses rely on the notion of refinement as conventionally used in process algebra. We use the symbol $\sqsubseteq$ to denote a relation of refinement between two processes. When there is no subscript this may denote either traces refinement (normally $\sqsubseteq_T$) or failures refinement (normally $\sqsubseteq_F$) depending on the context. We review the definition of traces refinement in Section 5.

4. Consistency checking

As we have noted, the semantics of P3P lack a formal, unambiguous definition, and this has been the source of criticism. Using the CSP model of a P3P policy presented in the previous section, we can perform checks using the FDR model checker that the stipulations of a policy are plausible and compatible with the P3P Recommendation. While the Recommendation lists a number of predefined values for the `PURPOSE`, `RECIPIENT` and `RETENTION` specified in any statement, it does not explicitly describe dependencies and potential conflicts between those values. It is assumed by the authors of the Recommendation that users of the language will make sensible and compatible choices when authoring policies; we feel that consistency checks are not only necessary to ensure syntactic correctness of a policy, but that they can be used to warn against dangerous or implausible policy statements.

For instance, the Recommendation specifies that when *no_retention* is specified for a given datagroup, the data in question should be used only for a single online transaction and must be subsequently destroyed. When this is given as the retention policy, it should by definition preclude the possibility that the website owner will use the data for any other purpose than those corresponding to *tailoring* and *current*. Other predefined values for `PURPOSE` must not be permitted if data is not retained for a longer time, since they all assume that the data being collected will be held for further processing. The P3P Recommendation does not explicitly disallow other purposes for data collection, although this is clearly necessary as the resulting statements would be implausible.

Consider how this type of inconsistency can be prevented in the CSP model. We can define a process $TIMELY(d)$, for each datagroup d, which allows processing only for the *current* and *tailoring* purposes if d has been collected for *no_retention*. To do a consistency check on a given policy $POLICY_A$ we will use a process $CONSISTENCY_1$ which permits only this behaviour for all datagroups.

$$
\begin{aligned}
TIMELY(d) \quad &= collect.d.\_.\_.t \rightarrow\\
&\quad (\text{if } t = no\_retention\\
&\quad \text{then } CHAOS(process.d.current, process.d.tailoring)\\
&\quad \text{else } CHAOS(process.d) \,\triangle\, TIMELY(d))\\
CONSISTENCY_1 &= \Big|\Big|\Big|_{d} \bullet TIMELY(d)
\end{aligned}
$$

The consistency check is performed by testing that (note that we need to ignore *share* events):

$$CONSISTENCY_1 \sqsubseteq_T POLICY_A \backslash \{share\}$$

Note that in this form, the process $CONSISTENCY_1$ does not allow for two different *collect* events to occur prior to processing/sharing. This is just a convention used for the needs of the examples in this paper, but the definition can be generalised.

A similar situation in P3P arises with the predefined values for recipients. The values *delivery*, *other_recipient*, *unrelated* and *public* are all used in cases where the recipient of the data generally follows different data collection practices than the owner of the website with the policy. In the descriptions of most predefined purposes for which data may be collected, it is assumed that a third party will follow the same practices as the website owner. The Recommendation does not explicitly constrain, for which purposes, sharing with third parties that follow different practices should be permitted. It is likely, and definitely in line with customer expectations, that third parties following different data collection practices to the owner of a site should be prevented from obtaining this data with very few exceptions. As a consistency check, it may be sensible to check that a policy which allows sharing of data with such recipients only does so for the purpose of contacting a customer (namely, in the case of the predefined constant *contact* for the PURPOSE.

It is also sensible to have a check that prevents collection of data for unspecified purposes. The P3P Recommendation allows a policy to specify a retention value, for a given datagroup, of *indefinitely*, while also permitting an unspecified purpose (in the case of the value *other_purposes*.

These checks can be expressed in our CSP model in an analogous manner to $CONSISTENCY_1$.

5. Refinement checking

As explained in the Introduction, it is important to be able to compare two given policies together in order to see if they express the same intention, if they are simply equal, or whether they permit similar data collection, processing and sharing practices.

We can perform a simple test for trace refinement to see whether the top-level processes describing two policies refine one another. If we have two policies, P_1 and P_2, with corresponding CSP processes $POLICY_1$ and $POLICY_2$, this amounts to checking whether:

$$POLICY_1 \sqsubseteq_T POLICY_2 \tag{1}$$

$$POLICY_2 \sqsubseteq_T POLICY_1 \tag{2}$$

where the notation $P \sqsubseteq_T Q$ denotes trace refinement.

It is worthwhile to remind the reader here of the notion and use of trace refinement as used in process algebra. A *trace* is any finite or infinite sequence of steps (actions) that a given process may take or perform, according to its specification, in a given execution. The set of all traces of a process P is denoted $traces(P)$. For given processes P and Q we say that *P is refined by Q*, or $P \sqsubseteq_T Q$, if and only if $traces(P) \supseteq traces(Q)$. Process Q refines process P in the sense that its possible behaviours are fewer than those of P, so any user or system expecting to work or interact with process Q will be content with encountering P instead.

If the relations (1) and (2) both hold, then this means that the two P3P policies P_1 and P_2 are exactly identical, specifying the same statements in effectively the same order. This is not a case of much practical interest, and it may be more helpful to compare the individual practices permitted by P_1 with those permitted by P_2.

In Section 5.1 we discuss a realistic example (which uses refinement as shown in (1)) in which two policies are compared as to their overall effect (namely, the totality of practices permitted by each of the two policies). In Section 5.2 we develop checks which allow to detect differences between the practices permitted by two policies.

5.1. Detailed example of a full refinement check

In this section we demonstrate our approach by building a CSP model to compare two realistic privacy policies, shown in Figs. 3 and 4.

It is evident that both policies refer to three different datagroups, `#dynamic.http`, `#dynamic.clickstream` and `#dynamic.cookies`. Policy C comprises three distinct statements, one for each datagroup, while Policy D has two statements, with the second statement applying to both datagroups `#dynamic.http` and `#dynamic.cookies`. Notice the differences in retentions between the statements.

Our aim is to show that Policy C is refined by Policy D, or $C \sqsubseteq D$.

The differences between these policies are summarised in Figs. 5 and 6.

Using our translation tool (as discussed in Section 3), we convert these policies to CSP models with top-level processes $POLICY_C$ and $POLICY_D$. The corresponding CSP code, along with the check below is available online at http://www.dcs.warwick.ac.uk/~nikos/downloads/sampleref.csp.

To check refinement, we abstract away from the different purposes and retentions in *collect* events, as it is their occurrence that resets the retention counter, independently of the other parameters. To do this we define a function *abs*:

$$abs(P) = P[\forall dg, p, r, t \bullet collect.dg.p.r.t \mapsto learn.dg.r]$$

and the refinement check which shows that $C \sqsubseteq D$ is:

$$abs(POLICY_C) \sqsubseteq_T abs(POLICY_D)$$

This check succeeds in FDR as expected.

5.2. Separate checks for collection, processing and sharing practices

If any of the refinement checks mentioned in this section fails, it is an indication of a difference in particular practices in the policies under comparison. This allows a user to locate particular differences that may be of interest.

```
<POLICY xml:lang="en">
    <STATEMENT>
    <PURPOSE><admin/>
             <current/>
             <develop/>
    </PURPOSE>
    <RECIPIENT><ours/></RECIPIENT>
    <RETENTION><indefinitely/>
    </RETENTION>
    <DATA-GROUP>
    <DATA ref="#dynamic.clickstream"/>
    </DATA-GROUP>
</STATEMENT>
<STATEMENT>
    <PURPOSE><admin/>
             <current/>
    </PURPOSE>
    <RECIPIENT><ours/></RECIPIENT>
    <RETENTION><legal-requirement/>
    </RETENTION>
    <DATA-GROUP>
    <DATA ref="#dynamic.http"/>
    </DATA-GROUP>
</STATEMENT>
<STATEMENT>
    <PURPOSE><current/>
             <develop/>
    </PURPOSE>
    <RECIPIENT><ours/></RECIPIENT>
    <RETENTION><stated-purpose/>
    </RETENTION>
    <DATA-GROUP>
    <DATA ref="#dynamic.cookies"/>
    </DATA-GROUP>
</STATEMENT>
</POLICY>
```

Fig. 3. Policy C.

In order to compare data collection practices of P_1 and P_2 (so as to see, for example, if one policy requires the collection of the same types data or more types than the other), we define a set of processes which detect, for all datagroups *dg*, all the *collect* events for the different allowed retentions. We define:

$$\begin{aligned}
Collections(no\_retention) &= \Box\, dg : Datagroup \bullet collect.dg.p.r.no\_retention \\
&\quad \rightarrow Collections(no\_retention) \\
Collections(stated\_purpose) &= \Box\, dg : Datagroup \bullet collect.dg.p.r.stated\_purpose \\
&\quad \rightarrow Collections(stated\_purpose) \\
&\vdots \\
Collections(indefinitely) &= \Box\, dg : Datagroup \bullet collect.dg.p.r.indefinitely \\
&\quad \rightarrow Collections(indefinitely)
\end{aligned}$$

By composing $POLICY_1$ with $Collections(p)$, for each of the purposes p, while synchronising on collection events, we obtain a set of new processes that represent the collection practices permitted by policy P_1. We do similarly for policy P_2.

$$\begin{aligned}
TEST_1(t) &= POLICY_1 |[\{collect.dg.p.r.t\}]| Collections(t) \\
TEST_2(t) &= POLICY_2 |[\{collect.dg.p.r.t\}]| Collections(t)
\end{aligned}$$

We can compare the collection practices of the two policies by checking that, for each t:

$$TEST_1(t) \sqsubseteq_F TEST_2(t)$$

where the notation $P \sqsubseteq_F P'$ denotes failures refinement, i.e. $failures(P') \subseteq failures(P)$.

```
<POLICY xml:lang="en">
<STATEMENT>
    <PURPOSE><admin/>
             <develop/>
    </PURPOSE>
    <RECIPIENT><ours/></RECIPIENT>
    <RETENTION><stated-purpose/>
    </RETENTION>
    <DATA-GROUP>
    <DATA ref="#dynamic.clickstream"/>
    </DATA-GROUP>
</STATEMENT>
<STATEMENT>
    <PURPOSE><current/>
    </PURPOSE>
    <RECIPIENT><ours/></RECIPIENT>
    <RETENTION><no_retention/>
    </RETENTION>
    <DATA-GROUP>
    <DATA ref="#dynamic.http"/>
    <DATA ref="#dynamic.cookies"/>
    </DATA-GROUP>
</STATEMENT>
</POLICY>
```

Fig. 4. Policy D.

Statement	Datagroup	Purposes	Recipients	Retention
1	`dynamic_clickstream`	$\{current, admin, develop\}$	*ours*	*indefinitely*
2	`dynamic_http`	$\{current, admin\}$	*ours*	*legal_requirement*
3	`dynamic_cookies`	$\{current, develop\}$	*ours*	*stated_purpose*

Fig. 5. Comparing policies: Policy C.

Statement	Datagroup	Purposes	Recipients	Retention
1	`dynamic_clickstream`	$\{admin, develop\}$	*ours*	*stated_purpose*
2	`dynamic_http` `dynamic_cookies`	$\{current\}$	*ours*	*no_retention*

Fig. 6. Comparing policies: Policy D.

Failures refinement is used here in order to compare the sets of actions which these processes will be unable to perform by definition. If we were to compare traces this would only allow us to compare the space of all possible behaviours; but to compare the abilities of processes *not* to perform certain events (in this case, events representing particular data collection practices) allows us to contrast them more directly.

We have similar tests for the processing and sharing practices of the two policies. For comparing processing practices, we define a set of processes which detect, for all datagroups *dg*, all the *process* events for the different allowed purposes:

$$\begin{aligned}
Purposes(current) &= \Box\, dg : Datagroup \bullet process.dg.current.r.t \\
&\quad \rightarrow Purposes(current) \\
Purposes(admin) &= \Box\, dg : Datagroup \bullet process.dg.admin.r.t \\
&\quad \rightarrow Purposes(admin) \\
&\vdots \\
Purposes(other\_purposes) &= \Box\, dg : Datagroup \bullet process.dg.other\_purposes.r.t \\
&\quad \rightarrow Purposes(other\_purposes)
\end{aligned}$$

In an analogous fashion to the tests for collection practices, we define:

$$\begin{aligned}
TEST_3(p) &= POLICY_1|[\{process.dg.p.r.t\}]|Purposes(p) \\
TEST_4(p) &= POLICY_2|[\{process.dg.p.r.t\}]|Purposes(p)
\end{aligned}$$

Comparing the processing practices of the two policies is done by checking that, for each purpose p,

$TEST_3(p) \sqsubseteq_F TEST_4(p)$

Finally, for comparing sharing practices, we define:

$$\begin{aligned}
Sharing(ours) &= \Box\, dg : Datagroup \bullet process.dg.p.ours.t \rightarrow Sharing(ours)\\
Sharing(same) &= \Box\, dg : Datagroup \bullet process.dg.p.same.t \rightarrow Sharing(same)\\
&\vdots\\
Sharing(unrelated) &= \Box\, dg : Datagroup \bullet process.dg.p.unrelated.t \rightarrow Sharing(unrelated)
\end{aligned}$$

We then check refinement between the processes $TEST_5$ and $TEST_6$ for each possible retention r, where:

$$\begin{aligned}
TEST_5(r) &= POLICY_1|[\{share.dg.r\}]|Sharing(r)\\
TEST_6(r) &= POLICY_2|[\{share.dg.r\}]|Sharing(r)
\end{aligned}$$

The refinement check in this case is $TEST_5 \sqsubseteq_F TEST_6$.

6. Conclusions and future work

In this paper we have studied how formal modelling and verification can be applied to the analysis of privacy policies expressed in P3P. As we have discussed, there are three different classes of problem that this approach is intended to solve: (a) the need to compare higher-level policies with lower-level implementations, (b) the need to ensure that any one policy is sensible, plausible, and consistent relative to some objectively agreed standard (since no formal semantics were provided, especially in the case of P3P, by its authors), and (c) the situation which arises when there are two similar, but not exactly equal, policies which need to be compared.

We took note of existing work on the formalisation and analysis of policies, though noted a lack of work particularly related to privacy. Our approach was influenced by [16] and also inspired by the ideas in [18]. We presented a novel and practical technique for analysing privacy policies and capturing the intentions of their authors. Furthermore, we developed a modelling framework and code generation tool which can be used to reason about actual P3P policies and which is capable of many further extensions.

In Section 4 we discussed how the CSP model of a P3P policy can be used as a basis to check for internal semantic consistency. Rather than a syntactic validation (which is a commonly available feature in any XML parser), we are able to express relationships between different predefined values in the P3P Recommendation.

In Section 5 we presented refinement checks which may be performed on a pair of policies (after conversion to CSP using our tool) using the FDR model checker.

Formalising an enterprise's overall privacy requirements is an arduous and time-consuming task; however, we are aware that large companies conduct substantial efforts doing just this for their corporate rulebooks. With such a formalisation at hand, one can check (using the techniques presented in this paper) that the actual privacy policies being used, e.g. on corporate webpages, conform to company rules. In this paper we have focused on refinement checks between pairs of policies, and on consistency checking, with the expectation that the techniques used apply equally in the above case.

We believe that this work offers many opportunities for further and related investigations, including the following.

In order to make use of the policy comparison offered by our method, we plan to analyse and contrast privacy policies from several commercial websites. The Privacy Finder search engine is particularly useful in this regard, since we can locate sites offering various degrees of privacy and extract their P3P source. We are likely to encounter subtleties and special cases which will improve our CSP model further. We have used the IBM P3P Policy Editor [11] to create the sample policies for the analyses presented here.

In order to develop further internal consistency checks for P3P policies, it may be beneficial to combine our techniques with the algorithm described in [14]. This is likely to result in the development of a comprehensive policy analysis tool, which invokes FDR behind the scenes.

In connection with our work as part of the EnCoRe project [3], we are particularly keen to investigate the impact of privacy policies on users, and how best to implement user interfaces and systems that enforce users' privacy preferences. A user-side privacy agent may need to compare two websites that a user wishes to view as to their collection, sharing, and processing practices, so as to show the less privacy-invasive one to the user first; the techniques we have described in this paper are suited directly to this and related tasks.

We have also developed a logical approach to reasoning about users' privacy preferences, namely, about the consent and revocation mechanisms that may be available to them during online transactions [26].

Acknowledgements

We wish to thank Marco Casassa Mont and Siani Pearson (HP Labs) for insights into P3P, XACML and policy enforcement in enterprises.

References

[1] The PRIME project. URL: https://www.prime-project.eu/.
[2] The PrimeLife project. URL: http://www.primelife.eu.
[3] The EnCoRe project. URL: http://www.encore-project.info.
[4] L. Cranor, B. Dobbs, S. Egelman, G. Hogben, J. Humphrey, M. Langheinrich, M. Marchiori, M. Presler-Marshall, J.M. Reagle, M. Schunter, D.A. Stampley, R. Wenning, The Platform for Privacy Preferences 1.1 (P3P1.1) specification, World Wide Web Consortium, Note NOTE-P3P11-20061113 (November 2006).
[5] G. Hogben, A technical analysis of problems with P3P v1.0 and possible solutions, in: Proceedings of W3C Workshop on the Future of P3P, 2002. URL: http://www.w3.org/2002/p3p-ws/pp/jrc.html.
[6] T. Moses, eXtensible Access Control Markup Language TC v2.0 (XACML) (February 2005). URL: http://docs.oasis-open.org/xacml/2.0/access_control-xacml-2.0-core-spec-os.pdf.
[7] T. Moses, Privacy policy profile of XACML (16 September 2004). URL: http://docs.oasis-open.org/xacml/access_control-xacml-2_0-privacy_profile-spec-cd-01.pdf.
[8] J.D. Moffett, M.S. Sloman, Policy hierarchies for distributed systems management, IEEE J. Sel. Areas Commun. 11 (1993) 1404–1414.
[9] A.W. Roscoe, Understanding Concurrent Systems, Springer Verlag, 2010.
[10] Formal Systems (Europe) Ltd., FDR2 User's Manual version 2.82 (June 2005). URL: http://www.fsel.com/documentation/fdr2/fdr2manual.pdf.
[11] IBM P3P Policy Editor. URL: http://www.alphaworks.ibm.com/tech/p3peditor.
[12] Privacy Finder. URL: http://www.privacyfinder.org/.
[13] M.J. May, C.A. Gunter, I. Lee, Strong and weak policy relations, in: Proceedings of IEEE International Symposium on Policies for Distributed Systems and Networks, POLICY'09, London, UK, 2009.
[14] T. Yu, N. Li, A.I. Antón, A formal semantics for P3P, in: Proceedings of ACM Workshop on Secure Web Services, Fairfax VA, USA, 2004.
[15] J.W. Stamey, R.A. Rossi, Automatically identifying relations in privacy policies, in: Proceedings of the 27th ACM International Conference on Design of Communication, SIGDOC'09, ACM, New York, NY, USA, 2009, pp. 233–238. doi:10.1145/1621995.1622041.
[16] J. Bryans, Reasoning about XACML policies using CSP, in: Proceedings of the 2005 Workshop on Secure Web Services, SWS'05, ACM, New York, NY, USA, 2005, pp. 28–35. doi:10.1145/1103022.1103028.
[17] P. Ryan, R.R. Arnesen, A process–algebraic approach to security policies, in: E. Gudes, S. Shenoi (Eds.), DBSec: IFIP Conference Proceedings, vol. 256, Kluwer, 2002, pp. 301–312.
[18] K. Fisler, S. Krishnamurthi, L.A. Meyerovich, M.C. Tschantz, Verification and change-impact analysis of access-control policies, in: Proceedings of the 27th International Conference on Software Engineering, ICSE'05, ACM, New York, NY, USA, 2005, pp. 196–205. doi:10.1145/1062455.1062502.
[19] N. Zhang, M. Ryan, D.P. Guelev, Synthesising verified access control systems in xacml, in: Proceedings of the 2004 ACM Workshop on Formal methods in Security Engineering, FMSE'04, ACM, New York, NY, USA, 2004, pp. 56–65. doi:10.1145/1029133.1029141.
[20] C.A. Gunter, M.J. May, S.G. Stubblebine, A formal privacy system and its application to location based services, in: D. Martin, A. Serjantov (Eds.), Proceedings of PET 2004, in: Lecture Notes in Computer Science, vol. 3424, Springer, 2005.
[21] M.J. May, C.A. Gunter, I. Lee, Privacy APIs: Access control techniques to analyze and verify legal privacy policies, in: Computer Security Foundations Workshop, CSFW, July 2006, Venice, Italy, 2006.
[22] C. Fournet, A.D. Gordon, S. Maffeis, A type discipline for authorization policies, ACM Transactions on Programming Languages and Systems 29 (5) (2007) 25. doi:10.1145/1275497.1275500.
[23] D. Walker, A type system for expressive security policies, Tech. rep., Ithaca, NY, USA, 1999.
[24] M.Y. Becker, A. Malkis, L. Bussard, S4P: A generic language for specifying privacy preferences and policies, Technical Report MSR-TR-2010-32, Microsoft Research, April 2010.
[25] J. Ouaknine, S. Schneider, Timed CSP: A retrospective, in: Proceedings of the Workshop on Algebraic Process Calculi: The First Twenty Five Years and Beyond, in: ENTCS, vol. 162, 2006.
[26] I. Agrafiotis, S. Creese, M. Goldsmith, N. Papanikolaou, The logic of consent and revocation (2010) (submitted for publication).