# A Decision Support System for Design for Privacy

Siani Pearson, Azzedine Benameur

# A Decision Support System for Design for Privacy

Siani Pearson, Azzedine Benameur
Cloud and Security Lab
Hewlett-Packard
Bristol, UK
{siani.pearson, azzedine.benameur}@hp.com

**Abstract.** Privacy is receiving increased attention from both consumers, who are concerned about how they are being tracked and profiled, and regulators, who are introducing stronger penalties and encouragements for organizations to comply with legislation and to carry out Privacy Impact Assessments (PIAs). These concerns are strengthened as usage of internet services, cloud computing and social networking spread. Therefore companies have to take privacy requirements into account just as they previously had to do this for security. While security mechanisms are relatively mature, system and product developers are not often provided with concrete suggestions from a privacy angle. This can be a problem because developers do not usually possess privacy expertise. In this paper we argue that it would be useful to move beyond current best practice – where a set of searchable privacy guidelines may be provided to developers – to automated support to software developers in early phases of software development. Specifically, our proposal is a decision support system for design for privacy focused on privacy by policy, to be integrated into the development environment. We have implemented a proof of concept and are extending this work to incorporate state-of-the art consent mechanisms derived from the EnCoRe (Ensuring Consent and Revocation) project [1].

**Keywords:** Decision Support, Expert System, Patterns, Privacy, Software engineering

## 1 Introduction

A key challenge for software engineers is to design software and services in such a way as to decrease privacy risk. As with security, it is necessary to design privacy in from the outset, and not just bolt on privacy mechanisms at a later stage. There is an increasing awareness for the need for design for privacy from both companies and governmental organisations [2,3]. However, software engineers may lack privacy knowledge and the motivation to read and inwardly digest long and complicated guidelines. Therefore, to support software engineers in implementing privacy aware systems, it is necessary to move beyond a set of searchable guidelines. In this paper we address this problem by first extracting relevant high-level privacy design concepts from existing guidelines (with a focus on current best practice for privacy by policy [4]), and we then translate these concepts into context-dependent rules and privacy design patterns. This allows us to build decision support systems to help

developers design privacy in early phases of the software development life cycle and potentially also improve design at a later stage.

## 2 Related work

For some decades, mechanisms have been developed to address the issue of taking security into account in early phases of the software development life cycle. However, until recently this has not been the case for privacy. To address this problem some software companies have issued privacy guidelines for their developers but these guidelines are not easily applicable by developers and often rely on their own interpretations, leading to an error-prone process. Our approach limits errors and potential misunderstanding of guidelines by shifting the reasoning and understanding from developers to a decision support system.

*Privacy design techniques* are not a new concept: various companies, notably Microsoft [2], have produced detailed privacy design guidelines. Cannon has described processes and methodologies about how to integrate privacy considerations and engineering into the development process [5]. Privacy design guidelines in specific areas are given in [6,7], and [3] considers the case of cloud computing. In November 2007 the UK Information Commissioners Office (ICO) [8] (an organisation responsible for regulating and enforcing access to and use of personal information), launched a Privacy Impact Assessment (PIA) [8] process (incorporating privacy by design) to help organizations assess the impact of their operations on personal privacy. This process assesses the privacy requirements of new and existing systems; it is primarily intended for use in public sector risk management, but is increasingly seen to be of value to private sector businesses that process personal data. Similar methodologies exist in Australia, Canada and the USA [9]. This methodology aims to combat the slow take-up of designing in privacy protections at the enterprise level: see [10] for further discussion, [11] for further background, and [12] for a useful classification system for online privacy. There has also been related encouragement of a 'privacy by design' approach by the Canadian Privacy Commission. Our approach can be viewed in this context as a manifestation of a design for privacy or privacy by design approach that addresses a core subset of the concerns needed, focused around consent and notice mechanisms. Unlike a Privacy Impact Assessment, which is directed at organizations to provide an assessment of risk related to projects or activities, it is directed at developers in order to help them design privacy into online products and services.

Our approach focuses on "privacy by policy", by which means privacy rights are protected through laws and organizational privacy policies, which must be enforced [4]. Privacy by policy mechanisms focus on provision of notice, choice, security safeguards, access and accountability (via audits and privacy policy management technology). Often, mechanisms are required to obtain and record consent. The 'privacy by policy' approach is central to the current legislative approach, although there is another approach to privacy protection, which is 'privacy by architecture' [4], which relies on technology to provide anonymity. The latter is often viewed as too expensive or restrictive, as it is not suitable for all situations, limits the amount of data

available for data mining, research and development, targeting or other business purposes, and may require more complicated system architectures and expensive cryptographic operations. We consider in this paper a solution to advising developers that is focused on privacy by policy as the elements can more easily be broken down; we plan in future to extend this approach to cover a hybrid approach with privacy by architecture.

In order to provide a practical technique for design, we utilise *design patterns* [13]. Some previous work has been carried out in the privacy design pattern area: [14] describes four design patterns applicable to the design of anonymity systems. These could be integrated into our approach at a later stage, when we move on from considering our current "privacy by policy" rules to extend this to "privacy by architecture" approaches, hybrid approaches and assessment of the relative merits of the patterns.

A number of existing tools provide a framework for the generation of decision support systems [15,16,17,18]. Decision support systems have been developed for privacy [19], but not for design related to privacy by policy, nor addressing our focus on suggesting design patterns related to provision of appropriate notice and consent mechanisms.

In the security domain several support systems for security have been proposed, the closest approach to our work being SERENITY [20,21,22]. The latter supports developers from the early design phase by providing security solutions, going beyond a set of design patterns to provide both patterns and executable components. However, this approach has not been provided with privacy in mind and does not offer any privacy features. Delessy et al. [23] have discussed how to build upon model-driven development and the use of security patterns in order to secure applications in service-oriented architectures. Laboto et al. [24] have proposed the use of patterns to support the development of privacy policies, but – unlike in our approach – a rule engine was not proposed to automatically select appropriate patterns at the design stage, and the focus again was on security.

## 3 A DSS focused on privacy by policy

In this section we provide more information about the decision support system that we propose to address core aspects of privacy in the software design phase. We first give an overview of how the system operates, and then provide some examples of design patterns and explain more about the rulebase and inference procedures.

### 3.1 System overview

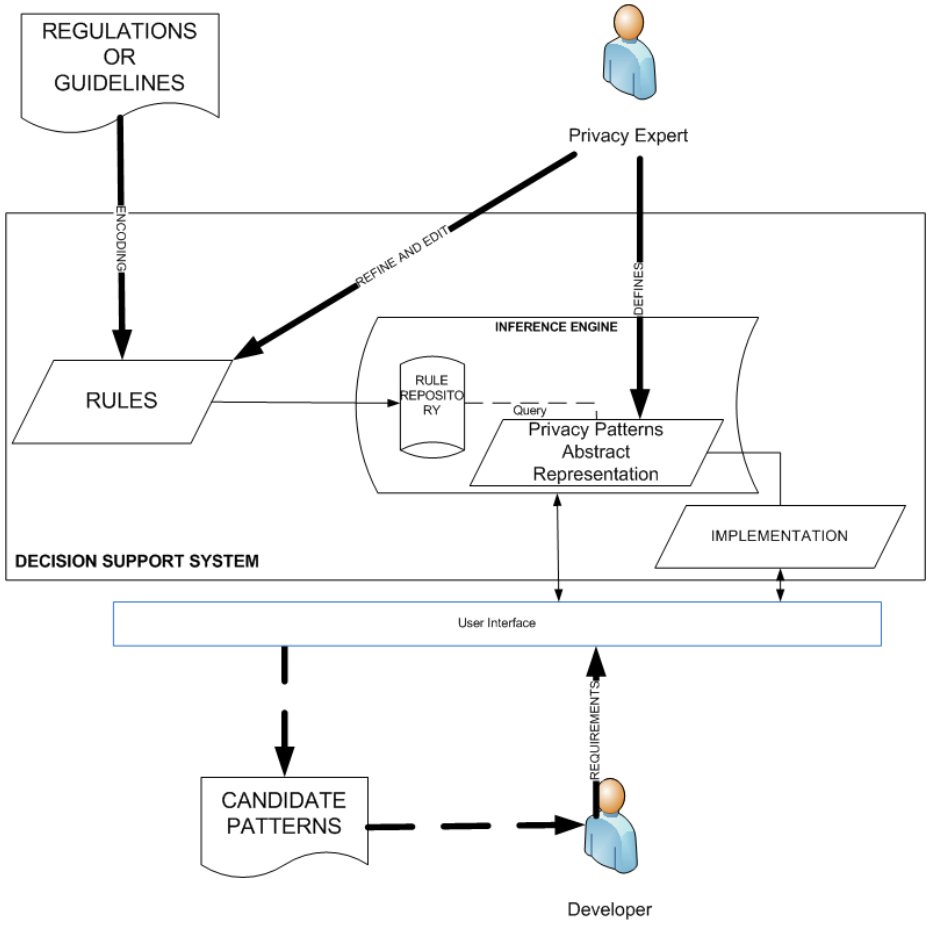


**Figure 1. Decision Support System Architecture**

The architecture of our Decision Support System (DSS) is depicted in Figure 1. The thick solid lines represent input to or within the system while the thick dashed lines are the output of the system. In our DSS, two actors are present: a privacy expert in charge of refining the rules derived from guidelines and of providing an abstract representation of corresponding privacy patterns, and a developer who wishes to integrate privacy into their design and who will achieve this by implementing privacy patterns output by the DSS. The developer is the target end-user of the DSS. As a user of the DSS, the developer provides his/her requirements, as input to a questionnaire shown by the system. This operation is done through a user-friendly interface but could instead be integrated into standard integrated development environments such as Eclipse. The DSS forwards this input to an inference engine. Based on the particular requirements and context set from the answers to the questionnaire, the

engine queries the rule repository to obtain potential applicable patterns. The inference engine uses information about the implementation and abstract representations to reason and produce an output that is a set of candidate patterns matching the developer's requirements and context. The core of the DSS relies on the encoding of best practice guidelines into a machine readable format and this is why the assistance of privacy experts is crucial in order to have a sound rulebase.

To date we have analyzed the Microsoft privacy design guidelines [2] and some other documentation related to privacy by policy, and extracted the core concepts. A couple of these concepts are presented as examples in the next section. As discussed further in Section 5, we are now extending the knowledge base to include design techniques for consent that we have been developing within the EnCoRe project [1].

### 3.2 Design patterns

Design patterns are a way to capture solutions to commonly occurring problems. In this section we present two examples related to the concept of "explicit consent". For this purpose we use radio buttons and checkboxes as some examples of how explicit consent may be provided.

***Design Pattern 1:***
**Name**: checkbox
**Classification**: opt-in consent
**Intent**: provide an opt-in consent mechanism
**Motivation**: an explicit consent experience that is opt-in means that the proposition presented will only occur after the customer takes an action
**Context**: you are presenting an option with a checkbox
**Solution**: the checkbox cannot be pre-checked: the customer must actively check the box with text containing the privacy choice in order to enable the data collection, use, or transfer.
**Use case**: checkboxes, one of which is not pre-checked and is associated with the text "I want to help make Microsoft products and services even better by sending Player usage data to Microsoft"
**Related patterns**: radio button

***Design Pattern 2:***
**Name**: radio button
**Classification**: opt-in consent
**Intent**: provide an opt-in consent mechanism
**Motivation**: an explicit consent experience that is opt-in means that the proposition presented will only occur after the customer takes an action
**Context**: you are providing multiple choices to the customer
**Solution**: the customer must select an option that was not selected by default. The design should not select any radio buttons by default, i.e. it should require the customer to actively select one of them, to consent or not consent to collection or transfer of data.

**Use Case:** radio buttons, one of which is not pre-checked and is associated with the text "I want to sent statistic data about my usage of Chrome to Google"
**Related patterns**: checkbox

These design patterns describe mechanisms for providing consent and explicit consent using radio buttons and checkboxes and their related application context. The software developer, via UIs, provides his/her input to the decision support system that triggers a set of rules to find matching patterns related to both the input and the context. We then map these results to design patterns that show the developer how in practice this can be achieved.

In this section we have presented two simple patterns related to the expression of "explicit consent". However there are a number of other design patterns that provide alternative methods of consent (including implicit consent) and in addition various mechanisms for provision of notice (both discoverable and prominent), each suitable for different contexts. We have defined such patterns, based upon techniques described in the Microsoft developer guidelines, and these are used within our initial prototype, as described in Section 4.

In addition, we have been extending this approach to build up a broader set of design patterns, including more complex consent management mechanisms, usage of obligations and sticky policies. An example from this set is of flexible policy negotiation between two entities (e.g. the data subject and data controller), where a number of different protocols for such negotiation (based upon inputting initial policies from both parties and then running a negotiation protocol to try to achieve agreement on a common, or new, set of policies) can each generate design patterns based upon the following generic template:

***Design Pattern:***
**Name:** Negotiation
**Classification:** Data and policy management
**Intent:** to allow negotiation of preference attributes to share only what is absolutely necessary
**Motivation:** A scenario where this would be useful is when a service provider (SP) subcontracts services, but wishes to ensure that the data is deleted after a certain time and that the SP will be notified if there is further subcontracting
**Context:** You are a user interacting with a system and want to share the least possible information to be able to interact with this system. You define a subset of your personal data. This subset represents the data you would absolutely not want to share. You can also express a sharing option on each attribute of your personal data.
**Problem:** Systems have policies that require a certain subset of your personal data to be able to perform some transactions. The key problem here is to find a matching between the user's preference over his/her personal attributes and the system requirements/organizational policies. The negotiation is here to provide more flexible policies and the least possible information disclosure for users.
**Solution:** Use a policy negotiation protocol. This protocol requires users to express preferences for each of their PII's elements. Then the negotiation consists of mandatory attribute requests from the system and seeing if this matches user

preferences. A successful negotiation is when this exchange leads to a minimized set of PII attributes being shared.

**Design issues:**

- Need to have a mechanism to express preferences over each attributes for users.
- Not compatible with legacy systems.

**Consequences:** Benefits: privacy preferences of users are respected; systems benefit because it may lead to more usage as users are sharing only attributes that they want.

**Related patterns:** Privacy Policy

As described in Section 5, the extension of our design pattern repository is the subject of ongoing research.

### 3.3 Rules and representations

In order to be able to reason about different privacy patterns, natural language is not suitable; instead, we use a more formal representation of the patterns introduced in subsection 3.2. in order to make inferences about patterns. In this section we consider a representation for our system of deduction, the knowledge base and the inference mechanisms that serve as a foundation for the implementation of such a system described in the following section.

#### System of deduction

To formalize our system of deduction we may use the representation of a propositional logic $L_1$, where $L_1$=L(P,{ ¬ , &,∨, →}, $Z$, ∅) where P is a finite alphabet of propositional variables, $Z$ are the rules of inference, there are no axioms and the connectives negation ¬, conjunction &, disjunction ∨ and material implication → are used to build up well-formed formulae using the symbols of P according to the standard inductive process for doing this (see for example [25] for further details) .

The rules of inference $Z$ correspond to the exact propositional formulation used: for example, reduction ad absurdum, double negative elimination, conjunction introduction, conjunction elimination, disjunction introduction, disjunction elimination, modus ponens, conditional proof – or else, for example, if a sequent calculus were used, the axiom and inference rules of the propositional version of Gentzen's sequent calculus LK (see [25] for further details).

The logical representation above is minimal, in the sense that a propositional logic is used as the basis for the representation. There is scope for a more complex representation to be used for this problem, notably modal logic – where we could distinguish between necessary, preferable and possible relationships [26], or possibilistic logic, where necessity-valued formulae can represent several degrees of certainty [27].

Let us now consider our system. In our case, as an initial example, $P = \mathrm{CN} \cup \mathrm{Con}$ where CN are the set of consent and notice requirements, and Con are the contextual settings. We define:

$$CN = \{C, IC, EC, CP, CA, SMC, AP, OU, OI, N, DN, PN\}$$

$$Con = \{S, E, D, CF, CS, EA, CPI, SI, U, A, AC, AS, SP, TP, STP, PC, SO, T, TA\}$$

where the meaning of these symbols is as follows:

C: consent is required
IC: implicit consent is required
EC: explicit consent is required
CP: consent is required by a parent
CA: consent is required by an application or system administrator
SMC: separate mechanism for consent required
AP: authentication is required by a parent
OU: opt-out is required
OI: opt-in is required
N: notice is required
DN: discoverable notice is required
PN: prominent notice is required
S: sensitive PII is stored
E: data is used in ways that exceed the original notice
D: discrete transfer of anonymous and pseudonymous data
CF: file extensions already associated with another application are being changed
CS: users' PII is being exposed in a sharing or collaboration feature
EA: anonymous or pseudonymous data is being exposed in a sharing or collaboration feature
CPI: there is collection and disclosure of children's PII
SI: installation of software is involved
U: automatic update of software is involved
A: anonymous data is continuously collected and transferred
AC: sensitive PII is stored for the purpose of automatic completion
AS: age will be exposed in a sharing or collaboration feature
SP: PII transferred will be used for secondary purposes
TP: PII is transferred to or from the customer's system
STP: PII is shared in an independent third party
PC: PII will be stored in a persistent cookie
SO: sensitive information will be transferred and retained in a 1-time transaction
T: data will be transferred from the server over the Internet
TA: PII is transferred to an agent

**Knowledge base**

Our knowledge base kb is a finite and consistent set of formulae $p_i$ for $i = 1. . .n$ st.$p_i \in L_1$is a propositional formula. We assume the set of the propositional formulae occurring in kb (denoted $Prop_{kb}$) form a propositionally consistent set and as such do not lead to a contradiction. The formulae in kb have as possible interpretations all

classical interpretations ("worlds") I of the propositional variables P, i.e. these are "models" (denoted |=) of the propositional formulas in kb. The knowledge base is able to answer queries based on $|=_I$ in $L_1$. That is, the input is a propositional query formula p of $L_1$ and evaluation of this query outputs a formula q of $L_1$, such that q is true under the interpretation I.

We initialize the knowledge base to a set of well-formed formulae of $L_1$,which are the privacy consent and notice rules, which have truth value T in the current interpretation. We also, as described below, add an additional well-formed formula to the knowledge base that corresponds to the contextual settings of the UI from the user and that also has truth value T in the current interpretation.

The initial settings of the knowledge base are as follows:

$Prop_{kb} = \{C \rightarrow IC \vee EC, EC \rightarrow OU \vee OI, N \rightarrow PN \vee DN, PN \rightarrow DN, SMC \rightarrow C, S \rightarrow C \& N, E \rightarrow SMC, D \rightarrow C \& DN, CF \rightarrow C \& N, CS \rightarrow C \& PN, EA \rightarrow C \& N, CPI \rightarrow CP \& AP, SI \rightarrow EC \& PN, U \rightarrow EC \& PN, A \rightarrow EC \& PN, AC \rightarrow EC, AS \rightarrow EC \& PN, SP \rightarrow EC \& PN \& SMC, TP \rightarrow OI \& DN, STP \rightarrow OI \& PN, PC \rightarrow OI \& SMC, SO \rightarrow CA, T \rightarrow DN, TA \rightarrow DN\}$

The interpretations possible for a given case (i.e. the particular models that are possible corresponding to a system usage) will be restricted by the choices selected by the end user when answering the questionnaire, i.e. if the parameter selected by an end user via the questionnaire (e.g. *sensitivePIIinvolved*, *PIIexposedincollaboration*, *collectionofchildsPII*) is set to "yes", the corresponding propositional symbol (from the set Con described above) is assigned the truth value T; and if the parameter is set to "no" then the corresponding propositional symbol would be assigned the truth value F. Multiple interpretations are possible if not all values are known. Corresponding to this interpretation, we add a formula $u \in L_1$ to $Prop_{kb}$, where u is a clause made up of a conjunction of literals such that if $p_i \in Con$ has truth value T in the current interpretation, we add the literal $p_i$ but if $p_i \in Con$ has truth value F in the current interpretation, we add the literal $\neg p_i$. i.e.

$u = p_l \& \ldots \& p_r \& \neg q_l, \ldots \& \neg q_t$, where $p_i, q_i \in Con$

corresponds to the contextual settings of the UI from the user, and u will have truth value T in the current interpretation.

**Deduction**

In summary, we deduce the requirements (new well-formed formulae of $L_1$) from $Prop_{kb}$ using the rules of deduction *Z* (and then map formulae with certain properties onto design patterns as the output of the tool).

The inference rules are applied to $Prop_{kb}$ to deduce new formulae of $L_1$. In particular, we want to find $q_i \in P$ such that $Prop_{kb} \vdash q_i$. As we deduce these, we add them to $Prop_{kb}$. Alternatively, we can use a backwards chaining search method, starting with each $q_i \in CN$.

We assume that the user is asked all questions, or that those that are not asked can have the associated propositions in *Con* given a false truth value. In this way, we can assume the conjunction of all the propositional variables in Con (or their negations, if

that has been indicated according to the user response). Then, what is required is for the system to deduce which propositions (or their negations) from the set CN follow.

To make the inference process easier, we may convert $Prop_{kb}$ into conjunctive normal form (CNF), i.e. into a conjunction of clauses where each clause is a disjunction of literals (atomic formula or its negation), and then reduce this further (using the commutative properties of v and &, and the logical equivalence of *( ¬A v B) & A* to *B & A*) [28] in order to highlight the resultant literals. Thus, $Prop_{kb}$ can be reduced to *u* & *v*, where *v* is a clause in CNF that represents the additional knowledge derived by the system. In a separate process we then map the conjuncts within *v* onto design patterns.

**Example**

Let us consider the case where anonymous data is being collected and transferred within the system that the developer is designing. A central rule that corresponds to consent and notice advice for this situation that is given within the privacy guidelines in [2] may be paraphrased as "if anonymous data is continuously collected and transferred then explicit consent and prominent notice is required" and this may be represented using the formalism above as '*A → EC & PN',* which is one of the rules in the knowledge base of our system.

If we have a scenario where anonymous data is continually collected, transferred, and shared with others, then according to the developer's answers in the questionnaire, in this particular case we may assume the truth of the following formula:

¬S & ¬E & ¬D & ¬CF & ¬CS & EA & ¬CPI & ¬SI & ¬U & A & ¬AC & ¬AS & ¬SP & ¬TP & ¬STP & ¬PC & ¬SO & ¬T & ¬TA

By the mechanisms described above we may deduce the following additional formulae for this scenario that are of particular interest:

C, N, EC, PN, OU v OI, DN

By focusing on the more specific instances, we have the requirements: EC, PN.

As a follow-on inference process, these propositions are then mapped by our system to a choice of design patterns, i.e. some of these $q_i \in Prop_{kb}$ (in this case, EC and PN) are used to map onto design patterns as the output of the tool. So, the user will be shown a choice of patterns to enable him/her to implement explicit consent and prominent notice. This mapping may be more complex if desired than 1-1 or 1-many: for example, we could use a Boolean trigger that includes other contextual conditions (corresponding to the values of selected propositions in P) to output a suggested design pattern, and this can be useful in refining the design patterns to particular contexts.

## 4 Implementation

We have presented above the rule formalism used within the inference engine. The implementation of these rules into executable components is done in Java using JBoss Drools [29].

The implementation of our DSS is done as a plug-in added into a standard Integrated Development Environment (IDE) named Eclipse [30]. The concept of such an extension relies on views, which are different elements of the user interface, and on perspectives, which are layouts with several views. The Decision Support perspective includes three views for three different purposes: the privacy expert view, the developer view and the output view.

The *privacy expert view* is the dedicated user interface accessed by privacy experts to add design patterns. In our implementation we decided to condense the format of the design patterns (considered above in section 3.2) to only include the following fields: name, classification, source, solution. The classification is the basis for automated analysis about the relationship between patterns. The solution can contain text, code snippets and examples. The context is no longer defined within the pattern itself, but within the rules of the system.

The *developer view* is what developers need to answer in order for our DSS to offer a set of patterns addressing their issues. These questions represent the contextual settings of our system and in our implementation they correspond directly to the propositions within Con, as introduced in section 3.3. So the developer would be asked questions including the following examples and given the option to answer 'yes', no' or 'unsure', together with associated help where required as to what the questions mean:

- is personal data transferred to or from the customer's system?
- will the transferred personal information be used for secondary purposes?
- is personal data shared with an independent third party?
- is there collection or disclosure of children's personal data?
- is an automatic update of software involved?
- is sensitive data stored for the purposes of automatic completion?
- is personal data stored in a persistent cookie?
- will sensitive information be transferred and retained in a one-time transaction?

The set of candidate patterns is then presented in the *output view*, from which developers can select and implement the most suitable ones.

The logic of the system is implemented using a rules engine [29]. Figure 2 is an example of the implementation of this part of the knowledge base: in this case, $AS \rightarrow EC \ \& \ PN$. This propositional formula is linked to a question (i.e. "is age exposed in a sharing or collaboration feature?", with identity number 6, user options to answer 'yes', 'no' or 'unsure' and associated help) and the rule shown in Figure 3 expresses that if the answer given by the user to this question is 'yes' then the list of requirements should include Explicit Consent and Prominent Notice. Similarly, there are analogous JBoss rules that represent the other inference rules of the system, in our case that correspond to all the formulae of $Prop_{kb}$.

Here, the Java function 'addToRequirementsList' will delete duplication and the more general requirements (that duplicate the more specific requirements in the list) to output the most specific requirements. This final list is then mapped to design patterns, using the classification field of the design patterns, and these are output in a list to the developer.

```
rule "Age Sharing"
    when
    Question ( id == 6, value == "Yes")
    then
    addToRequirementsList (List, [EC,PN])
end
```

**Figure 2. Example rule implementation**

## 5 Further Work

While this approach enhances privacy design it is possible to extend the rulebase in order to provide more accurate reasoning capabilities and a more comprehensive set of output choices.

A next step is to build up our repository of privacy patterns, for example by incorporating other privacy guidelines (including those from HP and Sun). We are also investigating integration of other methodologies.

Furthermore, within the EnCoRe project, we are currently developing techniques for providing enhanced consent and revocation [1] and we are extending our design pattern set and rulebase to capture such mechanisms. This approach will enable assessment of legacy systems and other contextual requirements in determining an appropriate approach to consent and revocation solutions, as well as a broader range of solutions.

In order to import, export or interface our knowledge base to other systems using a different rule language (which may be the case in particular for different domains) we plan to consider the use of the rules interchange format [31] to allow such interactions.

Although it did not seem necessary in order to capture the MS guidelines, our representation allows for capturing more complex rules if desired (for example, $A$ & $B \rightarrow C$, and arbitrarily complex expressions of $L_1$). These correspond in our implementation to the use of Boolean trigger conditions within the JBoss rules.

It is also possible to have a more intelligent way in which to ask the questions to the user in which the questions can be nested and only asked if appropriate, in order to reduce the number of questions asked. For example, is the answer to 'is users' personal data being exposed in a sharing or collaboration feature?' is 'no' then there is no need to ask the user the question 'will age be exposed in a sharing or collaboration feature?' because age is a type of personal data. This can be done by using templates of questions in which some are hidden, by using ontologies to detect

semantic hierarchies or else via defining JBoss rules that govern the generation of the questions themselves (see for example [32] for further details of such an approach).

We also wish to include further analysis within the pattern selection process, potentially involving grading of patterns. At present we are using a simplified mapping between requirements and design patterns, using the classification field. So for example, Design Pattern 1 above has a classification of 'opt-in consent', which indicates that it is a candidate for satisfying the opt-in consent requirement. In our implementation, the system groups together patterns for all security contexts that have been set to T, but this decreases the granularity of the suggestions. We plan to explore the use of more complex mappings, which for example allows a hierarchy of patterns to be defined more explicitly, and also can allow other background information (which can be gathered within the questionnaire) to be taken into account. More specifically, by grouping suggested patterns based on the corresponding privacy contexts from which they were derived, patterns that need to be implemented multiple times can be suggested with reference to each individual context, and the system output can display the corresponding context satisfied by implementing the patterns.

## 6 Conclusions

We have presented an approach to support developers in designing privacy in early phases of development. As a first step, we have taken Microsoft developer privacy guidelines, encoded them into rules to allow reasoning about privacy design and implemented a working decision support system. In doing this, we have moved beyond the state of the art (being a set of searchable privacy guidelines), to provide an architecture and a system that supports decision, selection and integration of privacy patterns, and to start building up a knowledge base that offers both privacy-enhancing mechanisms (in the form of a privacy pattern repository) and also rules that advise on the appropriate usage of those mechanisms.

## References


1. The EnCoRe project: Ensuring Consent and Revocation. http://www.encore-project.info (2008)
2. Microsoft Corporation, "Privacy Guidelines for Developing Software Products and Services", Version 2.1a, 26th April (2007).
3. Information Commissioners Office, "Privacy by Design", Report, November. www.ico.gov.uk (2008)
4. Spiekermann, S. Cranor, L., "Engineering Privacy", IEEE Transactions on Software Engineeing, Vol 35, No 1, January/February (2009)
5. Cannon, J.C., "Privacy: What Developers and IT Professionals Should Know", Addison Wesley (2004)
6. Patrick, A., Kenny, S., "From Privacy Legislation to Interface Design: Implementing Information Privacy in Human-Computer Interactions". R. Dingledine (ed.), PET 2003, LNCS 2760, pp. 107-124, Springer-Verlag Berlin Heidelberg (2003)

7. Belloti, V., Sellen, A., “Design for Privacy in Ubiquitous Computing Environments”, Proc. 3rd conference on European Conference on Computer-Supported Cooperative Work, pp. 77-92 (1993)
8. Information Commissioner"s Office, “PIA handbook”. http://www.ico.gov.uk/ (2007)
9. Office of the Privacy Commissioner of Canada, “Privacy impact assessments”, Fact Sheet http://www.privcom.gc.ca/ (2007)
10. Information Commissioners Office, “Privacy by Design”. Report, www.ico.gov.uk (2008)
11. Jutla, D. N., Bodorik, P.: Sociotechnical architecture for online privacy. IEEE Security and Privacy, 3(2), pp. 29-39. IEEE (2005)
12. Spiekermann, S., Cranor, L. F.: Engineering privacy. IEEE Transactions on Software Engineering, pp. 1-42. IEEE (2008)
13. Alexander, C., Ishikawa, S., Silverstein, M., Jacobson, M., Fiksdahl-King, I., Angel, S., “A Pattern Language: Towns, Buildings, Construction”. Oxford University Press (1977)
14. Hafiz, M., “A collection of privacy design patterns”. Pattern Languages of Programs, NY, pp. 1-13. ACM (2006)
15. Dicodess: Open Source Model-Driven DSS Generator, http://dicodess.sourceforge.net (2009)
16. XpertRule: Knowledge Builder, http://www.xpertrule.com/pages/info_kb.htm (2009)
17. Lumenaut: Decision Tree Package, http://www.lumenaut.com/decisiontree.htm (2009)
18. OC1 Oblique Classifier 1, http://www.cbcb.umd.edu/~salzberg/announce-oc1.html (2009)
19. Pearson, S., Sander, T., Sharma, R., “Privacy Management for Global Organisations”. Data Privacy Management and Autonomous Spontaneous Security, J. Garcia-Alfaro et. al. (eds.), LNCS 5939, pp. 9-17. Springer-Verlag Berlin (2010)
20. SERENITY: System Engineering for Security and Dependability. http://www.serenity-project.org (2009)
21. Kokolakis, S., Rizomiliotis, P., Benameur, A., Kumar Sinha, S.,: Security and Dependability Solutions for Web Services and Workflows : A Patterns Approach, Security and dependability for Ambient Intelligence, Springer, (2009)
22. Benameur, A., Fenet, S., Saidane, A., Khumar Sinha, S.: A Pattern-Based General Security Framework: An eBusiness Case Study, HPCC, Seoul, Korea (2009)
23. Delessy, N. A., d Fernandez, E. B.: A Pattern-Driven Security Process for SOA Applications, ARES: 416-421 (2008)
24. Lobato, L.L., d Fernandez, E. B., Sergio Donizetti Zorzo: Patterns to Support the Development of Privacy Policies. ARES: 744-74 (2009)
25. Mendelson, E., Introduction to Mathematical Logic, D. Van Nostrand Co (1964).
26. Blackburn, P., de Rijke, M., Venema, Y. Modal Logic, Cambridge University Press. ISBN 0-521-80200-8
27. Benferhat, S., Dubois, D., Prade, H., “Towards a possibilistic logic handling of preferences”. Applied Intelligence 14(3), 303–317 (2001)
28. Bundy, A. The Computer Modelling of Mathematical Reasoning, Academic Press, 2nd edition (1986).
29. JBoss, Drools, http://www.jboss.org/drools/ (2010)
30. Eclipse, http://www.eclipse.org/ (2010)
31. W3C, Rule Interchange Format, http://www.w3.org/2005/rules/wiki/RIF_Working_Group (2010)
32. Pearson, S., Rao, P., Sander, T., Parry, A., Paull, A., Patruni, S., Dandamudi-Ratnakar, Sharma, P. “Scalable, Accountable Privacy Management for Large Organizations”, 2nd International Workshop on Security and Privacy Distributed Computing, Enterprise Distributed Object Conference Workshop, pp. 168-175. IEEE (2009)