Automating Privacy Enforcement in Cloud Platforms

Peng Yu[1], Jakub Sendor[2], Gabriel Serme[3], and Anderson Santana de Oliveira[2]

[1] Université de Technologie de Compiègne, France
peng.yu@etu.utc.fr
[2] SAP Research, France
firstname.lastname@sap.com
[3] Eurecom, France
serme@eurecom.fr

Abstract. Privacy in cloud computing is a major concern for individuals, governments, service and platform providers. In this context, the compliance with regards to policies and regulations about personal data protection is essential, but hard to achieve, as the implementation of privacy controls is subject to diverse kinds of errors. In this paper we present how the enforcement of privacy policies can be facilitated by a Platform as a Service. Cloud applications developers can use non-obtrusive annotations in the code to indicate where personally identifiable information is being handled, leveraging the aspect-oriented programming (AOP) features. Subsequently the evaluation of user defined preferences is performed by trustful components provided by the platform, liberating developers from the burden of designing custom mechanisms for privacy enforcement in their software.

1 Introduction

In order to speed up the deployment of business applications, and to reduce overall IT capital expenditure, many cloud providers nowadays offer the Platform as a Service (PaaS) solutions as an alternative to leverage the advantages of cloud computing. We can mention for instance SAP NetWeaver Cloud, Google App Engine, or VMware Cloud Foundry, to cite a few. PaaS brings an additional level of abstraction to the cloud landscape, by emulating a virtual platform on top of the infrastructure, generally featuring a form of mediation to the underlying services akin to middleware in traditional communication stacks.

As the consequence of that shift we observe that more and more personally identifiable information (PII) is being collected and stored in cloud-based systems. This is becoming an extremely sensitive issue for citizens, governments, and companies, both using and offering cloud platforms. The existing regulations, which already established several data protection principles, are being extended to assign new responsibilities to cloud providers with respect to private data handling.

The provision of privacy preserving services and tools will be one of the arguments favoring the choice of one PaaS provider over the other when a company is hesitating where to deploy new cloud application. The proposed reform of the European data protection regulation points out that privacy-aware applications must protect personal data by design and by default: "Article 22 takes account of the debate on a 'principle of accountability' and describes in detail the obligation of responsibility of the controller to comply with this Regulation and to demonstrate this compliance, including by way of adoption of internal policies and mechanisms for ensuring such compliance. Article 23 sets out the obligations of the controller arising from the principles of data protection by design and by default. Article 24 on joint controllers clarifies the responsibilities of joint controllers as regards their internal relationship and towards the data subject[4]."

The correct enforcement of privacy and data usage control policies has been recently subject of several incidents reported about faulty data handling, perhaps on purpose, see for instance the cases of Facebook[5].

Therefore, addressing compliance requirements at the application level is a competitive advantage for cloud platform providers. In the specific cases where the cloud platform provider is also considered a joint controller, a privacy-aware architecture will address the accountability requirement for the PaaS provider with regards to the next generation of regulations. Such architecture can enable compliance also for the Software as a Service delivery model, if we assume the software was built over a privacy-aware platform. On the other hand, this could be hardly achieved in the context of Infrastructures as a Service, since there would be no interoperability layer on which the privacy controls can rely on.

In order to achieve this, the PaaS must implement some prominent, possibly standardized, privacy policy framework (such as EPAL[2], P3P[7]), where privacy preferences can be declared in a machine-readable form, and later enforced automatically. In such a setting, the privacy enforcement controls could be easily incorporated into new deployment landscape accelerating the development process of compliant applications. Furthermore the cloud platform can offer the guaranties ensuring the correct implementation of the enforcement components. This could be offered either via a certification mechanism or an audit of an existing cloud landscape that would be executed by the governing entities.

In this paper we present work towards the implementation of privacy-aware services in a PaaS. We aim to empower the cloud platform with capabilities to automatically enforce the privacy policy that is result of the end-user consent over the application provider privacy policy. End-user policies and service provider terms of use are defined in a state of the art privacy and usage control language [3]. In order to leverage the provided implementation of privacy-aware services, cloud application developers need to introduce simple annotations to the code, prior to its deployment in the cloud. These indicate where PII is being handled, towards automating privacy enforcement and enabling compliance by

[4] http://ec.europa.eu/justice/data-protection/document/review2012/com_2012_11_en.pdf

[5] http://mashable.com/2011/10/21/facebook-deleted-data-fine/

design and by default. The idea is outlined in Figure 1, and consists of design-time steps (declaring policies, annotation of the code and deployment in the cloud); and run-time steps (including policy matching, privacy control and obligation execution).

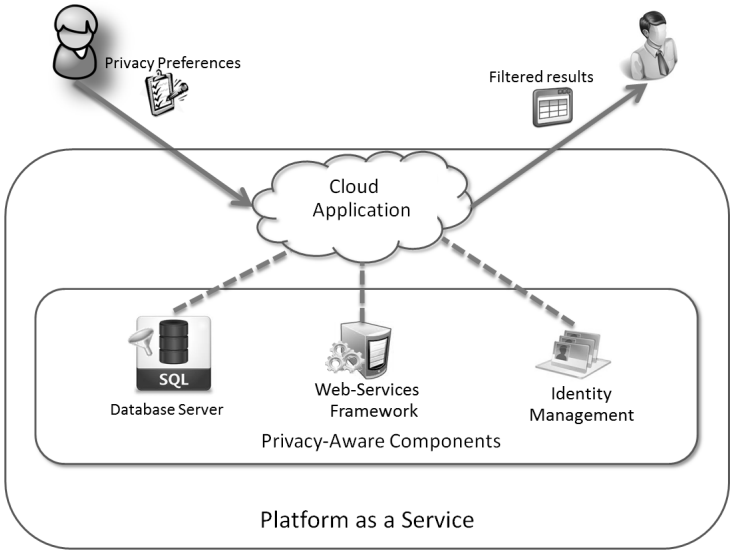

Fig. 1: Privacy aware PaaS components

The enforcement mechanisms are provided by the platform with the help of a new approach for aspect-oriented programming where aspects can be manipulated at the process and at the platform levels [8]. That approach gives a possibility to maintain a more flexible configuration of the enforcement mechanisms. The mechanisms interpret end-user preferences regarding handling of the PII, presented in form of opt-in or opt-out choices among available privacy policies of a cloud application, and later perform the required actions (filtering, blocking, deletion, etc). We experimented on a Java-based Platform as a Service, SAP NetWeaver Cloud, to demonstrate how privacy preferences can be handled automatically thanks to the use of simple Java annotation library provided in our prototype. The platform provider can make in this way an important step towards providing built-in compliance with the personal data protection regulations transparently, as we describe in the next sections.

The remainder of the paper is organized as follows: in Section 2 we present our use case and we give a brief overview of the privacy policy language we adopt in this work, in Section 3 we introduce the technical architecture allowing to enforce privacy on multiple PaaS layers, Section 4 brings a discussion on related works and Section 5 presents future perspectives and concludes the paper.

2 Privacy-Aware Applications in the Cloud

In this section we present our use case involving multiple stakeholders accessing users' PII in the cloud, as well as some background on privacy policy language that we used.

2.1 Use case

In our use case we consider a loyalty program offered by a supermarket chain, accessible via a mobile shopping application that communicates with back-end application deployed on the PaaS cloud offering. The supermarket's goal is to collect the information about consumers' shopping behavior that results in the creation of a consumer profile. This profile could then be used to provide consumers more precise offers and bargains. Supermarket's business partners may also want to access this information in order to propose personalized offers to the mobile shopping application users themselves.

The back-end application for the supermarket loyalty program is developed using Java programming language and uses the cloud persistency service to store application data. The interface to access the persistency service is based on Java Persistence API (JPA)[6], which is nowadays one of the most common ways of accessing a relational database from Java code.

The supermarket employees can access detailed results of database queries regarding the consumers' shopping history and also create personalized offers, via a web-based portal. Moreover, the cloud application exposes web services through which third parties interact with the back-end system to consume collected data: both for their own business analysis, but also to contact directly the consumers for marketing purposes.

The interface for the consumers makes it possible to indicate privacy preferences with respect to the category of products (health care, food, drinks, etc) that one wants to share his shopping habits about. The consumer can also indicate whether he permits the supermarket to share personally identifiable information with its business partners, among other usages. This choices are then reflected by the private data access control mechanism that we will describe in Section 3.

2.2 Background: Privacy Policy Language

The users of the mobile shopping application are asked to provide various kinds of personal information, starting from basic contact information (addresses, phone, email) to more complex data such as shopping history or lifestyle preferences. Service providers describe how users' data are handled using a privacy policy, which is explicitly presented to users during the data collection phase.

In this paper we adopt the PrimeLife[7] Policy Language (PPL) [3], which extends XACML with privacy-related constraints for access and data usage.

[6] `http://docs.oracle.com/javaee/5/tutorial/doc/bnbpz.html`
[7] `www.primelife.eu`

```
- <ppl:DataHandlingPreferences>
   - <ppl:AuthorizationsSet>
      - <ppl:AuthzUseForPurpose>
            <ppl:Purpose>http://www.w3.org/2002/01/P3Pv1/individual-analysis</ppl:Purpose>
            <ppl:Purpose>http://www.w3.org/2002/01/P3Pv1/admin</ppl:Purpose>
            <ppl:Purpose>http://www.w3.org/2002/01/P3Pv1/contact</ppl:Purpose>
        </ppl:AuthzUseForPurpose>
        <ppl:AuthzDownstreamUsage allowed="false"/>
     </ppl:AuthorizationsSet>
   + <ob:ObligationsSet>
  </ppl:DataHandlingPreferences>
```

Fig. 2: Excerpt of a PPL policy rule

PPL policy is then used by the application to record its privacy policy. It states how the collected data will be used, by whom, and how it could be shared. On the other hand, the end-user also selects among the possible choices as to the conditions of the data usages, that are derived from privacy policies specific to the application. This user opt-in/opt-out choice is managed by the application and as such is not part of the generic enforcement mechanism developed by us. Before disclosing personal information, the user can match his preferences against the privacy policy of the service provider with the help of a policy matching engine. The result of the matching process is an agreed policy, which is then translated into the set of simple rules that are stored together with users' data inside the cloud platform's database servers.

In summary a PPL policy defines the following structures [3]:

- Access Control Elements: inherited from the XACML attribute-based access control mechanism to describe a shared resource (in our case PII) in general, as well as entities (subjects) that can obtain access to the data.
- Data Handling Preferences: expressing the purpose of data usage (for instance marketing, research, payment, delivery, etc.) but also downstream usage (understood here as sharing data with third parties, e.g. advertising companies), supporting a multi-level nested policy describing the data handling conditions that are applicable for any third party retrieving the data from a given service.
- Obligations: specify the actions that should be carried out with respect to the collected data, e.g. notification to the user whenever his data is shared with a third party, or deletion of the credit card number after the payment transaction is finished, etc. Obligations in PPL can be executed at any moment throughout whole lifetime of the collected data and can affect future data sharing transactions, e.g. with third parties.

An excerpt of a policy is shown in Figure 2. It shows part of a policy rule, stating the consent to use the data collected for three distinct purposes (described using P3P purpose ontology), but forbids downstream usage.

Consumer opt-in/opt-out choice is linked with PPL policy rule via XACML conditions that we adopted for this purpose. We have reused *EnvironmentAttributeDesignator* elements syntax to refer to the actual recorded consumer choice in the application data model, as shown in Figure 3. The location is

```
<xacml:Condition>
<xacml:Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:and">
    <xacml:Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
        <xacml:EnvironmentAttributeDesignator
                    AttributeId="CONSUMER_CONSENT:CATEGORY_ID"
                    DataType="http://www.w3.org/2001/XMLSchema#string" />
        <xacml:EnvironmentAttributeDesignator
                    AttributeId="PRODUCT_CATEGORY:CATEGORY_ID"
                    DataType="http://www.w3.org/2001/XMLSchema#string" />
    </xacml:Apply>
</xacml:Apply>
</xacml:Condition>
```

Fig. 3: Excerpt of a PPL policy condition

provided as the *AttributeId* value and can be read as `TABLE_NAME:COLUMN_NAME` of the database table where this choice is stored (`CONSUMER_CONSENT`) as well as a foreign key to the product category table (`CATEGORY_ID`) that is used to join products table. This information is used when enforcement mechanism is put in place to take consumer consent into account whenever information about consumer's shopping history (for certain product categories) is requested. This policy definition of how user consent is linked to the rest of the application data model is left in charge to the application developer as he is the one possessing full knowledge of the application domain.

3 Privacy Enhanced Application Programming

We have designed a framework able to modify, at the deployment time, the architectural elements (such as databases, web service frameworks, identity management, access control, etc) enriching it with the further components in order to enforce user privacy preferences. In this landscape the new applications deployed on the modified platform can benefit from privacy-aware data handling.

3.1 Programming Model

The privacy-aware components are integrated seamlessly with cloud application at the deployment time, so that the enforcement of privacy constraints is done afterwards automatically. They mediate access to the data sources, enforcing privacy constraints. In this case we are taking full benefit of the uniform database access in the PaaS landscape that is exposed via standard Java database interfaces such as JDBC (Java Database Connectivity) or JPA.

Usually the application code handling privacy related data is scattered and tangled over the application, being difficult to handle and to maintain if any changes in the privacy policy are introduced. As we observed in the existing applications the operations, which are performed on the private user data to ensure that privacy policies are enforced, are typically cross-cutting concerns in aspect-oriented programming paradigm. Inspired by this, we designed a process

```
1  @Entity
2  @PII
3  public class ShoppingHistory implements Serializable
4  {  ... ...  }
```

Fig. 4: JPA entity class annotation indicating persistency of private information

for the application developer that contributes to simplifying a way the data protection compliance could be achieved. It consists of adding meta-information to the application code via Java annotation mechanism in the JPA entity classes. Entity class in JPA terms is the one that is mapped into a database structure (usually a table, but also more complex type of mappings exist, e.g. to map object inheritance hierarchy) and enables the objects of that class to be persisted in a database. We provide also a second type of annotations, for the methods that make use of a private data, to indicate the purpose of the data usage.

The modifications to the code are non-intrusive, in the sense that the application business functions flow will stay exactly the same as before, except for the data set it will operate on, that will be obtained from database by adhering to the privacy policy. The changes are as transparent as possible from the application point-of-view as new platform components propose the same set of API as in the traditional platforms (in our case this API is JPA) and additional functionality is obtained via non-obtrusive code annotations that in principle could be easily removed in case described features are not required or not available.

This approach adds value with respect to legacy applications while allowing privacy management when needed. Another advantage is that the cloud service provider can easily move to another cloud platform without being locked into the certain vendor, apart from the fact that the guarantees given by the platform about private data handling could not be the same.

The platform we used to develop our prototype offers the enterprise level technologies available for Java in terms of web services and data persistency (JEE, JPA). In most of the examples we present along the paper we assume that the application developer will likely use a framework such as the JPA to abstract the database access layer.

In our approach developers are required to add annotations to certain constructs, such as *@PII* annotation in the JPA entity class (Figure 4). This annotation indicates that the class comprises one or more fields having private data (that usually are represented in database as columns) or that all fields are to be considered as PII (thus whole database table row needs to be either filtered or kept during the privacy enforcement, as JPA entity is by default mapped to a database row).

In the business code that is handling the private data we propose to use two other annotations to indicate class and method that processes PII sets. An example of annotated code is shown in Figure 5. In this figure the method annotation holds the information that the shopping history list items will be processed for marketing purpose.

```
5  @PiiAccessClass
6  public final class ShoppingHistoryDAO extends DaoImpl<ShoppingHistory>
7  {
8     ...
9     @Info(purpose="http://www.w3.org/2006/01/P3Pv11/marketing")
10    public final List<ShopHistory> getHistories()
11    { ... }
12    ...
13 }
```

Fig. 5: Annotating private data usage class with PII meta-information

In summary our library provides three different annotations:

***@PII*:** It is a flag to indicate personally identifiable information inside a JPA entity class definition. Such information is usually stored in a database as a table or a column. In Figure 4 this annotation involves the scope of the class declaration, see lines 2 and 3.

***@PiiAccessClass*:** This annotation should be put in the class to indicate where it contains access methods to personal data (see line 5 in Figure 5). We assume that PII access method performs queries to the database that are requesting private user data.

***@Info*:** This annotation is applied to PII access method, to describe the purpose or set of purposes of the query performed in that method (see lines 9 and 10 in Figure 5).

We expect the application developers to use this annotations to mark each usage of personal data as well as to indicate correct purposes. Ultimately they seek compliance to regulations, therefore we trust them to correctly indicate via the annotations the intended usage of the data. One can envisage that automated code scanners and manual reviews can take place during an audit procedure in order to check whether the annotations are rightfully used.

3.2 Implementation

In this section we detail the components of our prototype architecture. Technically our code components are packaged as several OSGi (Open Services Gateway initiative framework[8]) bundles. A bundle is a component which interacts with the applications running in the cloud platform. Some of them are to be directly deployed inside the PaaS cloud landscape and managed by the cloud provider while the other are part of the library to be used by the cloud application developers. Cloud providers can easily install or uninstall bundles using the OSGi framework without causing side effects to applications themselves (e.g. no application restart is required if some of the bundles are stopped). In the context of our scenario, we have three main bundles managed by the cloud provider

[8] `http://www.osgi.org`

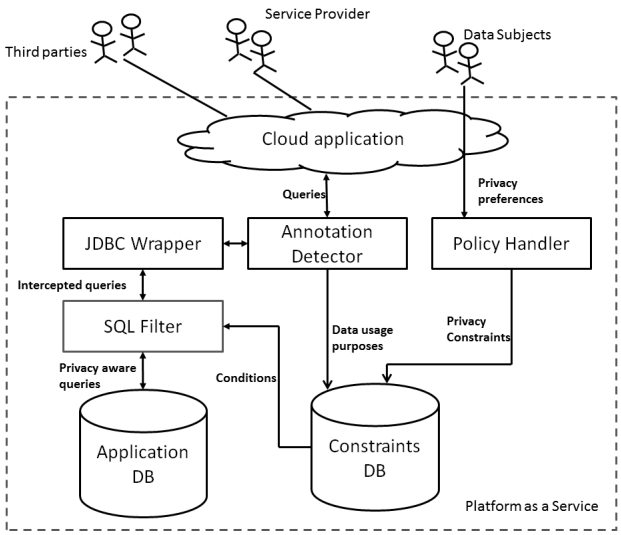

Fig. 6: Enforcement components

(JDBC Wrapper, Annotation Detector and SQL Filter) and one additional bundle (Policy Handler) that is providing a translation from an application privacy policy file written in the PPL language into an internal representation stored in the Constraints Database. The diagram in Figure 6 presents the architecture of the system, which we are going to describe in more details in the following subsections.

JDBC Wrapper The Wrapper intercepts all queries issued by the cloud application directly or by the third parties which want to consume the collected data containing shopping history of the fidelity program participants. This component is provided on the platform as an alternative to the default JDBC driver in order to enforce consumers' privacy preferences. Actually the wrapper makes use of the default driver to eventually send the modified SQL calls to database.

JDBC Wrapper bundle implements the usual interfaces for the JDBC driver and overrides specific methods important to the Java Persistence API, necessary to track the itinerary of SQL queries. As a matter of fact, it is wrapping all JDBC methods that are querying the database, intercepting SQL statements and enriching them with proper conditions that adhere to privacy policy (e.g. by stating in the `WHERE` clause conditions that refer to the consumer consent table). In order to identify the purpose of each query, its recipient and the tables referred, we retrieve the call stack within the current thread thanks to the annotations described in the previous section. We look for the PII access class, then we look for the method that sends the request to get the further parameters that help properly enforce privacy control.

Annotation Detector First task of this component is to scan Java classes at the deployment time and look for the JPA entities that are containing privacy-related annotation in its definition (`@PII`). List of such classes is then stored inside the server session context. Information about entities considered as PII is used to determine which database calls need to be modified in order to help preserve consumer privacy preferences.

In the second run the annotation detector scans the application bytecode in order to gather information concerning the operation that the application intends to perform on the data, annotated with `@PiiAccessClass` and `@Info` annotation. It is important to recall that the annotations are not a "programmatic" approach to indicate purpose, as they are independent from the code, which can evolve on its own. The assumption is that developers want to reach compliance, thus the purpose is correctly indicated, in contrast to [4], where it is assumed that end-users themselves indicate the purposes of the queries they perform. The cloud platform provider can instrument the annotation detector with a configuration file where the required annotations are declared. The detector can recognize custom annotations and stores information about related entity class in the runtime for future use.

SQL Filter This component allows us to rewrite original queries issued to the database by replacing the requested data set with a projection of that data set that takes into account consumers' privacy choices. SQL Filter modifies only the `FROM` part of a query, implementing an adapted version of the algorithm for disclosure control described in [12], also similar to the approaches described in [1], [13], and [16].

The query transformation process makes the use of the pre-processed decisions generated by the Policy Handler that concerns each possible combination of the triple purpose, recipient and PII table.

The transformation of the SQL query happens at the runtime. Consumer's privacy preferences are enforced thanks to the additional join conditions in the query, relating data subject consent, product category and filtering rules. The output is a transformed SQL query that takes into account all stated privacy constraints and is still compatible with the originally issued SQL query (it means that the result set contains exactly the same type of data, e.g. number of columns and their types). From a business use-case perspective, it was always possible to visualize relevant data, e.g. shopping history information, etc, without disclosing personal data when user didn't give his consent. The process is illustrated in Figure 7. It depicts the process of query modification when application is accessing data from the **SHOPPING_HISTORY** table (top-left corner of this figure). Original query (bottom-left) is transformed so that it takes into account the information derived from privacy policy that was put by the Policy Handler in the **CONSUMER_CONSENT** table (top-center). This table stores the association between the consumers and the different product categories with which these consumers opt to reveal their shopping history. Modified query (bottom-right) yields the data set of the same structure as original query but without

disclosing the information that consumers declined to share, as it can be seen in the **RESULT** table (top-right).

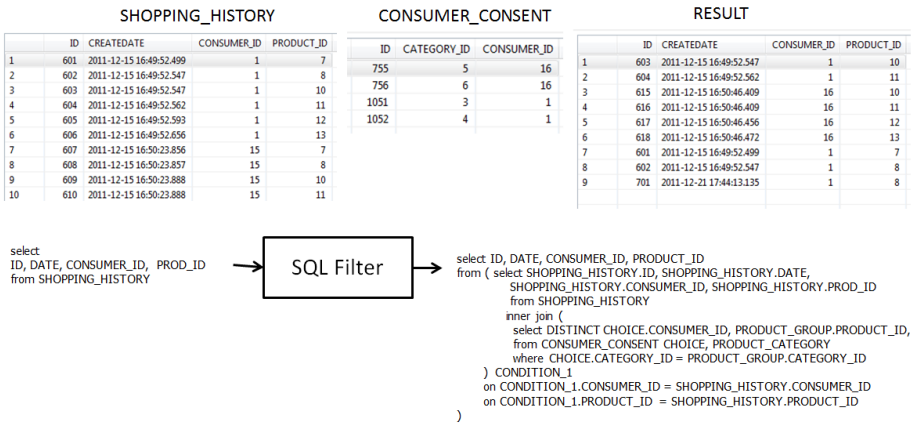

Fig. 7: SQL transformation example

The negotiated privacy policies are stored under the form of constraints together with the data in the database component provided by the cloud infrastructure. Whenever a query is launched by the application, we use the information collected by the annotation detector in order to modify queries on the fly, thus using the constraints to filter out the data that is not allowed to appear in the query results.

This approach is interesting because the behavior of the application itself is not modified. The impact on the performance of the system is minor, as the policy enforcement is actually pushed into a database query and also the complexity of this query transformation algorithm is low, as shown in previous works [12]. The work in [1] brings some performance evaluation for the same kind of transformations. We advocate that the ability to implement privacy controls is more important than these performance questions when dealing with private data in cloud computing.

4 Related Works

There are many similarities between our approach and the work described in [13]. It proposes a holistic approach for systematic privacy enforcement for enterprises. First, we also build on the state of the art access control languages for privacy, but here with an up-to-date approach, adapted for the cloud. Second, we leverage on the latest frameworks for web application and service development to provide automated privacy enforcement relying on their underlying identity management solutions. We also have similarities on the way privacy is enforced, controlling access at the database level, which is also done in [1].

Although the query transformation algorithm is not the main focus of our work, the previous art on the topic [6,16,4] present advanced approaches for privacy preserving database query over which we can build the next versions of our algorithm. Here we implemented an efficient approach for practical access control purposes, but we envisage to enrich the approach with anonymization in the future.

On the other hand, we work in the context of the cloud, where a provider hosts applications developed by other parties, which can in their turn communicate with services hosted in other domains. This imposes constraints outside of the control of a single service provider. We went further in the automation, by providing a reliable framework to the application developer in order to transfer the complexity of dealing with privacy preferences to the platform provider. Our annotation mechanism provides ease of adoption without creating dependencies with respect to the deployment platform. More precisely, no lock in is introduced by our solution. However, changes in the database schema that involves PII data require an application to be redeployed in the plataform in order to process the eventually new annotations.

The work in [11] presents an approach based on privacy proxies to handle privacy relevant interactions between data subjects and data collectors. Proxies are implemented as SOAP based services, centralizing all PII. The solution is interesting, but it is not clear how to adapt the proxy to specific data models corresponding to particular applications in a straightforward way.

Our work is aligned with the principles defended in [15], in particular we facilitate many of the tasks the service designers must take into consideration when creating new cloud-based applications. In [14], a user-centric approach is taken to manage private data in the cloud. Control is split between client and server, which requires cooperation by the server, otherwise obfuscated data must be used by default. This is a different point of view from our work, where we embed the complexity of the privacy enforcement in the platform itself.

Automated security policy management for cloud platforms is discussed in [10]. Using a model driven approach, cloud applications would subscribe to a policy configuration service able to enforce policies at run-time, enabling compliance. The approach is sound but lacks of fine-grained management for privacy policies, as it is not clear how to deal with personal data collection and usage control.

In [9], cryptographic co-processors are employed to ensure confidentiality of private data protection. The solution is able to enforce rather low level policies using cryptography as an essential mechanism, without explicit support to design new privacy compliant applications. Several works exist on privacy protection in Web 2.0 and peer-to-peer environments, such as in [18], where an access control mechanism is adopted for social networks. Some of these ideas can be reused in the context of cloud applications, but our approach differentiates from this line of work in the sense we empower the cloud applications developers with ready to use mechanisms provided directly by the cloud platform.

In [5], aspect-oriented programming is used as well to enforce privacy mechanisms when performing access control in applications. The work adopts a similar approach to ours, but privacy mechanisms are created in a per-application basis. In our approach, by targeting the platform as a service directly, we are able to facilitate enforcement in multiple applications.

5 Conclusion and Future Works

We presented a solution to simplify the process of enabling personal data protection in Java web applications deployed on Platform as a Service solution. We augment cloud applications with meta-data annotations and private data-handling policies which are enforced by the platform almost transparently from the developer perspective (the major overhead is only in placing the right annotations in the code).

The cloud consumer applications indicate how and where personally identifiable information is being handled. We adapt the platform components with privacy enforcement mechanisms able to correctly handle the data consumption, in accordance with an agreed privacy policy between the data subject and the cloud consumer.

The advantages of our approach can be summarized as follows: the implementation details of the privacy controls are hidden to the cloud application developer; compatibility with legacy applications, since the annotations do not interfere with the existing code; cloud applications can gracefully move to other platform providers that implement privacy-aware platforms in different ways. Sensible changes in the database schema, specifically those modifying PII, require the application to be redeployed in the cloud, possibly with new annotations.

Some future directions include the orchestration of other components such as event monitors, service buses, trusted platform modules, etc, in order to provide real-time information to users about the operations performed on their personal data. We plan to generalize our approach to enforce other kinds of policies, such as service level agreements, separation of duty, etc.

An important improvement of this work is the integration of advanced k-anonymization [17] process at the database access level. Such solution would be more adapted to business applications than access control, since the end-users could obtain more meaningful information, without fully disclosing the identities of the data subjects.

6 Acknowledgements

This work was supported by the CESSA Project - Compositional Evolution of Secure Software with Aspects - grant number 09-SEGI-002-01 - from the French National Research Agency (ANR). Many thanks to Theodoor Scholte for his valuable comments on a previous version of this paper.

References

1. Agrawal, R., Kiernan, J., Srikant, R., Xu, Y.: Implementing p3p using database technology. In: Data Engineering, 2003. Proceedings. 19th International Conference on. pp. 595 – 606 (march 2003)
2. Ashley, P., Hada, S., Karjoth, G., Powers, C., Schunter, M.: Enterprise privacy authorization language (epal). Research report 3485 (2003)
3. Bussard, L., Neven, G., Preiss, F.S.: Matching privacy policies and preferences: Access control, obligatons, authorisations, and downstream usage. In: Camenisch, J., Fischer-Hbner, S., Rannenberg, K. (eds.) Privacy and Identity Management for Life, pp. 117–134. Springer Berlin Heidelberg (2011)
4. Byun, J.W., Bertino, E., Li, N.: Purpose based access control of complex data for privacy protection. In: Proceedings of the tenth ACM symposium on Access control models and technologies. pp. 102–110. SACMAT '05, ACM, New York, NY, USA (2005)
5. Chen, K., Wang, D.W.: An aspect-oriented approach to privacy-aware access control. In: Machine Learning and Cybernetics, 2007 International Conference on. vol. 5, pp. 3016 –3021 (aug 2007)
6. Cohen, S., Nutt, W., Serebrenik, A.: Rewriting aggregate queries using views. In: Proceedings of the eighteenth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems. pp. 155–166. PODS '99, ACM, New York, NY, USA (1999)
7. Cranor, L.: P3p: making privacy policies more useful. Security Privacy, IEEE 1(6), 50 – 55 (nov-dec 2003)
8. Idrees, M.S., Serme, G., Roudier, Y., de Oliveira, A.S., Grall, H., Südholt, M.: Evolving security requirements in multi-layered service-oriented-architectures. In: García-Alfaro, J., Navarro-Arribas, G., Cuppens-Boulahia, N., di Vimercati, S.D.C. (eds.) DPM/SETOP. Lecture Notes in Computer Science, vol. 7122, pp. 190–205. Springer (2011)
9. Itani, W., Kayssi, A.I., Chehab, A.: Privacy as a service: Privacy-aware data storage and processing in cloud computing architectures. In: DASC. pp. 711–716. IEEE (2009)
10. Lang, U.: Openpmf scaas: Authorization as a service for cloud & soa applications. In: CloudCom. pp. 634–643. IEEE (2010)
11. Langheinrich, M.: A privacy awareness system for ubiquitous computing environments. In: Borriello, G., Holmquist, L. (eds.) UbiComp 2002: Ubiquitous Computing, Lecture Notes in Computer Science, vol. 2498, pp. 315–320. Springer Berlin / Heidelberg (2002)
12. LeFevre, K., Agrawal, R., Ercegovac, V., Ramakrishnan, R., Xu, Y., DeWitt, D.J.: Limiting disclosure in hippocratic databases. In: Nascimento, M.A., Özsu, M.T., Kossmann, D., Miller, R.J., Blakeley, J.A., Schiefer, K.B. (eds.) VLDB. pp. 108–119. Morgan Kaufmann (2004)
13. Mont, M.C., Thyne, R.: A systemic approach to automate privacy policy enforcement in enterprises. In: Danezis, G., Golle, P. (eds.) Privacy Enhancing Technologies. Lecture Notes in Computer Science, vol. 4258, pp. 118–134. Springer (2006)
14. Mowbray, M., Pearson, S.: A client-based privacy manager for cloud computing. In: Bosch, J., Clarke, S. (eds.) COMSWARE. p. 5. ACM (2009)
15. Pearson, S., Charlesworth, A.: Accountability as a way forward for privacy protection in the cloud. In: Jaatun, M.G., Zhao, G., Rong, C. (eds.) CloudCom. Lecture Notes in Computer Science, vol. 5931, pp. 131–144. Springer (2009)

16. Rizvi, S., Mendelzon, A., Sudarshan, S., Roy, P.: Extending query rewriting techniques for fine-grained access control. In: Proceedings of the 2004 ACM SIGMOD international conference on Management of data. pp. 551–562. SIGMOD '04, ACM, New York, NY, USA (2004)
17. Sweeney, L.: k-anonymity: A model for protecting privacy. International Journal on Uncertainty Fuzziness and Knowledgebased Systems 10(5), 557–570 (2002)
18. Tootoonchian, A., Saroiu, S., Ganjali, Y., Wolman, A.: Lockr: better privacy for social networks. In: Liebeherr, J., Ventre, G., Biersack, E.W., Keshav, S. (eds.) CoNEXT. pp. 169–180. ACM (2009)