Automating the Generation of Privacy Policies for Context-sharing Applications

Wolfgang Apolinarski, Marcus Handte, Pedro José Marrón
Networked Embedded Systems
Universität Duisburg-Essen
Essen, Germany
firstname.lastname@uni-due.de

***Abstract*—Enabling the automated recognition and sharing of a user's context is a primary motivation for many pervasive computing applications. In the past, a significant amount of research has been focusing on the aspect of effective and efficient recognition. Yet, when context is shared with others, the resulting disclosure of personal information can have undesirable privacy implications. A common solution to this problem is the manual creation of an application-specific privacy policy that defines which information may be shared with whom. However, as the number of applications increases, such a manual approach becomes increasingly cumbersome and over time, it is likely to lead to incomplete or even inconsistent policies. In this paper, we discuss how a privacy policy can be derived automatically by analyzing the user's sharing behavior when using online collaboration tools. Our approach retrieves shared content and the associated sharing settings, detects context types and automatically derives a privacy policy that reflects the user's past sharing behavior. To validate our approach, we have implemented it as an extensible software library for the Android platform and we have developed plug-ins for two popular collaboration tools, namely Google Calendar and Facebook.**

***Keywords*-Privacy policy, online services, policy derivation, context-sharing**

I. Introduction

Nowadays, more and more pervasive computing, crowdsourcing or participatory sensing applications share context among their users [1], [2], [3]. Also commercially available applications (e.g., Windows Media Player) may share context among their users, for example the song they are currently listening to (e.g., using the Skype or Facebook status message). In the past, the research focus usually lay on the *efficient recognition* of context information. Therefore, privacy concerns were often underestimated, with the major exception of location privacy [4], [5]. Beside location privacy, also the sharing of other types of context information, especially the disclosure of personal context can have undesirable privacy implications. While a common solution to this problem is the manual creation of a privacy policy, this has several drawbacks. The created privacy policy will be application specific, i.e., not shared between different pervasive computing applications. As a result, the user needs to define a privacy policy for each application. This is cumbersome and can result in an incomplete or inconsistent policy since the user must manually maintain all privacy policies for different context-sharing applications.

Additionally, the user defines several privacy policies when using online collaboration tools such as Facebook or Google Calendar while sharing content with other users. Here, the user usually defines with whom the content should be shared manually, either by adding individual users, predefined user lists or individually inviting users to events. So, on the one hand, the user already defines an implicit privacy policy for her content. On the other hand, the shared content contains information about the context of a user, for example the current context (e.g., status messages) or even a future context (e.g., events in a calendar). If the context type is now recognized, it is possible to combine sharing settings and context types to derive a policy that can be applied to context-sharing pervasive computing applications.

In this paper, we discuss how a privacy policy can be derived automatically by analyzing the user's past sharing behavior when using online collaboration tools like Facebook or Google Calendar. Our approach retrieves shared content and the associated sharing settings, detects context types and automatically derives a privacy policy that reflects the user's past sharing behavior. Context-sharing applications can then use this privacy policy directly or offer it to the user as a basis for further customization. In addition, they can frequently update the policy in order to avoid conflicts and minimize inconsistencies. To validate our approach, we have implemented it as an extensible software library for the Android platform and we have developed plug-ins for two popular collaboration tools; Google Calendar and Facebook. To demonstrate and evaluate the library, we discuss its use in a location sharing application.

The reminder of the paper is structured as follows. In Section II, we discuss our approach for a framework which automates the privacy policy generation for context-sharing applications. Then, we present our prototypical implementation that detects the context type *location* and uses plug-ins to derive a privacy policy from Google Calendar and Facebook. In Section IV, we show our location sharing application which benefits from the automatic generation of a privacy policy. In Section V, we evaluate our approach using the prototypical application. Finally, we discuss related work in Section VI and conclude the paper in Section VII.

II. Approach

Our goal is to derive a privacy policy from the settings defined in an online collaboration tool like Facebook or Google Calendar. This allows users to *re-use* their previously specified privacy settings for the use with context-sharing applications. Often, users already define their individual privacy needs in online collaboration tools, for example by adding sharing settings to a (shared) resource (e.g., a status message, photo, event). This is usually accomplished by manually choosing the users with whom a resource should be shared. Additionally, user groups may be predefined to ease this process, e.g., allowing to choose a predefined group named *family* instead of selecting each individual family member. We assume that these sharing settings can be translated to context sharing settings for applications, effectively forming a privacy policy. Our goal is now to use the resource and the associated sharing setting to derive a general, context-dependent privacy policy for the use with pervasive computing, context-sharing applications.

A. System Model

For the derivation of privacy policies from online collaboration tools, we require the technical basis described here. Users are sharing *resources* by specifying *sharing settings* in online collaboration tools. Effectively, it is possible to derive *context types* from a resource that could be used by context-sharing applications. The resources are shared with other users using a *network*. Regarding each individual building block, this is the system model for the derivation of a privacy policy from online collaboration tools:

- Resource: Resources can be shared between users. A resource can be a status message, a photo or similar content shared using an online collaboration tool. Often, a resource describes parts of the current situation of a user.
- Sharing setting: Each resource contains sharing settings that constrain the access to a set of users (e.g., *friends* in Facebook). This setting can be specified by the user. It can be configured individually for each resource.
- Context Type: Since resources in online collaboration tools often describe details of the current situation of a user, they may be used to derive their context types. An example would be a status message resource where the user reveals her current position. Although the type of the resource (e.g., image or text) might influence the context type, there is no direct connection.
- Network: The resource and the associated sharing settings must be accessible remotely over a network such as the Internet. For this purpose, online collaboration tools usually exhibit an API which can be used by third party tools.

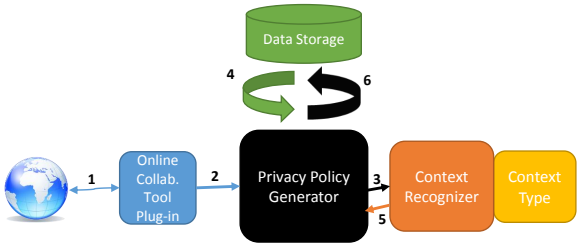


Figure 1. Privacy Policy Generation Framework Architecture & Data Flow

B. Design Rationale and Goals

The overall goal is the derivation of a user-specific privacy policy by analyzing shared resources in online collaboration tools. Therefore, we design a framework for the policy derivation which should be widely applicable. To achieve this, we define the following design goals.

- Generic: Pervasive computing applications use many different kinds of context information. As a result, the framework should be generic with regard to context types. It should support any type and format of context. Additionally, different context scopes (e.g., for location: country, city, street) should be supported.
- Extensible: Beside widely-used social networks like Facebook or online business tools like Google Calendar, there exist many other online collaboration tools. The framework should be extensible such that it is possible to extend the existing framework to support these other tools.
- Automation: In online collaboration tools, users already specify their privacy needs manually using sharing settings. The derivation of the privacy policies should therefore run fully automatic. Additionally, it should support the detection of conflicting sharing settings and present them to the user.
- Low Overhead: Manually defining sharing settings in online collaboration tools already needs the user's time and attention. The privacy policy generation should have a low overhead in terms of data transfer and the time that it takes to derive a policy from a resource.

C. Privacy Policy Generation Framework

The privacy policy generation framework consists of several components that are combined to derive a privacy policy from online collaboration tools. All components and their interactions are depicted in Figure 1. The privacy policy generator has a plug-in interface for online collaboration tools and another interface for context recognizers. Additionally, a data storage is connected to the privacy policy generator to store the derived privacy policy. This allows the framework to support both, different types of online collaboration tools as well as different context types. Using this architecture, the privacy policy generator does not need

to know any details about the collaboration tool plug-ins, while the context types (and the associated recognizers) are registered with the generator.

The policy generation can either be executed one-time (e.g., at the first start of a pervasive computing application) or regularly (e.g., to re-check the validity of the current privacy policy). The generation is started when a plug-in inputs a resource and its associated sharing settings into the privacy policy generator. The generator then utilizes the registered context types and their recognizers to detect context types from that resource. If the context recognizers detect a context type with a certain probability (definable both by the recognizer and the policy generator), the type and the sharing settings are transformed into a privacy policy and stored in the data storage. If a previous policy for the same context type contradicts the new policy, the new policy is not stored, but can be presented to the user to resolve the conflict. This ensures that only conflict free privacy policies are stored in the data storage. In detail, the privacy policy generation data flow looks as follows (see also Figure 1).

1) Resource Retrieval: A plug-in uses the API of an online collaboration tool to retrieve resources and associated sharing settings.
2) Resource Preparation: The plug-in prepares the context type detection by adding meta information to the resource (possible context types, remove redundant users from the sharing settings). The prepared resource and the sharing settings are then transferred to the privacy policy generator.
3) Context Detection: The privacy policy generator receives the resource and sends it to all (relevant) context recognizers. Each recognizer detects the context type (e.g., by using data mining techniques) and associates a probability with a (possible) detection. A recognizer is not constrained to one context type, but can recognize several different types.
4) User Matching: In parallel to the context type detection, the privacy policy generator matches the users contained in the sharing settings with the user database of the pervasive computing application. Users that are not in this database are removed from the settings.
5) Privacy Policy Generation: The privacy policy generator gets the result from the context recognizers, uses the context type with the highest probability (if it is above a predefined threshold) and generates a privacy policy from the context type and the sharing settings.
6) Storing the Policy: The generated privacy policy is now transferred to the data storage. If the policy conflicts with an already stored policy, the storage marks it as ”preliminary” and asks the user to perform a conflict resolution for the newly added, conflicting context types. Preliminary policies are not evaluated during normal operation, i.e., the used privacy policy is always free of conflicts.
   If there are no conflicts, or the conflicts were resolved, the privacy policy is stored in the data storage and will be queried when applications want to share context.

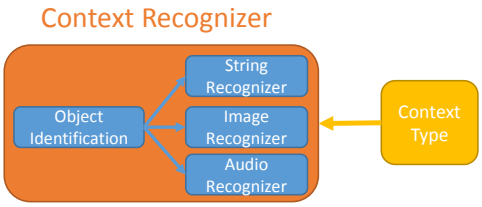


Figure 2. Context Type with Context Recognizer

The successful execution of these steps results in the generation of a privacy policy containing the context type(s) published at a particular online collaboration tool and the intersection of the users that (a) use the pervasive computing applications and (b) are mentioned in the sharing setting(s).

The steps can be executed several times for each online collaboration tool. Each execution will then extend the policy for an existing context type or add new context types to the policy. The Steps 3 and 4 can be executed in parallel, reducing the time for the policy generation. Similarly, all steps can be parallelized with different plug-ins. Only Step 6 must be synchronized between all parallel instances to keep the privacy policy in the storage conflict free.

III. Implementation

Validating the concepts of our approach, we have implemented it as an extensible framework. The core part is developed as a library on the Android operating system. Android was chosen since it is an operating system for mobile devices (such as smartphones) that are used by pervasive computing applications. User interfaces are implemented using so-called activities, computations in the background can be either implemented as services (long-running) or as asynchronous tasks (short running, result usually changes the user interface). The prototypical implementation focuses on the context type *location*, but is designed generic which allows support for other context types. Additionally, the extensible implementation includes two plug-ins for popular online collaboration tools. The first plug-in uses *Google Calendar* and evaluates shared events and sharing settings for the calendar, while the second plug-in analyzes status messages of users in *Facebook*.

A. Context Types and Recognition

As can be seen in Figure 2, a context type is always associated with at least one context recognizer. This allows the framework to recognize context types of arbitrary resources (such as a status message or an image). As a result, context types without recognizer cannot be derived from resources and are therefore not considered by the framework.

```
Permission {
    User 616
    Context type location
    Scope 0
}
```

Figure 3. A Permission of the Privacy Policy

In general, a context type usually describes a single context like *location* and also allows to define scope levels (such as *country*, *city* or *street* using the *location* context type). This allows an application to share context on different scopes.

For the privacy policy generation, the context recognizers perform an important task, the recognition of the context type from a (Java) object (the resource) that is passed on by a caller. While this paper does not present context recognizers, it is possible to add several recognizers to the framework. Possible examples are the NARF activity recognition framework [6] or other stand-alone (not server-based) recognizers like CenceMe [1] that can be executed on Android devices. To fulfill the context type recognition task, each recognizer will first identify if the object type is supported and execute the appropriate recognizer component. The example depicted in Figure 2 supports string, image and audio contents. After the appropriate context recognizer is executed, it will assign a probability (which can be 0%) to each object type that was passed on. The probability will then be reported back to the caller.

Eventually, the caller (usually the privacy policy generator) can evaluate all answers received from the context recognizers. Then, the caller decides, based on the individual probabilities, which context type should be assigned to the resource, if any.

B. Privacy Policy Generator Library

The privacy policy generator library is the central element of the privacy policy generation framework. The data flow to and from the generator is depicted in Figure 1 and described in Section II-C. To be as flexible as possible, each online collaboration tool plug-in can trigger the privacy policy generator and start the policy generation. This can also be done in parallel, only the access to the common data storage component must be synchronized to provide a consistent view on the stored data.

The library uses then the transferred sharing settings and reads out the individual users. Hereby, it only extracts users that are allowed to access a shared resource. The reason for this is that our current privacy policy is using a binary grant/deny access scheme on context types. Of course, this model can be extended with a more complex scheme (e.g., involving the time of the day when access to certain context types is granted). After the individual users are read out, a user matching is performed. The user matching process is twofold. At first, only users whose unique account identification for an online collaboration tool (e.g., the Facebook id) that is already known to the framework are considered. In a second step, we match the user name and e-mail address. In the future, we plan to extend this with a more sophisticated user matching process involving multiple attributes (e.g., location, birthday, etc.) to match users even when other attributes (like the e-mail address) are not publicly available. This process shows that it is possible to match users even when few attributes are available and it has also proven to be effective [7]. In the end, all matched users are added to a preliminary privacy policy permission object.

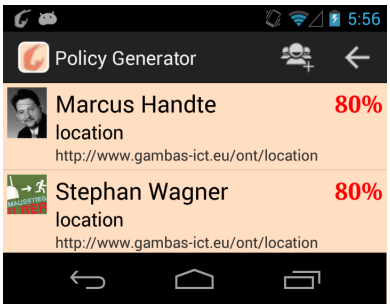


Figure 4. A Screen Showing the Generated Permissions

In parallel, the privacy policy generator uses the context recognizers to detect context types from the transferred resource. Both information, the detected type and the users are then combined to one policy element, in our implementation called *permission*. An example can be seen in Figure 3. All permissions together form the privacy policy of the device.

At the end, the library stores these permissions in the data storage (usually after user approval, see Figure 4), adding them to the privacy policy. The screen depicted here suggests the user that the context type *location* should be shared with two users. As an additional information, the context detection mechanisms provided a probability of 80%. Conflicting permissions are not added to the policy. Instead, the user is presented a screen to choose which permission should be added. The privacy policy generated by the library can then be accessed through the data storage by other (context-sharing) applications.

C. Google Calendar Plug-in

Google calendar is a popular tool for sharing calendars between users. It allows to set different levels of calendar sharing. A user may choose to only share her free/busy state, but can also share the calendar completely, including the ability to create appointments in the calendar.

Additionally, the appointments in the calendar contain fields that may exhibit different context information. The *where* field for example indicates that location information is shared, the guest list shows not only with whom this appointment is shared, but could also reveal if this is a private or a work-related appointment.

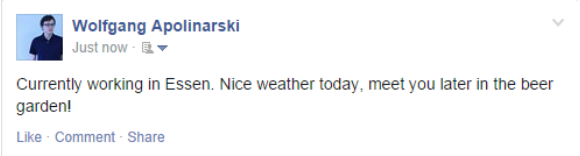


Figure 5. A Status Message on Facebook


```
"privacy": {
            "description": "Marcus Handte, Stephan Wagner",
            "value": "CUSTOM",
            "friends": "SOME_FRIENDS",
            "networks": "",
            "allow": "100001057542495,645219883",
            "deny": ""
 },
"message": "Currently working in Essen. Nice weather today,
meet you later in the beer garden!",
"id": "100001039541728_819141028130541",
"created_time": "2014-09-04T14:20:26+0000"
```


Figure 6. Graph API Representation of Figure 5

The Google calendar plug-in uses Google's API to extract information and sharing settings from the appointments and sends it to the privacy policy generator library. For fields with known context types (such as the *where* field), the plug-in defines the context types that are going to be detected by the library. For other fields (like *description*), the library determines the context types (if any). For all detected types, the library then extracts the set of users the types are shared with and adds the derived permissions to the privacy policy.

D. Facebook Status Messages Plug-in

Another popular online collaboration tool is Facebook. It is used mostly for private interactions and supports the creation of small statements that describe the current status of a user, so-called *status messages* (see Figure 5). Since the messages are usually describing the current situation of a user, they can often be used to derive context information.

Additionally, the user creating the status message can constrain access to these messages by providing sharing settings. Each status message may be shared with a different set of users (called *friends* in Facebook) or lists of users (*friendlists*). The user can also define a default sharing setting that will apply, if she did not specify a custom setting.

The Facebook status messages plug-in uses the Facebook Graph API and retrieves status messages from Facebook. An example JSON response by Facebook can be seen in Figure 6. Additionally, the plug-in extracts friends that have access to these messages from the sharing settings and also processes friendlists. Each status message and its associated sharing setting can be transferred to the privacy policy generator library to detect possible context types and add them, together with the set of users that is allowed to access these types, as a permission to the privacy policy.

IV. Application

To validate our approach, we have implemented a location sharing application called *Locator*. The application works similar to the localization components of *LifeMap* [8]. It performs the location detection using three different localization methods. It executes a WiFi scan, uses the GSM/UMTS network and GPS, depending on the current (detected) status of the user (e.g., indoor, moving or outdoor). The combination of these three different localization methods allows the Locator application to perform an energy-efficient localization of the user. Additionally, Locator supports different scopes for the context location. E.g., instead of showing the actual GPS coordinates of the position of a user, it is possible to only show the current city. Locator uses the geo-coding abilities of Google's location service to obtain the address and then extracts parts of the address that coincide with the scope the user specified.

In addition to the LifeMap features, Locator allows to share the location between its users, creating an application which works similar to the Google application *Latitude*[1]. The application itself builds on the GAMBAS middleware [9] which provides Locator with a query processor that is used to query for the location of other users. More details on the GAMBAS middleware and SDK can be found in [9].

Since location sharing is sensitive for most users, Locator requires user consent when sharing the current location information. As with similar context-sharing applications, user consent is obtained by a privacy policy that must be edited by the user. The privacy policy generation framework allows to automate this step. When a user now wants to use Locator instead or in addition to manually written status messages to share her current location (see Figure 5), she can either manually edit her privacy policy and allow sharing her location with another user, or, use the framework to automatically generate privacy policies (see Figure 4) which (after user approval) allow the sharing according to previous sharing behavior in online collaboration tools. As soon as the policy is applied, it is possible for users that were included in the policy to retrieve the user's location (see Figure 7(a)). Additionally, Locator allows a user to view her own trip history (see Figure 7(b)). Since the trip history is only stored on the user's own device, no privacy policy applies in this case. Similar to Google Latitude, Locator does not store the location history of other users, for privacy reasons.

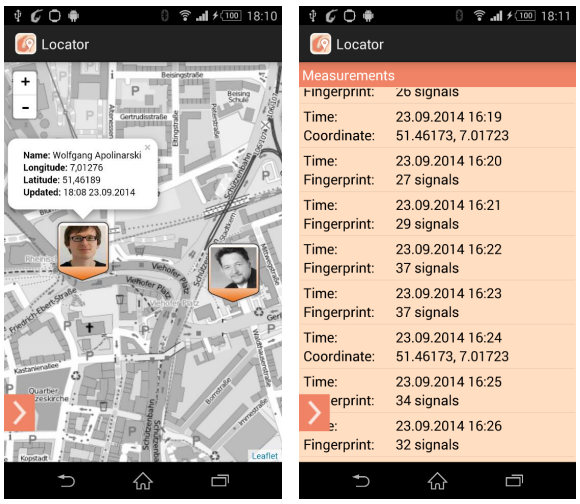
V. Evaluation

In this section, we evaluate our approach for the automatic generation of privacy policies. We concentrate on the four design goals that we introduced in Section II-B.

A. Generic

Our approach for the automatic generation of privacy policies does not constrain the type of context, i.e., it is possible to use it with every type of context. Similarly, the approach does also not constrain the recognizer in any

[1]Google Latitude was retired on August 9th, 2013. A similar feature is now offered to users of the social network Google+.



(a) Location Sharing (b) Trip History

Figure 7. The Locator location-sharing Application

way. A context recognizer could support the recognition of context types from any kind of data. Although context types that cannot be recognized by a recognizer are supported, the generated privacy policy will never include these types of context, since they could not be detected. These kinds of unrecognizable context types can be added to the policy manually by the user, though.

Additionally, the privacy policy generation framework does allow for a context type to have different scopes. If supported by the context recognizer, the generated privacy policy contains the proper scope for a recognized type. So, it is possible to use an arbitrary ontology that defines different context types and scopes like the CoBrA ontology [10].

B. Extensible

There are several providers with resources that can be used to derive a privacy policy from them. In our implementation, we were using Facebook status messages and Google Calendar events. Additionally, the framework can be extended with other plug-ins. Each plug-in can use one or more resource types from an online collaboration tool for the policy derivation. Only two kinds of information must be accessible and delivered to the framework: (a) the contents of the resource and (b) the users with whom the resource is shared. Using this information, the framework can execute the context recognizers and create permissions for the context types. Additionally, a plug-in can also deliver a list of possible context types that the resource can exhibit. This might speed up the context recognition process since the framework will then concentrate the recognition on a subset of all possible context types.

In summary, the framework can be extended easily by plug-ins for other online collaboration tools. Additionally, we argue that the threshold for the creation of such plug-ins is low since the plug-ins only need to extract few information from the online collaboration tools.

	Mean (in ms)	Standard deviation (in ms)
Facebook Plug-in	868 ms	79.4 ms
Context Recognition (location)	0.60 ms	0.081 ms
Privacy Policy Generation	52 ms	16.4 ms

Figure 8. Measuring the Privacy Policy Generation depicted in Figure 1

C. Automation

The generation of a privacy policy for context-sharing applications is executed fully automatic. All steps depicted in Figure 1 can be performed automatically. Plug-ins for the framework only need user interaction when the user authorizes the application. The type of authorization depends on the online collaboration tool, but often involves an OAuth-like [11] process. The generation of the privacy policy itself is executed without any user interaction. When a conflict-free privacy policy is generated, it could be applied to the context-sharing applications automatically.

Because of the possible unreliability of context type detection and sharing settings chosen by mistake, the generated privacy policy should be reviewed by the user. This would introduce a manual step, before the policy is applied and stored. Similarly, if a conflict is detected, the user would need to manually resolve it. The framework can prepare possible solutions, but the user needs to choose the right one. This manual step is inevitable to ensure that the user really agrees to the generated policy. In our implementation, as can be seen in Figure 4, we visualize the generated permissions (which form the privacy policy), helping the user in validating the automatically generated privacy policy.

D. Low Overhead

The overhead of the automatic generation of a privacy policy depends heavily on the used plug-ins for the online collaboration tools and the context recognizers. If, for example, a plug-in transfers high resolution images and let the context recognizer perform a complicated image processing task, the overhead is much higher than when only text is transferred and a relatively simple word processing task is executed. When complicated processing tasks are executed, the framework can ensure that the generation of the privacy policy is only executed when (a) WiFi is available (to avoid exhausting the data plan) and (b) the smart phone is not used (e.g., at night time) and charging.

To measure the framework's overheads, we follow the steps shown in Figure 1. At first, we measured our Facebook plug-in (retrieving status messages, Step 1 and 2), then our recognizer (detecting the context type *location* using a word list with more than 26500 cities, Step 3) as well as the complete time, executing the policy generator and

the recognizer (Steps 3 to 5), computing and displaying the generated policy to the user. Our measurement setup consists of one Nexus 5 smart phone that is using a dedicated WiFi network. We performed each measurement 100 times and computed the average as well as the standard deviation. The results can be seen in Figure 8. The total time for Steps 1 to 5 stays well below $1000ms$ ($868ms + 52ms$), with a standard deviation of $\sigma = 95.8ms$ ($79.4ms + 16.4ms$).

As can be seen easily in Figure 8, the main latency ($868ms$) is introduced by the online social collaboration tool plug-in. The execution of the privacy policy generator only takes $52ms$, including the context recognition of $0.60ms$.

Regarding the amount of transferred data, we use the message that is displayed in Figure 5 as a reference for an average status message. Facebook status messages can be longer, but often tend to stay well below the maximum length of a Twitter message (140 characters). As can be seen in Figure 6, the sharing settings are often longer than the actual message. The status message presented here uses a data volume of 383 bytes. This results in a very low data volume, even if the user has many recent status messages.

Although the actual data volume and execution time depends on the used plug-ins and context recognizers, we conclude that the components we implemented in our framework only introduce a low overhead.

VI. Related Work

The privacy policy generator is a continuation of the idea to use online collaboration tools for pervasive computing applications. In our previous work PIKE [12], we used the tools to exchange cryptographic keys which allowed data-sharing over an encrypted and authenticated communication channel. In contrast to that, we exploit the sharing of information (i.e., context) with users of online collaboration tools here. This work is focused on the privacy aspects, i.e., which kind of data should be shared with whom. Both approaches are therefore orthogonal to each other and can be combined to achieve a high level of security and privacy.

There are more and more pervasive computing applications that share context information. An example is the CenceMe [1] application which shares detected context. While currently these applications usually post directly to the online collaboration tools and use a user-defined policy for each of them, the privacy policy generator presented here allows them to use one consistent privacy policy for multiple online collaboration tools with several context types.

Several papers describe approaches to create or extend a privacy policy language [13], [14], [15]. Our currently used simple policy language can be replaced with such a language. This would also allow to describe more complex situations, e.g., if a context should be shared only in a specific time-slot. In this paper, we do not focus on the privacy policy language itself, but on the possibility to derive a privacy policy (or parts of it) from online collaboration tools. The privacy policy generation framework can export or convert the created policy in other policy languages, if necessary. Nevertheless, the simple policy that is generated currently can be used directly by pervasive computing applications as demonstrated in Section IV.

Toch et al. [16] and Fang et al. [17] describe that it is difficult for users to adjust their privacy settings to their needs properly. As a consequence, they try to help them and ease the process of privacy setting creation. Fang et al. [17] create a privacy wizard that can be trained by the user and allows to configure (at least parts of) the user's privacy settings automatically. Toch et al. [16] analyzes and clusters existing privacy settings, allowing a new user to choose from a popular set of privacy settings instead of starting from scratch. This allows users to set their privacy settings correctly, which is mandatory for our framework to be used reliably. Both approaches do not extract a privacy policy from the online collaboration tools or detect the context type that is shared, but they can be combined with our work which might result in a better fit of the generated privacy policy.

There also exists related work that is deriving privacy policies from online collaboration tools [18], [19], [20]. Danezis [18] is grouping users according to their mutual relationships and because of this relationship, a context is assigned to them. In contrast to our work, the context describes the group and not the type of data that is shared. Additionally, the user specified sharing settings of shared resources are not taken into account. Vyas et al. [19] tries to automatically manage privacy for different types of content, similar to our approach. Instead of using machine learning techniques to derive the context information, they require the user to manually assign *tags* to the content they are going to share. While this will work on user-published content like a Blog post, average users of online collaboration tools might not be willing to tag all their posts or events. Toch [20] describes how privacy preferences can be crowd-sourced using a crowdsourcing framework called *Super-Ego*. Beside crowdsourcing privacy preferences, they also predict preferences (using the crowd sourced data). In contrast to our work, they are using a centralized server to store manually made privacy preference decisions in a so-called crowd model. In this paper, we use several context sources from online collaboration tools, not using the crowd, but only one, individual user. We store the generated privacy policy exclusively on the user's own device, making it available to other pervasive computing applications executed by the user.

VII. Conclusion

Context-sharing applications have become more and more important in the domain of pervasive computing. Often, this has undesirable privacy implications which can be mitigated by defining a privacy policy for each application. Additionally, the user is already defining privacy policies

for shared content in online collaboration tools such as Facebook or Google Calendar. These policies can be used for the automated generation of privacy policies for context-sharing applications. The generated policies are then based on the user's previous sharing behavior. This eases the process of defining privacy policies which is often cumbersome for the user. Therefore, we have developed a tool that automatically generates privacy policies out of sharing settings used in online collaboration tools. Frequent re-runs of the tool update the policy, avoid conflicts and minimize inconsistencies. In this paper, we have shown that our concept is feasible and can, through the support of different online collaboration tools and context types, be extended easily to a wide range of pervasive computing scenarios.

Our tool, the privacy policy generator, is currently being integrated into the GAMBAS middleware. The generated policy will then be used by all context-sharing applications that are built on the basis of the GAMBAS SDK. With regard to the privacy policy generation, we are planning to attach the NARF activity recognition system [6] to the generator so that we are able to recognize more context types. Additionally, we want to provide better support to users who use only one collaboration tool account for professional and private life.

Acknowledgments

This work is supported by UBICITEC e.V. (European Center for Ubiquitous Technologies and Smart Cities), GAMBAS (Generic Adaptive Middleware for Behavior-driven Autonomous Services) and BESOS (Building Energy Decision Support Systems for Smart Cities) funded by the European Commission under FP7 with contract numbers FP7-2011-7-287661 and FP7-SMARTCITIES-2013-608723.

References

[1] E. Miluzzo, N. D. Lane, K. Fodor, R. Peterson, H. Lu, M. Musolesi *et al.*, "Sensing meets mobile social networks: The design, implementation and evaluation of the cenceme application," in *Proc. of the 6th ACM Conf. on Embedded Network Sensor Systems*, ser. SenSys '08. ACM, 2008.

[2] P. Shankar, Y.-W. Huang, P. Castro, B. Nath, and L. Iftode, "Crowds replace experts: Building better location-based services using mobile social network interactions," in *Perv. Comp. and Comm. (PerCom), 2012 IEEE Int. Conf. on*.

[3] D. Philipp, P. Baier, C. Dibak, F. Dürr, K. Rothermel, S. Becker *et al.*, "Mapgenie: Grammar-enhanced indoor map construction from crowd-sourced data," in *Perv. Comp. and Comm. (PerCom), 2014 IEEE Int. Conf. on*, March 2014.

[4] M. Wernke, F. Dürr, and K. Rothermel, "Pshare: Position sharing for location privacy based on multi-secret sharing," in *Pervasive Computing and Communications (PerCom), 2012 IEEE International Conference on*, March 2012, pp. 153–161.

[5] D. Riboni and C. Bettini, "Differentially-private release of check-in data for venue recommendation," in *Perv. Comp. and Comm. (PerCom), 2014 IEEE Int. Conf. on*, March 2014.

[6] M. Handte, U. Iqbal, W. Apolinarski, S. Wagner, and P. J. Marrón, "The NARF architecture for generic personal context recognition," in *Sensor Net., Ubiquitous, and Trustworthy Comp. (SUTC), 2010 IEEE Int. Conf. on*, June 2010.

[7] M. Ali, "Gathering and matching of user information derived from social networks," Bachelor's thesis, Universität Duisburg-Essen, March 2011.

[8] Y. Chon and H. Cha, "Lifemap: A smartphone-based context provider for location-based services," *Pervasive Computing, IEEE*, vol. 10, no. 2, pp. 58–67, April 2011.

[9] W. Apolinarski, U. Iqbal, and J. Parreira, "The GAMBAS middleware and SDK for smart city applications," in *Pervasive Computing and Communications Workshops (PERCOM Workshops), 2014 IEEE Int. Conf. on*, March 2014.

[10] H. Chen, T. Finin, and A. Joshi, "An ontology for context-aware pervasive computing environments," *Knowl. Eng. Rev.*, vol. 18, no. 3, pp. 197–207, Sep. 2003.

[11] D. Hardt, "The OAuth 2.0 authorization framework, draft-ietf-oauth-v2-31," July 2012.

[12] W. Apolinarski, M. Handte, U. Iqbal, and P. J. Marrón, "Secure interaction with piggybacked key-exchange," *Pervasive and Mobile Computing SI*, vol. 10, Part A, 2014.

[13] L. Kagal, T. Finin, and A. Joshi, "A policy language for a pervasive computing environment," in *Proc. of the 4th IEEE Int. Workshop on Policies for Distributed Systems and Networks*, ser. POLICY '03. IEEE Computer Society, 2003.

[14] D. Hong, M. Yuan, and V. Y. Shen, "Dynamic privacy management: A plug-in service for the middleware in pervasive computing," in *Proc. of the 7th Int. Conf. on Human Computer Interaction with Mobile Devices & Services*, ser. MobileHCI '05. ACM, 2005, pp. 1–8.

[15] W3C, P3P Working Group, "The Platform for Privacy Preferences 1.1 (P3P1.1) Specification." [Online]. Available: http://www.w3.org/TR/P3P11/

[16] E. Toch, N. M. Sadeh, and J. Hong, "Generating default privacy policies for online social networks," in *CHI '10 Ext. Abs. on Human Factors in Comp. Sys.*, ser. CHI EA '10, 2010.

[17] L. Fang and K. LeFevre, "Privacy wizards for social networking sites," in *Proc. of the 19th Int. Conf. on World Wide Web*, ser. WWW '10. ACM, 2010, pp. 351–360.

[18] G. Danezis, "Inferring privacy policies for social networking services," in *Proc. of the 2nd ACM Workshop on Security and Artificial Intelligence*, ser. AISec '09. ACM, 2009.

[19] N. Vyas, A. Squicciarini, C.-C. Chang, and D. Yao, "Towards automatic privacy management in web 2.0 with semantic analysis on annotations," in *Collaborative Computing: Networking, Applications and Worksharing, 2009. CollaborateCom 2009. 5th International Conference on*, Nov 2009, pp. 1–10.

[20] E. Toch, "Crowdsourcing privacy preferences in context-aware applications," *Personal and Ubiquitous Computing*, vol. 18, no. 1, pp. 129–141, 2014.