# Protecting User Profile Data in WebID-Based Social Networks Through Fine-Grained Filtering

Stefan Wild, Olexiy Chudnovskyy, Sebastian Heil, and Martin Gaedke

Technische Universität Chemnitz, Germany
{firstname.lastname}@informatik.tu-chemnitz.de

**Abstract.** The WebID identification approach allows users to manage their profile data at a self-defined place in the cloud and enables services as well as other requesters to retrieve data stored within these profiles. While existing access control mechanisms can secure entire user profiles from unauthorized access, they lack fine-grained protection of sensitive data within user profiles.

This paper presents an approach for applying requester-specific filters to cloud-stored user profile data in WebID-based distributed social networks. Our approach aims at enabling profile owners to protect sensitive user data within their profiles in a fine-grained manner. We demonstrate our solution by integrating the approach into a WebID identity provider and profile management platform.

**Keywords:** Security, Privacy, Trust, Identity, Social Web, Semantic Web.

## 1 Introduction

With increasing presence of social media in daily activities [1], the need for trustworthy collaboration is becoming more and more important [4]. Centralized social networks such as Facebook, Google+ or LinkedIn provide varied possibilities for personal information exchange and networking, but try to bind users within their own domains [15]. Although a growing number of social networks tends to make parts of their collected data available to the public through APIs [9], users are not in full control of their identity data. Avoiding the creation of data silos on the one hand and enabling users to remain in control of their data on the other hand asks for a distributed social network (DSN) [15].

A DSN can be implemented on the basis of W3C's WebID specification [13]. The WebID approach is a universal identification mechanism that enables persons and machines to identify themselves via client certificates, i.e., without entering any user name or password [11]. The client certificate refers via a URI to a resource containing further data about the identity owner. This specific URI is called WebID and the linked resource is called WebID profile. Users can automatically generate a WebID, an appropriate WebID profile, and a client certificate using a WebID identity provider. Data within a WebID profile describes attributes of the identity owner in a machine-readable way via RDF[1] using

[1] RDF Primer, http://www.w3.org/TR/2004/REC-rdf-primer-20040210/

Q.Z. Sheng and J. Kjeldskov (Eds.): ICWE 2013 Workshops, LNCS 8295, pp. 269–280, 2013.

domain specific vocabularies like FOAF[2]. While utilizing profile data for creating an improved user experience and optimizing customer services is advantageous, there is also a major problem related to this topic:

An unprotected WebID profile is a potential source of information for known and unknown as well as wanted and unwanted requesters. Since WebID profile documents are parsed during authentication to verify public key information, they have to be accessible for other services or agents. That is, also profile data irrelevant to the authentication procedure per se could be retrieved without further notice. For protecting WebID profiles from unauthorized access, data retrievals or tracking attempts, one could set access control rights to resources representing the WebID profile documents [5,2]. Existing mechanisms only provide coarse access control because they focus on resources instead of represented data. Enabling fine-grained access control with these mechanisms requires outsourcing sensitive WebID profile data to separate resources and set corresponding access permissions. This kind of profile data distribution, however, negatively affects flexibility, ease of maintenance, and portability of user profiles [7].

The paper presents our approach to protect WebID profile data from unauthorized access by providing these main contributions:

1. Theoretical foundation for fine-grained filtering of WebID profile data
2. Practical implementation of our approach using SPARQL
3. Demonstration as part of a WebID identity provider & management platform

The rest of this paper is organized as follows: We describe usage scenarios and derive challenges for a solution in Section 2. We then analyze related work in Section 3. Section 4 presents our approach. We evaluate the solution in Section 5 and show an example in Section 6. We conclude the paper in Section 7.

## 2 Usage Scenarios

The following scenarios illustrate various aspects of the problem and are used to derive the challenges that need to be dealt with:

*Scenario 1.* As a WebID identity owner, Alice intends to restrict her profile data's visibility. She wants to do this because all information available could be easily retrieved, if not properly addressed by appropriate access control mechanisms. Sensitive profile data could be used for purposes she does not agree with, e.g., social network analysis or product marketing. Although restricting access to her entire profile would be sufficient, Alice is not interested in losing advantages like authentication or single-sign-on to new yet unknown services. To keep associated services up-to-date, Alice wants to permit monitoring specific profile parts by third-party entities for changes. Alice wants to allow anyone to access profile data she marked as visible, even if Alice is currently unavailable or unauthenticated.

[2] FOAF Vocabulary Specification 0.98, `http://xmlns.com/foaf/spec/`

*Scenario 2.* A friend of Alice, called Bob, wants to retrieve her current address data. The identity owner Alice knows Bob and has granted him more visibility rights compared to anonymous in Scenario 1. While Bob is allowed to see Alice's private address data, Alice does not want to share this data with Eve, which might be a loose contact. Instead of private address data, only Alice's office address data is visible to Eve. Alice has to be enabled to express whom exactly she wants to make information available to. That is, any agent authenticated via WebID is to be treated differently when accessing data of Alice's profile.

*Scenario 3.* Alice plans to switch the server hosting her WebID profile. She has distributed her profile data to separate resources for applying access rights at the resource level. For migrating to a new hosting server, Alice has to find, consolidate and transfer all profile data being scattered among various resources as well as adjust access control lists (ACLs) for these resources. Depending on Alice's setup used for securing her personal data, this migration might be a complex undertaking.

Based on these scenarios, we infer following challenges a solution must deal with:

**Flexibility.** Defining filters on profile data for specific requesters must be flexible and expressive to cover all described scenarios.

**Portability.** Profile owners must be enabled to easily transfer filter specifications to other systems without making major adjustments. Filter processors have to be either available or easy to implement within new ecosystems.

**Maintainability.** Filters on profile data have to be standard-compliant to ease maintenance and avoid introducing too much overhead. Both users and machines must be enabled to understand and modify filters on profile data.

## 3 Related Work

Web Access Control (WAC) is a vocabulary to define access rights to resources at the document level [8]. Requesting agents and agent classes are supported as entities to define access rights to. ACLs specified by WAC are machine-readable through RDF and can be stored independently from the resources they protect. WAC is well-suited for scenarios involving many resources to grant access to [3]. However, WAC does not support directly controlling access to specific data within resources, e.g., data within WebID profiles. Outsourcing specific data as self-contained resources enables more control with WAC. This, however, complicates maintenance because the number of resources required to realize a less coarse-grained control depends on amount and structure of data to apply access rights to. Considering a WebID profile containing many triples to describe diverse attributes of a person, a fine-grained control at its best would result in outsourcing almost each triple to a separate resource. When applying changes, this approach is inflexible. Additionally, such data distribution and related definition of corresponding ACLs comes along with declining portability.

The Access Control Ontology (ACO) is similar to WAC, but adds support for roles and enables directly mapping permissions to HTTP verbs [12]. To protect data within resources with ACO, relevant data has to be outsourced to separate resources. ACO and WAC share the same maintainability and portability issues.

The approach proposed in [14] manipulates WebID profile data for particular profile requesters by introducing sets of triples as alternative information sources in relation to the original profile data. This allows to establish diverse views on profiles depending on specific requesters identified through WebIDs. While WAC and ACO only enable controlling access to resources, this vocabulary facilitates manipulating data represented by resources. These view definitions increase flexibility by providing improved filter expressiveness, e.g., new triples can be directly inserted into the profile view. Due to this facility, it is necessary to resolve alternative triples available in both the view definition and the actual WebID profile. While this resolution can be realized by merging and replacing relevant triples, there is further processing required to prioritize what triples are shown or hidden as part of the view. If view definitions are used as an additional layer of information, profile data would be available in two different places, which are probably conflicting with each other. Consequently, data stored in the user profile as well as in the view definitions must be managed. This decreases maintainability.

## 4 Protecting User Profile Data by Fine-grained Filtering

For improving the protection of user profiles in WebID-based DSNs, our approach defines a fine-grained filtering of data marked as sensitive by the profile owner. The filtering is applied during a graph-to-graph transformation. As transformation source, graph $G(V, E), G \in \mathfrak{G}$ represents a WebID profile containing data about identity owner $m \in I$, where $I$ is a set of all identities. $T$ is a set of RDF triples each consisting of subject $s$, predicate $p$, and object $o$. As $T$ spans a graph and $G$ defines a set of triples, we formalize this equivalence in (1).

$$T \sim G \Leftrightarrow \forall (s, p, o) \in T : s, p, o \in V \wedge (s, p) \in E \wedge (p, o) \in E \tag{1}$$

The identity-based graph-to-graph transformation $t$ is defined by (2).

$$t : \mathfrak{G} \times I \rightarrow \mathfrak{G} \tag{2}$$

Transformation $t$ maps graph $G$ to graph $G'$. Graph $G'$ represents identity owner's $m$ WebID profile filtered by sensitive data requester $r \in I$ is *not* allowed to retrieve. Equation (3) formalizes this transformation.

$$t(G, r) = G' = (V' \subseteq V, E' \subseteq E) \tag{3}$$

Our approach handles sensitive data as a subset of triples $T \sim G$. While all sensitive data is available in graph $G$, only data requester $r$ is allowed to see is present in graph $G'$. Filter function $f$ defines a mapping of a set of triples on $\{0, 1\}$ depending on the identity. While "1" means sensitive data and, therefore, set of

triples is present in graph $G'$, "0" means the opposite. Consequently, whitelisting or blacklisting of sensitive WebID profile data for particular requesters can be achieved using filter function $f$ as defined by (4).

$$f : I \times \{(s, p, o)\} \rightarrow \{0, 1\} \tag{4}$$

Function $f$ yields 1 for each triple in graph $G$ and identity owner $m$. Transformation $t(G, r)$ uses $f$ to create filtered graph $G'$ based on $G$ for requester $r$. RDF triples $T' \subseteq T$ span graph $G' = (V', E'), T' \sim G'$ as defined in Equation (5).

$$T' = \{(s, p, o) | f_r((s, p, o)) = 1\} \tag{5}$$

To relieve identity/profile owner $m$ from the need to define filter function $f_r$ for each potential $r$, we introduce fallback function $F(r)$ that yields the best possible fallback entity for a given requester $r$. Possible fallback entities are:

- *requesters* authenticated using WebID $U \subseteq I$,
- *specific requesters* defined by the profile owner $S \subseteq U$,
- requesters who are *friends* of the profile owner $K \subseteq U$, and
- *anonymous requesters* $A \subseteq I, A \cap U = \emptyset$.

Let $R = \{k, u, a, n\}$ be a set of special entities: $k$ for friend, $u$ for authenticated user, $a$ for anonym, $n$ for null. Equation (6) formalizes fallback function $F(r)$.

$$F(r) = e = \begin{cases} r & \text{if } \exists f_r \\ k & \text{if } \exists f_k \wedge r \in K \wedge r \notin S \\ u & \text{if } \exists f_u \wedge r \in U \wedge r \notin S \wedge r \notin K \\ a & \text{if } \exists f_a \wedge r \in A \\ n & \text{if } \nexists f_r \wedge \nexists f_k \wedge \nexists f_u \wedge \nexists f_a \end{cases} \qquad e \in (R \cup S) \tag{6}$$

Filter function $f_n$, cf. Equation (7), implements a behavior as if no filtering is active. This enables accessing profiles having no predefined filters.

$$f_n((s, p, o)) = 1 \forall (s, p, o) \in T \tag{7}$$

To use $F(r)$ as part of $t(G, r)$, we refine Equation (5) as shown in Equation (8).

$$T' = \{(s, p, o) | f_{F(r)}((s, p, o)) = 1\} \tag{8}$$

Filter specifications are hidden from anyone but profile owner $m$. Otherwise, this information is a potential subject to social engineering, e.g., profile analyzers could conclude group affiliations utilizing knowledge about $f_r$ or $F(r)$.

Figure 1 illustrates the theoretical foundation of our solution. When requester $r$ tries to retrieve data from the WebID profile of identity owner $m$, an appropriate filter is searched for requester $r$ using $F(r)$. Profile owner $m$ has to specify eligible filters prior to this step in order to achieve a protection of sensitive profile data. Filters are stored as filter specifications in the identity owner's WebID profile. Each filter specification $(e, t)$ is a set consisting of entity $e$ and

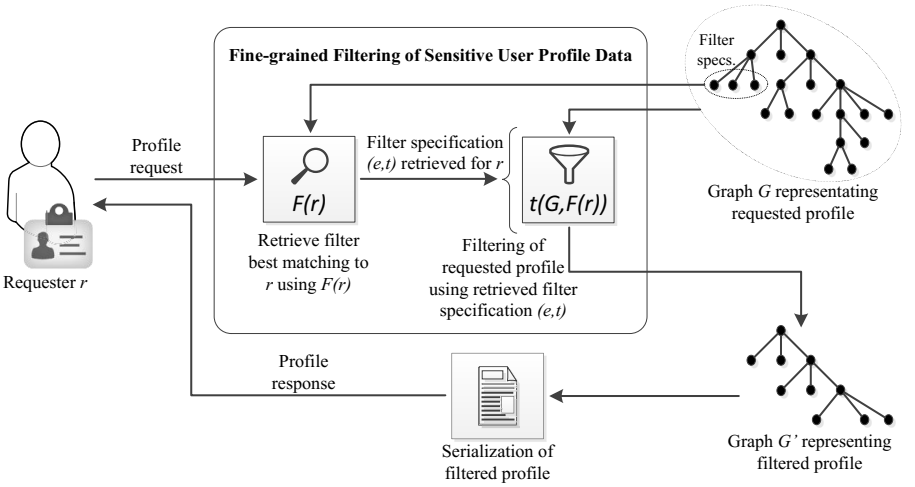


**Fig. 1.** Approach to Protect Sensitive User Profile Data by Fine-grained Filtering

transformation function $t$. As soon as a matching filter specification is detected, graph $G$, representing all WebID profile data of identity owner $m$, is converted via graph-to-graph transformation $t(G, F(r))$ into graph $G'$, representing this WebID profile filtered by data marked as sensitive by profile owner $m$. That is, the profile retrieved by requester $r$ only contains data satisfying the constraints defined by filter function $f_{F(r)}$.

While various technologies are suitable for implementing this approach, we introduce the *WebID Profile Filter Language* (WPFL) and use the SPARQL CONSTRUCT query form [6] as transformation and filter function. WPFL defines the proposed filter specification $(e, t)$. It consists of three elements: *entity name* for $e$, *filter command* for $t$ and a *specification element* to bind them together and connect the filter to the WebID profile. The specification element allows storing filter specifications either in the owner's profile, i.e., graph $G$, or separately as linked resources. These elements are described by three RDF triples, as exemplarily shown in Turtle[3] syntax below:

```
1 <WebID> filter:specification [
2    filter:entity ENTITY;
3    filter:command COMMAND
4 ] .
```

The SPARQL CONSTRUCT query form facilitates constructing a new graph $G'$ based on an existing graph $G$, as required by Equation (3). It can include or exclude data during construction of $G' \sim T'$ and, hence, implements

[3] Turtle Terse RDF Triple Language, `http://www.w3.org/TeamSubmission/turtle/`

Equation (5). A whitelisting - as defined by this equation - mentioning all data to be available in graph $G'$ is described by following filter command[4]:

```
1 CONSTRUCT { ?s ?p ?o } FROM <WebID> WHERE { ?s ?p ?o .
2   FILTER(?s in (Subject1, Subject2, [...])) .
3   FILTER(?p in (Predicate1, Predicate2, [...])) .
4   FILTER(?o in (Object1, Object2, [...]))
5 }
```

As an example, if solely the `foaf:knows` predicate is mentioned, all contacts are copied from $G$ to $G'$. To increase filtering granularity, it is beneficial to also mention subjects or objects of RDF triples, e.g., in order to include/exclude specific contacts. This all together defines one *filter directive*. SPARQL's `UNION` keywords enable to use several filter directives in one filter command. We utilize SPARQL Property Path [10] to cover filtering of context-dependent data. For instance, street data could be context-dependent as they are element of an address, which in turn could be element of either private or business contact data. Property paths help to address relevant elements in graph $G$ by specifying the routes between them. For example, a filter command to construct a new graph by including name and image of identity owner $m$ as well as city and country of his/her home - but not street, postal code etc. - is described as follows[5]:

```
1 CONSTRUCT { ?s ?p ?o } FROM <WebID> WHERE {
2   {?s ?p ?o . FILTER(?p in (foaf:name, foaf:img))} UNION
3   {?s ?p ?o . ?t con:home ?o} UNION
4   {?s ?p ?o . ?t con:home/con:address ?o} UNION
5   {?s ?p ?o . ?t con:home/con:address/con:city ?o} UNION
6   {?s ?p ?o . ?t con:home/con:address/con:country ?o}
7 }
```

A dedicated SPARQL query uses the identity information provided by requester $r$ to select the best-matching available filter specification based on the retrieved filter entity, as formalized in Equation (6). Once an appropriate filter specification is selected, the corresponding filter command is directly passed to a SPARQL processor that executes the graph-to-graph transformation.

## 5 Evaluation

The proposed approach enables profile owners to create filters on their WebID profile for specific requesters or groups of requesters. Profile owners can independently include or exclude each RDF triple present in their profile. This fine-grained and context-aware filtering allows to create customized profile views for

[4] In contrast to whitelisting, blacklisting data is also supported by SPARQL CONSTRUCT queries via `MINUS` statements.

[5] In the example, lines 3 and 4 create the context required for including city and country. Address data is described using the PIM ontology, `http://www.w3.org/2000/10/swap/pim/contact#`

diverse requesting entities. Unlike WAC and ACO, the proposed solution does not require outsourcing data to separate resources for implementing a flexible filtering. All necessary information can remain in one place. Compared to [14], user profile data and filter specifications are separated from each other in our solution. This simplifies updating, replacing or removing already existing filter specifications. The fallback mechanism $F(r)$ selects the most appropriate filter based on availability and provided identity data. Such a fallback mechanism is not part of any related work known to the authors.

In contrast to related work, both whitelisting and blacklisting of RDF triples are supported by our solution. We recommend whitelisting because it does not show what is hidden from requesters. Otherwise, knowledge about hidden data could be subject to speculations. Additionally, whitelisting eases constructing an empty graph representation of a profile, which might be relevant for identity owners having stringent requirements for privacy and, thus, want to forbid anonymous profile requests. Our solution also allows to create filters that remove all filter specifications during construction of the profile view. Consequently, profile requesters remain unaware of the filtering.

To apply filtering of sensitive profile data by utilizing $t(G, r)$, only the graph representing the WebID profile and the requesting entity are used as input parameters. In comparison to ACO, our approach introduces only minimal overhead with three RDF triples to define a filter specification for a specific requester. The owner's WebID profile can contain all filter specifications, i.e., while separation between profile data and filter specifications is allowed, it is not required. To further ensure maintainability, we did not develop an own language for filter commands, but use SPARQL as a well-established and proven language. SPARQL empowers to create flexible and complex filters, whereas related work tries reducing complexity by defining a restricted vocabulary. We assume that restricted vocabularies offer advantages in terms of usability, but they also limit possibilities of filtering and cause workarounds, like the necessity of outsourcing sensitive user profile data. Independent of the chosen solution, we expect that common profile owners do not have the skills to create and maintain filters without assistance through specialized user interfaces.

For seamlessly integrating our solution into existing systems, filter function $f_n$ implements a behavior as if no filtering is active. This facilitates accessing profiles having no predefined filters. As identity owners are enabled to store all necessary filter details within their WebID profile using our solution, the effort to transfer filter specifications to a new hosting system is reduced. While ACO and the approach proposed in [14] rely on particular processors to execute filters, high availability of SPARQL processors for many platforms and architectures contributes to our solution's interoperability and, thus, filter portability.

## 6 Example

While our solution is generic and can be implemented in any platform, we demonstrate it using Sociddea. Sociddea is a WebID identity provider and management

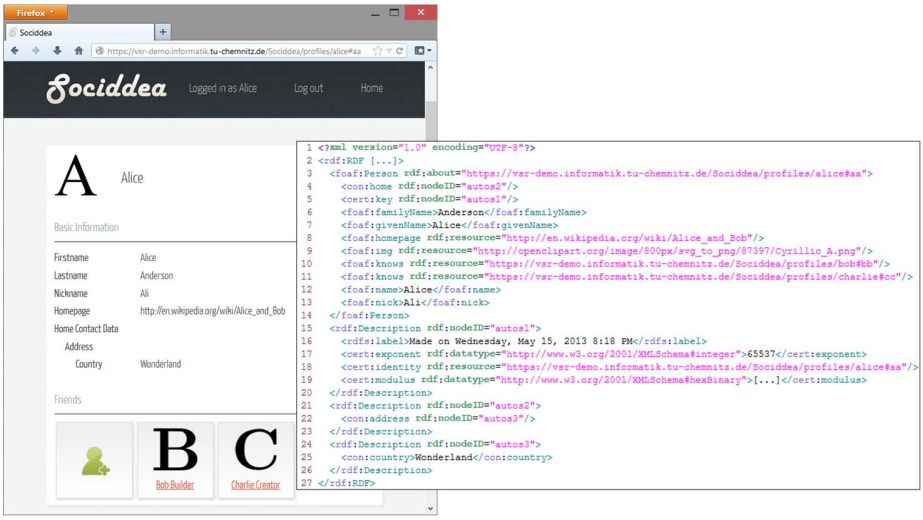


**Fig. 2.** Representations of a WebID profile hosted on Sociddea

platform developed with ASP.NET MVC4. With Sociddea, a user can automatically create a new WebID, an underlying WebID profile and an associated client certificate. Although Sociddea allows users to host their WebID profiles in the ecosystem provided by Sociddea, there is no constraint to do this. That is, users are also enabled to create new client certificates for profiles hosted somewhere else. Sociddea can represent a WebID profile in various ways. Figure 2 exemplifies an HTML and RDF/XML representation for the same WebID profile hosted on Sociddea.

Sociddea provides a graphical user interface to configure filters for profile data. Profile owners can switch from the common profile authoring to the filter specification mode. All identity attributes presented in the profile authoring mode can be used for specifying filters, i.e., each available identity attribute can be marked as either visible or hidden. At the moment of writing this paper, Sociddea's graphical filter editor supports including/excluding identity attributes through predicates, e.g., first name, last name or phone number. Through selecting an available entity, an already existing filter specification is used to visualize the former identity attribute selection by the user. Once the profile owner completed the selection for a specific entity, this configuration is verified and send to the Sociddea back-end. In order to enable machines to process this yet informal filter configuration, a SPARQL CONSTRUCT statement is automatically created. As whitelisting of attributes has been implemented, this SPARQL statement contains all identity attributes declared as visible for the specific entity. All three RDF triples relevant to specify the filter are stored within the owner's WebID profile. The process of creating such a profile data filter is shown in Figure 3.

Although the implementation generates SPARQL CONSTRUCT statements based on the identity attributes selected by the users, the solution is not limited

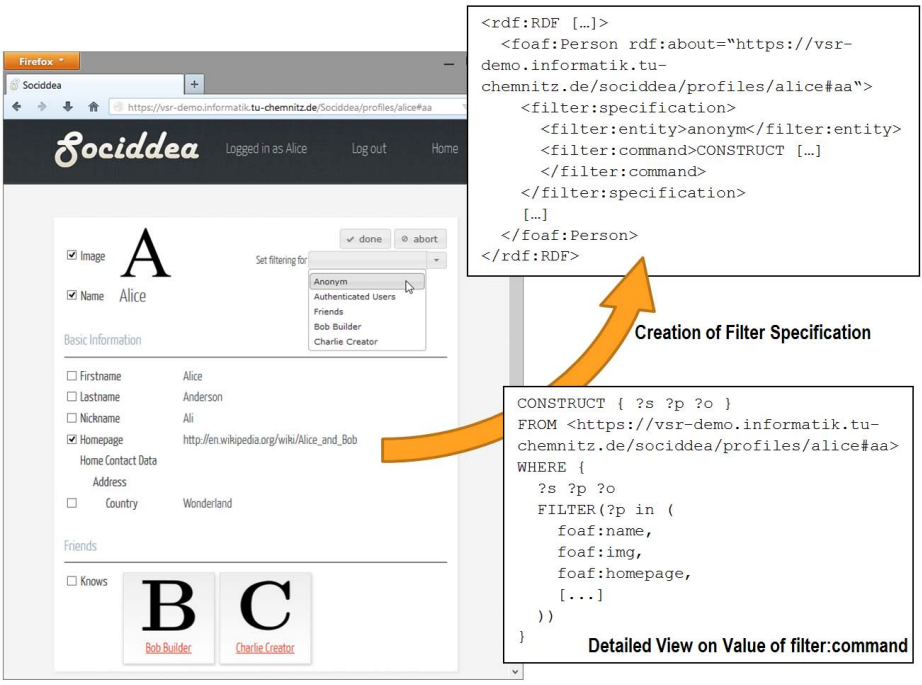


**Fig. 3.** Creation of Filter Specification Based on User Selection

to this. For generating profiles filtered by certain attributes, a profile owner is allowed to use any valid statement. Our solution's flexibility also allows to filter even identity attributes unsupported by the graphical user interface and facilitates to handle special cases like conditional filtering. Both can be accomplished via appropriate SPARQL commands. Once the filter specification has been created, it is automatically considered during all future attempts to access the particular profile. That is, when a requester tries to retrieve the profile, the solution searches for an appropriate filter specification using the provided identity data and the `filter:entity` triples within the WebID profile. Having found a matching filter entity, the `filter:command` triple belonging to the same `filter:specification` is extracted and directly passed to a SPARQL processor, i.e., no modification is made to the command. While results produced by the SPARQL processor are rendered as defined in the request, rendering as such is not subject to our solution. Figure 4 exemplifies the filtering of a WebID profile for an anonymous requester using the previously created filter specification.

Having no additional logic to be interpreted for filtering, the SPARQL processor can directly apply the filter specification and create a new filtered graph. This allows an efficient execution.

**Demonstration.** Further information to our solution and a link to the Sociddea WebID identity provider and profile management platform is available at `http://vsr.informatik.tu-chemnitz.de/demo/sociddea/`

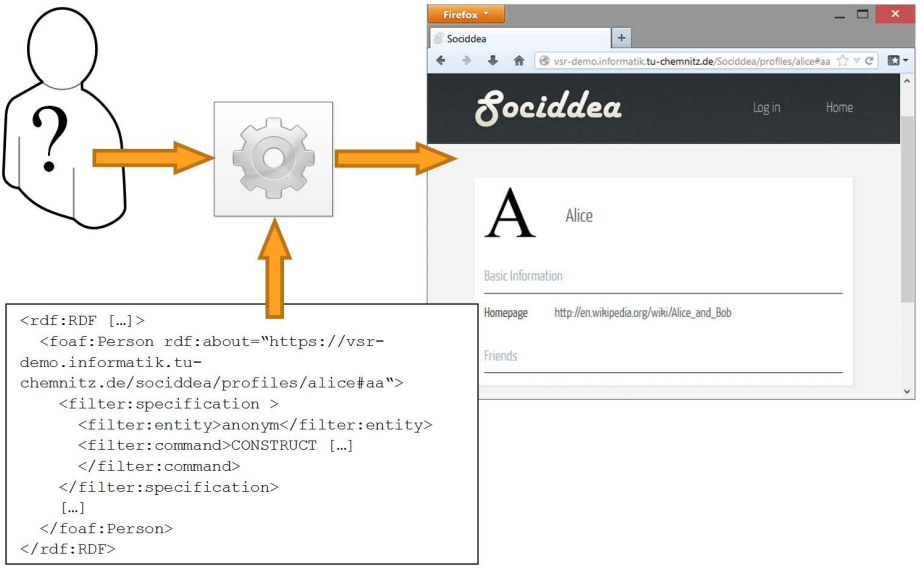


**Fig. 4.** WebID Profile Data Filtered for Anonymous Requester

## 7 Conclusion

In this paper we proposed an approach for enabling identity owners to control the way their profile data is exposed to others. We presented typical usage scenarios demonstrating the need for a flexible, portable and maintainable solution. Our solution is substantiated by both a theoretical foundation for fine-grained filtering and a practical implementation using SPARQL. We demonstrated the solution as part of Sociddea - a WebID identity management platform.

Introducing requester-specific filters on WebID profile data allows profile owners to keep control about amount and nature of personal data being presented to entities requesting their profile data. We defined a filter vocabulary to connect the current profile with the filter specification and established a fallback mechanism to automatically select the best-matching filter depending on the requester. To cover almost all scenarios of hiding and showing specifics within profiles, we used SPARQL CONSTRUCT statements as filter commands. We recommend whitelisting non-sensitive profile data per requester and exclude all filter specifications during filtering.

As future work, we plan to conduct a more extensive evaluation of the proposed solution including a user study focusing on creation and modification of filters. From the technical point of view, we will analyze filter cascades to apply several filters and combine protection needs. We also plan to add facilities for reusing filters by sharing them between users of a DSN. Finally, we intend to extend the requester parameter within filter specifications towards a customizable and machine-readable definition of the requesting party.

**Acknowledgment.** This work was funded by the European Commission (project OMELETTE, contract 257635).

## References

1. Social Media Report 2012 (2012), http://blog.nielsen.com/nielsenwire/social/2012/
2. Bonneau, J., Anderson, J., Anderson, R., Stajano, F.: Eight friends are enough: social graph approximation via public listings. In: Proceedings of the Second ACM EuroSys Workshop on Social Network Systems, pp. 13–18 (2009)
3. Chudnovskyy, O., Wild, S., Gebhardt, H., Gaedke, M.: Data Portability Using WebComposition/Data Grid Service. International Journal on Advances in Internet Technology 4(3 & 4), 123–132 (2012)
4. European Commission: ICT - Work Programme 2013 (2012)
5. Hackett, M., Hawkey, K.: Security, Privacy and Usability Requirements for Federated Identity (2012)
6. Harris, S., Seaborne, A.: SPARQL 1.1 Query Language (2012), http://www.w3.org/TR/sparql11-query/
7. Heitmann, B., Hayes, C.: Achieving privacy-enabled user profile portability with WebID and the Web of Data (2011)
8. Hollenbach, J., Presbrey, J., Berners-Lee, T.: Using RDF Metadata to Enable Access Control on the Social Semantic Web. In: Proceedings of the Workshop on Collaborative Construction, Management and Linking of Structured Knowledge (CK 2009), vol. 514 (2009)
9. Savitz, E.: Welcome To The API Economy - Forbes (2012)
10. Seaborne, A.: SPARQL 1.1 Property Paths (2010), http://www.w3.org/TR/sparql11-property-paths/
11. Sporny, M., Inkster, T., Story, H., Harbulot, B., Bachmann-Gmür, R.: WebID 1.0: Web Identification and Discovery (2011), http://www.w3.org/2005/Incubator/webid/spec/
12. Tomaszuk, D., Gaedke, M., Gebhardt, H.: WebID+ACO: A distributed identification mechanism for social web (2011)
13. Tramp, S., Frischmuth, P., Ermilov, T., Shekarpour, S.: An Architecture of a Distributed Semantic Social Network. Semantic Web (2012)
14. Tramp, S., Story, H., Sambra, A., Frischmuth, P., Martin, M., Auer, S.: Extending the WebID Protocol with Access Delegation. In: Proceedings of the Third International Workshop on Consuming Linked Data, COLD 2012 (2012)
15. Yeung, C.M.A., Liccardi, I., Lu, K., Seneviratne, O., Berners-Lee, T.: Decentralization: The future of online social networking. In: W3C Workshop on the Future of Social Networking Position, Papers 2 (2009)