SURPRISE: User-controlled Granular Privacy and Security for Personal Data in SMARTERCONTEXT

Juan C. Muñoz[1], Gabriel Tamura[1], Norha M. Villegas[1,2,3], and Hausi A. Müller[2,3]

[1]Icesi University, Cali, Valle del Cauca, Colombia
[2]University of Victoria, Victoria, British Columbia, Canada
[3]IBM Canada CAS Research, Markham, Ontario, Canada

Abstract

The Smart Internet relies on the exploitation of information obtained from interactions of users with web applications. A critical aspect for its success is the adoption of mechanisms that guarantee the protection of information sensitive to users. This paper presents SURPRISE, our solution to empower users with privacy and data security control for the access to their information, stored in Personal Context Sphere repositories. These repositories are defined and maintained by SMARTERCONTEXT, our Smart Internet infrastructure that improves the quality of user experience in their interactions with web applications. SURPRISE (i) allows users to configure access permissions to their sensitive personal information to third parties, selectively and with different levels of granularity; (ii) supports changes in these configurations at runtime to add or remove third parties or permissions, and (iii) realizes partial encryption to share non-sensitive data with not explicitly authorized third parties, while protecting user identity. We analyze details of our proof-of-concept implementation with respect to the SMARTERCONTEXT's privacy and security requirements.

1 Introduction

SMARTERCONTEXT is our dynamic context management framework that identifies meaningful context information from the interactions that users perform with web resources throughout their web experiences. SMARTERCONTEXT, driven by the user, stores this information in Resource Description Framework (RDF) repositories that we named *personal context spheres (PCSs)*. Most importantly, the access to the information stored in a PCS is granted by its owner, the user [14]. The case study described in this paper focuses on guaranteeing privacy and security of personal context information in on-line shopping scenarios supported by SMARTERCONTEXT.

Context management must guarantee privacy and security for personal context information. The level of protection depends on the context types from which the information is derived. From the perspective of privacy and security, we classify personal context information as *sensitive* and *non-sensitive* context. Sensitive context corresponds to the context information that must be encrypted along its life cycle. Examples of sensitive data include credit cards, user identities, personal agendas and current and preferred locations. In our e-commerce case study, examples of non-sensitive context data are user preferences about product and service categories. Since users control the access to PCSs, SMARTERCONTEXT must support them in specifying the information they want to share and the corresponding third parties with which this information is exchanged.

SmarterContext relies on RDF [3], RDF-S [12], and OWL-Lite [11] to represent and reason about context information. Thus, we are interested in approaches to privacy and security targeting RDF data. A security approach compliant with SmarterContext must address three requirements: (i) *dynamic selectivity*, (ii) *dynamic granularity*, and (iii) *partial encryption*. Dynamic selectivity empowers the user to decide, at runtime, about third parties authorized to exchange personal context with the SmarterContext engine. Dynamic granularity enables the user to select, at runtime, specific types of personal context data to be shared with a particular third party. Finally, partial encryption concerns the exploitation of non-sensitive context data stored in PCSs. That is, to enable PCS infrastructure providers to develop new business models based on the sharing of non-sensitive data, as long as user identities remain undisclosed.

One of the most representative partial encryption approaches for RDF data is *Partial RDF Encryption (PRE)* [7, 8]. Even though PRE does not support dynamic selectivity and dynamic granularity as defined in SmarterContext, we selected PRE as a baseline for the implementation of our solution for three main reasons. First, it is a sound solution for partial encryption. Second, both its specification and implementation are publicly available. Third, it is based on Jena [4], the semantic web framework that we used to implement the SmarterContext engine.

The main contribution of this paper is Surprise (Smartercontext UseR PRIvacy and SEcurity), our user-controlled approach to ensure the privacy and security of the RDF data stored in PCSs. To realize Surprise, we extended PRE in three main ways: (i) modifying its policy language to enable the definition of security policies as required in SmarterContext, (ii) modifying its data structures to improve efficiency, and (iii) extending its API to support the redefined policies and data structures.

The remaining sections in this paper are organized as follows. Section 2 introduces our online shopping case study. Section 3 provides an overview of SmarterContext and introduces its privacy and data security requirements. Section 4 explains PRE, the baseline of our approach. Section 5 highlights the most important features of Surprise, and explains its specification and implementation. Section 6 discusses related work. Finally, Section 7 outlines future work and concludes the paper.

2 Case Study

In a common on-line shopping scenario, the user browses several stores' web sites to shop for different kinds of products and services. Generally, the user shares common concerns with others while interacting with on-line stores in a particular shopping experience (e.g., a vacation trip or an anniversary). Suppose Anne, a frequent on-line buyer, is shopping for an anniversary gift for her husband. Anne browses through different web sites looking for a special gift according to her husband's preferences known to her. Thus, during this shopping experience Anne interacts with different web sites looking for products that could exploit her husband's interests actually stored in his web browsing history.

The interactions of users with web applications are valuable sources of meaningful information to understand their intents and situations. Moreover, the information gathered from the interactions with a particular web site could be exploited to improve the quality of user experience provided not only by the same web site, but also by others.

For conducting this case study we implemented several on-line stores compliant with SmarterContext.[1] These stores, known as *personal web-enabled sites (PWE-sites)*, were instrumented to exchange context information about users with the SmarterContext engine. SmarterContext exchanges personal context with these third parties by taking into account the nature of the PWE-site (e.g., the web sites with which Anne is interacting will receive information about Anne's preferences, only if they are related to the anniversary concern), and the level of privacy and security configured by the user (e.g., if Anne does not allow SmarterContext to share information

[1]http://smartercontext.org/shopsite1
http://smartercontext.org/shopsite2
http://smartercontext.org/ticketsite
http://smartercontext.org/musicsite

about her credit card, SmarterContext will not share this information with PWE-sites, even when they are part of Anne's PCS).

In this case study the categories of personal context information stored in PCSs correspond to calendar events gathered from personal agendas, preferred product and service categories, credit card details, and shipping and billing information. These data contain sensitive personal context information such as the user's identity, current and preferred locations, and her credit cards numbers, expiration dates, and verification numbers. Since the unauthorized disclosure of this sensitive information may have catastrophic consequences for users and businesses, SmarterContext must implement effective data security mechanisms to protect the information stored in PCSs, while still leveraging its value to provide users with more pleasant on-line shopping experiences.

3 SmarterContext Overview

SmarterContext is a dynamic context management infrastructure that monitors the interactions of users with web entities, such as products offered in an on-line catalog, to gather relevant context information [14]. The information gathered by the SmarterContext engine about a particular user is stored in a persistent repository called *personal context sphere* or PCS for short. SmarterContext reasons about this information to provide context-aware applications, such as daily-deal applications, with up-to-date information useful to understand users' situations and preferences.

The main components of SmarterContext are (i) its ontology, (ii) the service-oriented software infrastructure, and (iii) the users' PCSs. The SmarterContext ontology, which includes several vocabularies, supports context representation and reasoning [13]. The service-oriented infrastructure provides the software components required to manage the context information life-cycle: context gathering, processing, provisioning, and disposal [14]. PCSs are repositories that store the personal context data of SmarterContext users.

3.1 Personal Context Spheres

A *personal context sphere* (PCS) is a distributed repository provided by a third party that stores the context information of a user gathered by the SmarterContext engine [14]. These repositories use RDF [9] represent and manage personal context data. Most importantly, the access to RDF data is controlled by the user.

Context facts are represented as sets of RDF triples. Figure 1 depicts the graph representation of a simple RDF triple with corresponding subject, predicate and object. This triple provides context information about Anne's preferred location. Subject, predicate and object are identified by a URI. For convenience, RDF specifications use a shorthand for referring to URI references (QName). In this way, the full URI is defined by appending the local identifier to the abbreviation (QName prefix). Figure 2 depicts a partial view of Anne's PCS. In particular, it presents context facts related to her preferred payment method.

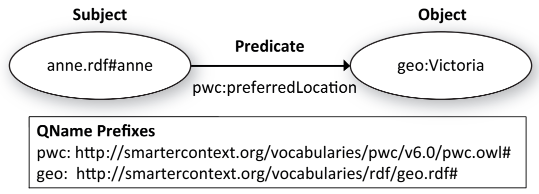

Subject
Object
Predicate
anne.rdf#anne
geo:Victoria
pwc:preferredLocation
QName Prefixes
pwc: http://smartercontext.org/vocabularies/pwc/v6.0/pwc.owl#
geo: http://smartercontext.org/vocabularies/rdf/geo.rdf#

Figure 1: An RDF triple representing context

SmarterContext realizes the vision of exploiting information about users, gathered from their past and present interactions with web applications, to deliver more pleasant and effective shopping experiences. For this, SmarterContext keeps track of simple interactions such as "likes", "wishes", "purchases" and "rankings" to gather meaningful information about the user's preferences, and share this information with on-line stores that offer related products or services. An important difference of e-commerce models envisioned in SmarterContext with respect to traditional ones, is that in existing on-line shopping applications the knowledge gained about the buyer's behavior is exploited by individual web sites only, which own and have full control of the users' data.

The integration of new context is performed

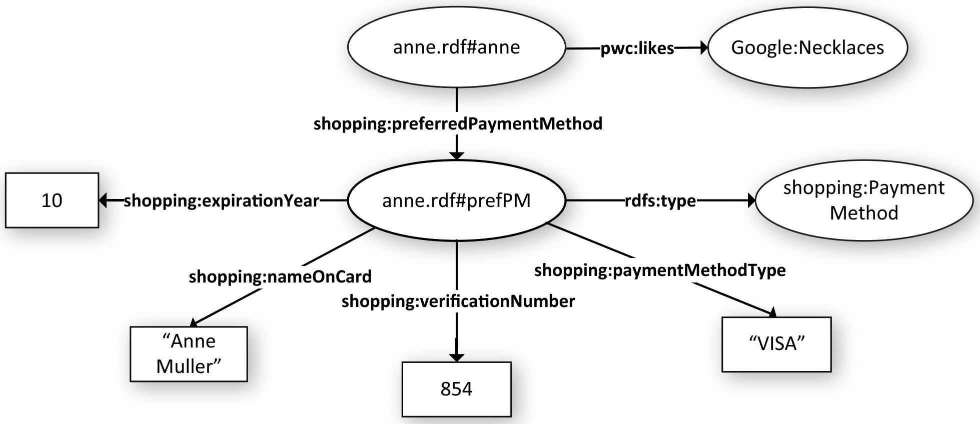


Figure 2: A partial view of Anne's PCS. Oval nodes correspond to RDF resources identified by URIs that represent either context types in the SmarterContext ontology [13], or instances of these types (e.g., anne.rdf#anne represents an instance of type pwc:user). Arcs represent the predicates defined in our ontology as context relationships and context properties. Rectangular nodes correspond to literals (values). Predicate pwc:likes and object Google:Necklaces correspond to non-sensitive context data.

by the user either through the PCS user interface directly, or assisted by SmarterContext when new data is gathered from web entities. PCSs also enable users to integrate web entities exposed through web services such as personal agendas, shopping lists, and social networks. These web entities constitute the consumers and providers of the user's personal context information gathered and provisioned by SmarterContext. Therefore, for a particular person, SmarterContext only shares context information with the applications and services integrated into that user's PCS. For example, when Anne created her PCS, she provided her preferred shipping and billing addresses using the SmarterContext user interface.

Users can configure privacy and security levels selectively. For example, Anne approved the disclosure of her credit card information with one of the PWE-sites in her PCS only. Thus, SmarterContext must guarantee that Anne's credit card information will not be shared with any other PWE-site. Moreover, given the sensitive nature of this information, Anne's credit card data must be protected against unauthorized access by third parties either during their provision to the PWE-site, or while stored in Anne's PCS. Similarly, her husband could enable his PCS's application to provide information for improving Anne shopping for her anniversary gift to him.

SmarterContext has the potential to trigger radical changes in existing e-commerce models. For example, information about users stored in PCSs can be exploited also to improve the accuracy of results in business analytics. Of course, without compromising the privacy and security of sensitive personal data. As long as user privacy is guaranteed, an important part of this information could be shared with third parties to enable new business opportunities based on information services. For example, information about Anne's preferred product categories, such as the fact that she likes necklaces (cf. Fig.2), could be shared with related retailers in an anonymous way (without disclosing her identity).

3.2 Privacy and Data Security Requirements

Important factors of the security quality attribute are privacy, integrity, and availability [1]. These quality factors concern the protection of data against unauthorized disclosure, modification, and destruction, respectively.

Using this characterization, we defined the privacy and data security requirements for

SmarterContext as follows:

RQ-1. *Dynamic Selectivity*: SmarterContext must exchange only selected personal context information with PWE-sites integrated into PCS, taking into account that the list of PWE-sites with which the user shares context information changes over time. To exchange non-sensitive information with third parties not included in the user's PCS, the SmarterContext engine must disassociate the user identity from this information to guarantee its anonymity.

RQ-2. *Dynamic Granularity:* Since the user may decide to share her context information partially, SmarterContext must allow the user to configure different levels of granularity (e.g., in the form of policies) for the access to her personal context information. Moreover, granularity levels may vary over time.

RQ-3. *Partial Encryption:* Sensitive personal information must be encrypted when stored, gathered, and provisioned. However, for SmarterContext to be able to reason about context information stored in PCSs, only sensitive data must be encrypted according to the specified policies.

Concerning requirement RQ-1 (dynamic selectivity), when the user integrates PWE-sites into her PCS, she specifies whether SmarterContext is allowed to either gather and send, only send, or only gather context information from and to the corresponding third party. In this sense, data security in SmarterContext guarantees that sensitive personal data (i.e., any datum associated with the identity of the user) is available only to PWE-sites or other PCSs that have been allowed by the user. Regarding requirement RQ-2 (dynamic granularity), in SmarterContext the user may decide to share some types of information only. Moreover, these types can vary from one third party to another. Finally, concerning requirement RQ-3 (partial encryption), since the SmarterContext infrastructure (including PCS repositories) is maintained by cloud providers (also third parties), data protection must be guaranteed even in cases where this information is accessed by not explicitly authorized parties. In such cases, the information will remain inaccessible because it is encrypted. However, if PCSs are encrypted fully, SmarterContext will not be able to reason about context information thus compromising the usefulness and value of these data. Operations performed by the SmarterContext engine such as union of RDF graphs, inferences, and filtering cannot be executed on encrypted triples.

4 Partial RDF Encryption

Some approaches, based on partial RDF encryption, have been proposed to address the aforementioned problems on RDF data security. One of the most representative is the proposed by Giereth [7, 8]. This approach presents a method called Partial RDF Encryption (PRE), which offers a Java API (PRE4J), to generate encrypted containers for sensitive data. This method uses synchronous and asynchronous keys to encrypt single subjects, predicates, objects, triples and triple sets with different encryption keys, algorithms and security features defined in policy files.

The two main components of PRE are (i) *encryption containers (EC)*, and (ii) *encryption (PREPolicy) policies*, whose structure and functions are summarized as follows.

4.1 Encrypted Containers

An *encrypted container (EC)* is a data structure used to store the encrypted RDF elements together with the key ciphers and the encryption metadata [7]. This metadata includes the encryption algorithms to be used with their parameters, hash values and the public key identification information. ECs are included in the original RDF graph as literals in new triples, replacing the original RDF segments that were encrypted. An EC contains an *EncryptedData* slot, which is composed of four sub-slots [7]:

i. `EncryptionMethod`: defines the encryption algorithm;
ii. `EncryptionProperties`: contains information such as the digest value, digest algo-

rithm and data type information;

iii. `KeyInfo`: stores the public encryption keys into `EncryptedKey` slots. The structure of each `EncryptedKey` corresponds to the structure of the `EncryptedData` slot; and

iv. `CipherData`: stores the encrypted RDF elements.

The PRE4J API supports multiple encryption key sets (public keys) for sensitive data using a session key in the encryption process. This session key is encrypted with each public encryption key and stored in the `KeyInfo` slot. In the decryption process, a valid private key is used to decrypt the session key from the respective `EncryptedKey` slot. Then, the session key is used to decrypt the data stored in the `CipherData` slot. Every time the data must be encrypted, an EC is generated for each RDF element. Moreover, every time the data in the EC must be modified, the whole EC is decrypted. That is, a decryption is performed whenever it is required to add, replace or delete keys, or access the encrypted data.

4.2 Encryption Policies

PREPolicy encryption policies are specified in the *PRE Policy Language (PRE-PL)* and define the RDF elements to be encrypted, the encryption properties and the encryption keys used [7].[2]

The XML structure of the PREPolicy policy defines three main sections, as illustrated in Fig. 3:

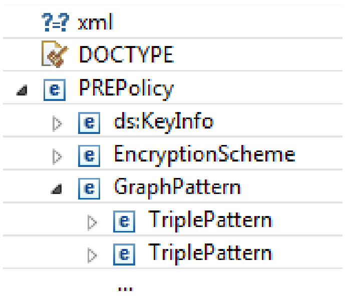
i. `KeyInfo`: defines child elements for the public keys used to store the session's encrypted key. This element is currently not supported by the implementation.

ii. `EncryptionScheme`: (documented as DefaultEncryptionScheme) defines the algorithms and digest methods for encryption.

iii. `GraphPattern`: defines one or more `TriplePattern` children elements to identify the RDF elements to be encrypted.

A `TriplePattern` specifies a pattern in the form of an RDF triple (subject, predicate, object), having the structure specified in Listing 1.

[2]The complete specification of PRE-PL is not publicly available. It was published partially in [7]. Moreover, after analyzing the available implementation carefully, we concluded that it does not support the available part of the specification completely.



```
?=? xml
DOCTYPE
PREPolicy
  ds:KeyInfo
  EncryptionScheme
  GraphPattern
    TriplePattern
    TriplePattern
    ...
```

Figure 3: The main elements of the PREPolicy specification.

Listing 1: `TriplePattern` specification

```
TriplePattern ::= '(' subject-p
            predicate-p object-p ')'
where
  subject-p ::= variable|URI
  predicate-p ::= variable|URI
  object-p ::= variable|URI|literal
```

Informally, the `TriplePattern` structure semantically represents a pattern to select RDF triples to either (i) query, or (ii) update (possibly encrypting) elements in the RDF data graph. In case (i), the `TriplePattern` selects triples whose subject, predicate or object matches `subject-p`, `predicate-p`, and `object-p` correspondingly. As usual, `variables` are bound to values in match operations, and their values are used in other `TriplePatterns` of the same `GraphPattern`. In this case, each of the RDF triples filtered must match all of the `TriplePatterns` consistently, that is, with corresponding matches with the shared variables. In the case of `URIs` and `literals`, the RDF item is bound to the respective value. When several `TriplePatterns` are specified, all of them are used in the same RDQL query to retrieve the triples of interest.

In case (ii), for the update operation, the `Encryption` optional element of the `TriplePattern` must be specified for the filtered triples to be encrypted, including the `Target` sub-element. This sub-element is currently supported with the options `'s'` (to encrypt the subject),`'p'` (predicate),`'o'` (object), or `'t'` (the whole triple). If these child elements are omitted, the `TriplePattern` is useful only to restrict other `TriplePattern`s with shared variables. In other words, the set of selected triples that result from the whole set of `TriplePattern`s are used to bound the free variables and update the RDF data, encrypting only the corresponding triple elements according to the policy specification, as explained above.

5 Our Approach: Surprise

This section presents SURPRISE, our solution to realize user-controlled privacy and data security for RDF data in SMARTERCONTEXT. SURPRISE is based on Partial RDF Encryption (PRE), the baseline approach proposed by Giereth [7, 8] as explained in Sect. 4 above.

5.1 Overview

Even though PRE supports requirement RQ-3 (cf. partial encryption in Sect. 3.2), its PREPolicy encryption policy is insufficient to address requirements RQ-1 (dynamic selectivity) and RQ-2 (dynamic granularity) as defined for SMARTERCONTEXT. This means that with PRE only, we can reason on Anne's non-sensitive context data because it supports partial encryption, but Anne neither can grant different privilege levels to different PWE-sites nor select particular context types to be shared with a particular PWE-site. To address these requirements, we extended PRE in three ways. First, we modified the PREPolicy structure and semantics to enable the definition of policies that support requirements RQ-1 and RQ-2 for SMARTERCONTEXT (hence called PREPolicySC). Second, we modified the PRE encryption containers (ECs) to support requirement RQ-1 more efficiently. Finally, we extended the PRE4J API to provide the functionality required to support the redefined encryption policies and ECs.

Table 1 summarizes the modifications implemented in SURPRISE with respect to four features directly related to the requirements in SMARTERCONTEXT (cf. Column 1 in Table 1). The new capabilities implemented in SURPRISE, the ones independent of PRE, are explained in Sect. 5.4.

5.1.1 Public Keys

To support requirement RQ-1 (dynamic selectivity), SURPRISE manages public keys at the user level. Thus, users can grant access to different sensitive context types of their personal context to particular PWE-sites. For this, our approach supports the user in linking a public key for each third party authorized to receive context information from her PCS. In our case study, this modification enables Anne to approve the disclosure of her credit card information with only one of the PWE-sites in her PCS. Public keys in SURPRISE, managed according to the Internet public key infrastructure [2], are stored in a repository of the SMARTERCONTEXT infrastructure and associated to users' PCSs, whereas in PRE are stored in the PREPolicy encryption policies. Finally, even though PRE-PL allows the specification of many public keys, we found no implementation evidence for this feature in the PRE4J API.

5.1.2 Policies

In contrast to the PREPolicy, PREPolicySC encryption policies support the specification of multiple sensitive context types to be encrypted. This first modification in the management of policies targets efficiency because only one query and iteration are required when encrypting and decrypting more than one sensitive context type. For example, to access the user's location only it is not required to decrypt other sensitive context such as her credit card information. The second change in this category is the association of triple patterns to sensitive context types rather than to public keys. This feature addresses requirements RQ-1 (dynamic selectivity) and RQ-2 (dynamic granularity) because multiple triple patterns for the identification of sensitive context types can be applied to a particular user. Moreover, these patterns can be combined in different ways depending on the information the user wants to share with the corresponding PWE-sites. With this, the user can decide to share her credit card and location information with some of the PWE-sites, but only her location with others.

Table 1: Modified features of PRE in SURPRISE

Feature	PRE	SURPRISE	SURPRISE additions to PRE
Public Keys	Integrated in the PRE-Policy definition (specified in PRE-PL but not implemented in PRE4J).	Defined at the user level. Stored in SMARTERCONTEXT and associated to users' PCSs.	User encryption keys deleted from the PREPolicySC file and stored in user's PCS to allow indirect association between encryption keys and sensitive context types. This is to address RQ-1 (dynamic selectivity).
Policies	One PREPolicy for each sensitive context type. Multiple queries and iterations required when encrypting multiple context types.	One PREPolicySC supports the specification of multiple sensitive context types to be encrypted. Only one query and iteration required to encrypt multiple context types.	Sensitive context types defined as a `KeyRefType` child element in the `TriplePattern` elements of the policy to support the encryption of multiple sensitive context types. This is to address RQ-2 (dynamic granularity).
	Triple patterns associated to public keys.	Triple patterns associated to sensitive context types.	
Encryption Containers	One EC required for each triple. Single EC unsupported for encrypting triple sets obtained from the application of policies.	One EC for all the triples in the same sensitive context type. Single EC supported for encrypting triple sets obtained from the application of triple patterns defined in policies.	Sensitive context triples grouped according to their context type defined in the SMARTERCONTEXT ontology. This is to address efficiency.
Support for Changes at Runtime	Public keys cannot added to or deleted from an existing EC at runtime.	The `Encryptedkey` slot is modified without affecting other slots in the EC, when adding or removing keys at runtime.	The encryption process was simplified to modify only the `KeyInfo` slots to add or remove public keys dynamically. This is to address RQ-1 (dynamic selectivity).
	The addition of new encrypted triples into an existing EC at runtime is unsupported.	New encrypted triples can be added into an existing EC without decrypting the stored data.	The EC meta-data was modified to add the context type of the encrypted data into the clear section of the EC to allow the addition of new encrypted triples without decrypting the existing data. This is to address RQ-2 (dynamic granularity).

5.1.3 Encryption Containers

To improve efficiency, SURPRISE redefined ECs to support the storage of multiple triples belonging to the same sensitive context type. The current semantics of PREPolicy compromises the space and time complexity of queries and updates of RDF triples since it requires one EC to store each encrypted triple. For example, to store the information about Anne's preferred payment method, that is her credit card and all of its attributes, one EC data structure is created for each of these attributes (e.g., one EC for the triple that represents the credit card number, or one for the verification number.).

5.1.4 Support for Changes at Runtime

Support for runtime change is a key feature of SMARTERCONTEXT given the dynamic nature of context information. In PRE, the addition of new public keys and the deletion of existing ones at runtime requires the creation a new EC. In PRE4J, changes at runtime imply the decryption of the whole data in the EC (even when the sensitive context data is not modified), the definition of a new list of public keys, and the creation of the new EC. Under these settings, performance is compromised and users could not grant access to their context information to new PWE-sites, nor deny permissions granted previously. For example, Anne could not integrate new PWE-sites to her shopping experience without compromising the privacy of her sensitive data. Moreover, once she has granted a set of privileges to a particular PWE-site, those privileges would apply forever. To support requirement RQ-1 (dynamic selectivity) efficiently, we modified PRE4J to allow the addition and removal of public keys to the `KeyInfo` slot without affecting other slots in the EC.

Another important improvement concerning changes at runtime is the addition of new encrypted triples without decrypting the stored data. In contrast, PRE does not support requirement RQ-2 (dynamic granularity) efficiently because existing ECs cannot be used for the addition of new triples. For example, if Anne just granted access to her location information to a PWE-site, she could not give further privileges to this web site in the future.

5.2 PRE Policies Redefined

Given the analysis presented in the previous section, to satisfy the privacy and security requirements established for SMARTERCONTEXT the first step in our strategy was to redefine the PRE policies.

As illustrated in Fig. 4, our encryption policy, PREPolicySC, is a modification of the PREPolicy policy (cf. Sect. 4.2). Structurally, we use the same policy elements of PREPolicy except for the `KeyInfo`, which is omitted to satisfy requirements RQ-1 (dynamic selectivity) and RQ-2 (dynamic granularity). In contrast, our definition adds a `KeyRefType` child element to the `Encryption` element of `TriplePatterns` to satisfy requirement RQ-3 (partial encryption).

Nonetheless, despite this structural similarity, the most important aspect of our policy redefinition is its semantics, that is, what the policy represents in terms of RDF encrypted data when applied to PCSs.

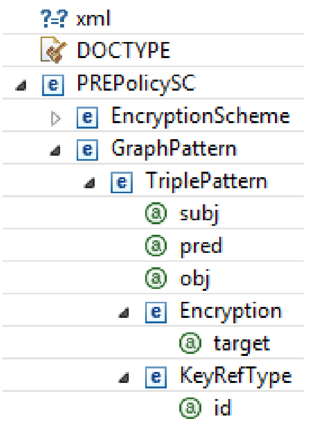
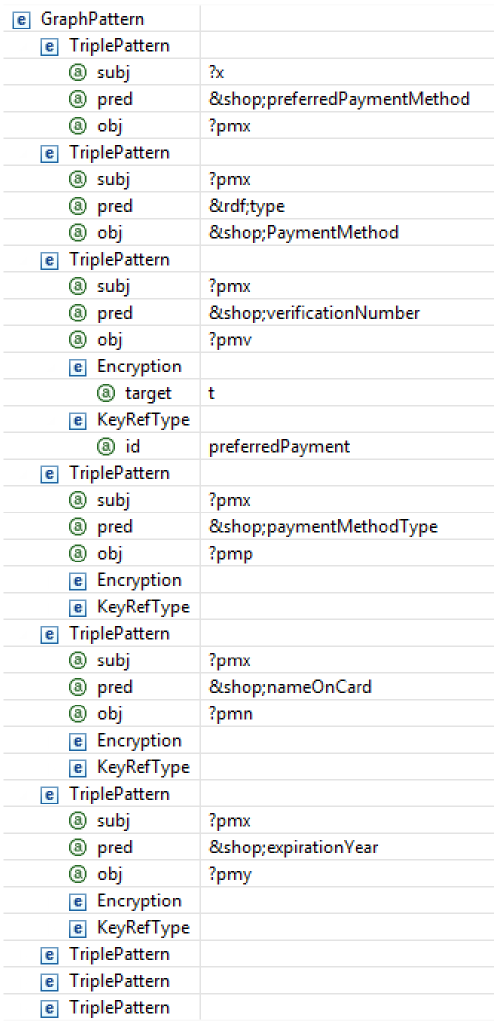


Figure 4: The main elements of PREPolicySC specification

To illustrate this aspect better, consider the application of encryption policies in our case study. Recall from Fig. 2 Anne's preferred payment method, represented in her PCS as an RDF sub-graph composed of six triples. Anne's PCS, depicted partially in Fig. 2, has an extra triple that represents non-sensitive data, in particular the fact that she likes necklaces. In addition, assume the policy `GraphPattern` specification illustrated in Fig. 5. This pattern can be used to encrypt preferred payment methods in SMARTERCONTEXT, such as Anne's method. It is worth noting that we can still omit the `KeyRefType` elements to be able to include the `GraphPattern` as a valid one in a PREPolicy encryption policy. Thus, informally, the semantics of such a PREPolicy applied to Anne's PCS yields the RDF graph presented in Fig. 6. Notice that the triple related to her product preferences remains unencrypted.

That is, the triples that match the triple patterns corresponding to `expirationYear`, `nameOnCard`, `verificationNumber` and `paymentMethodType` are bound to 10, "Anne Muller", 854, and "VISA", respectively. These triples are then encoded as triples $(\mathtt{BN}_i,$ `renc:encTriples`, $\mathtt{EC}_i)_{1\leq i\leq 4}$, respectively, where:

- $\mathtt{BN}_i$ is a blank node;
- `renc:encTriples` is the predicate that identifies encrypted triples; and



GraphPattern	
TriplePattern	
subj	?x
pred	&shop;preferredPaymentMethod
obj	?pmx
TriplePattern	
subj	?pmx
pred	&rdf;type
obj	&shop;PaymentMethod
TriplePattern	
subj	?pmx
pred	&shop;verificationNumber
obj	?pmv
Encryption	
target	t
KeyRefType	
id	preferredPayment
TriplePattern	
subj	?pmx
pred	&shop;paymentMethodType
obj	?pmp
Encryption	
KeyRefType	
TriplePattern	
subj	?pmx
pred	&shop;nameOnCard
obj	?pmn
Encryption	
KeyRefType	
TriplePattern	
subj	?pmx
pred	&shop;expirationYear
obj	?pmy
Encryption	
KeyRefType	
TriplePattern	
TriplePattern	
TriplePattern	

Figure 5: Example of a PREPolicySC `GraphPattern` for encrypting `preferredPaymentMethod` triples.

- $\texttt{EC}_i$ is the encryption container that encapsulates the encrypted RDF elements (e.g., the encryption of `(anne.rdf#prefPM, shopping:expirationYear, 10)` for the first triple), and the corresponding key ciphers and encryption metadata.

If we define a PREPolicySC policy with the `GraphPattern` specified previously, and apply it to encrypt the preferred payment method in Anne's PCS, the semantics yields the RDF graph presented in Fig. 7.

In this case, the four triples that match the specified triple patterns are encoded as the

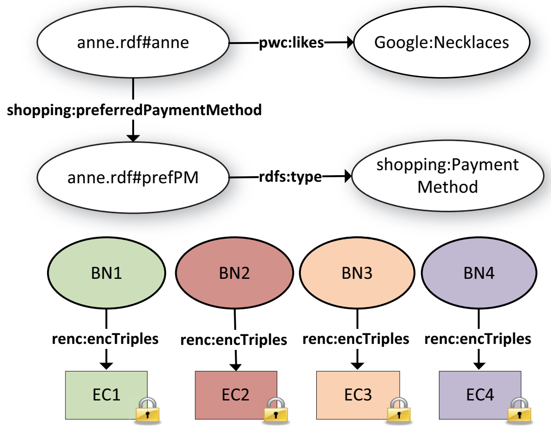


Figure 6: Semantic interpretation of the PREPolicy encryption policy of Fig. 5 applied to Anne's PCS.

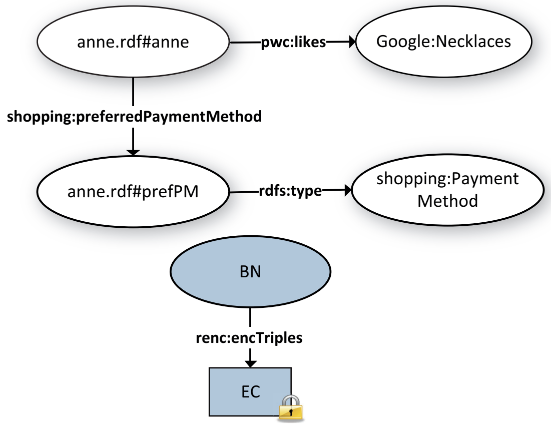


Figure 7: Semantic interpretation of the PREPolicySC policy of Fig. 5 applied to Anne's PCS.

triple `(BN, renc:encTriples, EC)`, where `BN` is a blank node; `renc:encTriples` is the predicate that identifies encrypted triples; and `EC` is the encryption container that encapsulates the four RDF elements encrypted, with the corresponding key ciphers and encryption metadata.

It is worth noting that the presented structural modifications to the encryption policy and its associated semantics constitute a sound basis to realize the features presented in Sections 5.1.3 and 5.1.4, specially allowing to address the efficiency concerns. In the following sections we present the details of how our proof-of-concept implementation realizes these modifications.

5.3 Modified Features

This sub-section presents details about the features of PRE that we modified in SURPRISE. To support policies in context management, we defined PREPolicySC files based on the PREPolicy files defined in PRE.

5.3.1 Public Keys

Deletion of user encryption keys from the PREPolicySC file. We redefined PRE-PL to delete public keys from the policy file in SURPRISE (cf. Fig. 4) and include them in the user's PCS. With this, SURPRISE supports queries of encrypted triple sets without referencing the user's encryption keys. Having no association between public keys and the PREPolicySC file enables the definition of different sets of keys for different types of context data, at several granularity levels that can be modified at runtime. This feature addresses requirement RQ-2 (dynamic granularity). For instance, to support our user in giving access to her location information to a PWE-site that had access to her credit card information only.

5.3.2 Policies

Specification of multiple sensitive context types on a single PREPolicySC file. We defined the sensitive context type as a `KeyRefType` child element in the `TriplePattern` elements of the policy (cf. Fig. 5). With this, SURPRISE matches triple patterns with key sets using the sensitive context types defined in the SMARTERCONTEXT ontology. This level of indirection based on the types of context information to be encrypted allows SURPRISE to support the definition of one policy file for multiple context types. This feature addresses requirement RQ-2 (dynamic granularity).

5.3.3 Encryption Containers

Storage of multiple triples of the same context type in one EC. Aiming at improving performance, SURPRISE minimizes the number of ECs required to encrypt the sensitive context data of each user, thus avoiding unnecessary key generation processes and optimizing spacial complexity. For this, SURPRISE groups sensitive context triples according to their context type defined in the SMARTERCONTEXT ontology. For example, to encrypt all the triples related to the user's credit card in one EC only. This optimization mechanism applies for the encryption of triples only, that is, it is unsupported for the encryption of RDF subjects, objects and predicates independently. This feature addresses efficiency.

5.3.4 Support for Changes at Runtime

Modification of encrypted public key sets. The integration of new PWE-sites into the user's PCS implies the addition of new keys into the corresponding ECs. Similarly, the deletion of PWE-sites implies the deletion of the corresponding keys from their ECs. To avoid the full decryption and encryption of ECs when modifying key sets, SURPRISE implements a simplified process where only the `KeyInfo` slots are modified. This feature addresses requirement RQ-1 (dynamic selectivity).

Suppose the user wants to integrate a new PWE-site with which she wants to exchange context information. For each context type to be shared with this third party, the key addition process has four steps: (i) identification of the EC that corresponds to the context type she wants to share; (ii) testing for decryption of the encrypted keys in the `KeyInfo` slot with private keys provided by users or PWE-sites, to obtain the appropriate session key; (iii) generation of a new encrypted key for the session key obtained in (ii), using the new public key; (iv) generation of the hash code and storage of the new encrypted key together with the hash value in the `KeyInfo` slot.

The key deletion process has three steps: (i) identification of the EC; (ii) identification of the `EncryptedKey` in the `KeyInfo` slot using its hash value; (iii) deletion of the identified encrypted key from the `KeyInfo` slot.

Modification of ECs to support the addition of triple sets dynamically. SURPRISE supports the addition of new triples into existing ECs without decrypting and encrypting all the information stored in the container, and without creating new ECs. To implement this feature, we modified the definition of the EC meta-data to add the context type of the encrypted data into the clear section of the EC. With this modification, it is possible to identify the container without accessing the encrypted data.

5.4 New Features

5.4.1 Graph Patterns for SmarterContext sensitive data

For this case study, we identified six sensitive context types from the categories defined in the SMARTERCONTEXT ontology. These types are related to the user's locations, payment methods, personal identification and other personal data such age, gender and marital status [13].

Triple patterns allow the identification of context data to be encrypted in users' PCSs. In Sect. 5.2 we explained the redefinition of the policy semantics for PREPolicySC, which includes the identification of sensitive context types from the triple patterns defined in the `GraphPattern` specification (cf. Fig. 5).

Consider the partial view of Anne's PCS presented in Fig. 2 and the `GraphPattern` specification in Fig. 5. This RDF graph presents a partial description of Anne's preferred payment method (node `anne.rdf#prefPM`). The `GraphPattern` specification defines one `TriplePattern` element for each of the triples with subject `anne.rdf#prefPM` in Anne's graph. The last three triple patterns, neither detailed in the `GraphPattern` of Fig. 5 nor in the graph of Fig. 2 for simplicity, correspond to the expiration month, the card number and the billing address associated with the preferred payment method.

According to this `GraphPattern` specification, the first triple pattern allows the identification of all the triples that match `(?x, shopping:preferredPaymentMethod, ?pmx)` from Anne's graph. In this case, the triple matching this pattern is `(anne.rdf#anne, shopping:preferredPaymentMethod, anne.rdf#prefPM)`. The remaining triple patterns with subject `?pmx`, excepting the second `TriplePattern` element that is intended for triples with `rdfs:type` as the predicate, indicate that all the triples in the graph with subject `anne.rdf#prefPM` (but with predicate different than `rdfs:type`) must be encrypted. Thus, the two first triple patterns are intended only for the discovery of the data sensitive context information in the PCS, and their corresponding triples in the graph remain clear to identify the existence of a preferred payment method but without any further details.

These triple patterns define also the `Encrytion` and `KeyRefType` elements that define the triple marker for encryption and the corresponding sensitive context type. In the example of the preferred payment method, these elements are bounded to `'t'` (triple) and `preferredPayment`, respectively. The triple patterns for the remaining sensitive context types are defined in the same way with the appropriate values for subjects, predicates, objects and markers for encryption.

5.4.2 Storage of Encryption Keys in the PCS

In SMARTERCONTEXT is the user who controls the sharing of the context data stored in her PCS. This implies the granting of context exchange privileges to PWE-sites and the encryption policies for sensitive context types. For this, we implemented the mechanisms to store in the SMARTERCONTEXT infrastructure the public keys of all the PWE-sites registered by the user. Then, these public keys are linked in the user's PCS to the sensitive context types that the corresponding PWE-site can access. Public keys are also added to ECs as explained in Sect. 5.3.4. Whenever the user decides to stop sharing a particular sensitive context type with a PWE-site, the corresponding key references are deleted from the PCS, and the encryption key from its `KeyInfo` slot as detailed in Sect. 5.3.4.

The semantics of privileges granted to PWE-sites in user PCSs is supported by the SMARTERCONTEXT ontology and its new SURPRISE module as follows. For each PWE-site integrated by the user, SMARTERCONTEXT creates a new triple in the PCSs with subject the user, predicate `pwc:hasIntegrated`, and object the URI that identifies the PWE-site. Then, each PWE-site is described by a set of triples using the following data property predicates: `sprise:receiveSCData`, `sprise:sentSCData`, `sprise:hasPublicKey`, and `sprise:hasAccess`. The domain of all these properties correspond to the `pwc:PWESite` type. `sprise:receiveSCData` and `sprise:sentSCData` have range boolean and define whether PWE-sites can send and receive, only send, or only receive context data to and from the user's PCS. `sprise:hasPublicKey` defines a string value with the identifier of the PWE-site's public key stored in the SmarterContext infrastructure's key repository. Finally, `sprise:hasAccess`, which range includes any context type defined in the SmarterContext ontology, defines the set of context categories that the PWE-site can access.

6 Related Work

A partial encryption solution for RDF data in SmarterContext must support the encryption of elements of the RDF graph with different sets of keys to address requirement RQ-2 (dynamic granularity). Moreover, for a particular RDF element or triple set the solution must support multiple encryption keys, to allow multiple third parties to decrypt the data without encrypting the information once for each different key in a PCS.

The *XML Encryption Syntax and Processing* solution, proposed by Esatlake [5], supports partial encryption of XML elements and has several similarities with PRE, our baseline approach. However, its main problem with respect to SmarterContext is that the structure of encrypted XML files is not compliant with RDF, the context representation language used in SmarterContext. That is, valid RDF graphs cannot be recovered from the encrypted XML files. Moreover, given the complexity of RDF graphs in SmarterContext, it would be impractical to transform from encrypted XML files to RDF files, to access encrypted information.

Full storage encryption solutions [10] such as the one used by Dropbox guarantees that the information stored in repositories provided by third parties, Amazon EC3 in this case, is only available for the Dropbox infrastructure. This is user transparent, which means that Dropbox has full access to and control of the user's data. This is completely contrary to the SmarterContext vision where users are empowered to control the access to their personal data. This type of solutions can be used in Surprise in addition to partial encryption to guaranty that all the information stored in a third party is only readable by the SmarterContext infrastructure according to the policies defined by the users.

In *file encryption solutions* the complete decryption of the file is required to access the encrypted information [10]. Even tough this schema allows the management of keys by either the infrastructure providers or the user, the application needs to have full access to the unencrypted file when accessing the user data. Thus, the whole data is exposed even when accessing to selected context types only.

RDFCrypto, proposed by Gerbracht [6], is another approach similar to PRE [7]. It also allows the encryption of elements in an RDF graph. Instead of containers, it defines new RDF statements to specify the information required to encrypt user data. This approach does not consider the use of multiple keys for the same data and provides no mechanism to identify the elements to be encrypted dynamically. Therefore, it does not address any of the three security and data privacy requirements in SmarterContext. Moreover, it has no available implementation.

7 Conclusions

This paper presented Surprise, our policy-based mechanism that provides private and secure access to PCSs maintained by SmarterContext, our dynamic context management framework for improving the quality of the user experience in Web interactions.

Even though SmarterContext enriches the quality of user experience by exploiting previous interactions of the user with Web applications and sites, the cost of realizing this is the exposure of user's sensitive information in subsequent Web interactions. To avoid compromising the user's privacy and security, we identified three requirements in SmarterContext: (i) *dynamic selectivity*, (ii) *dynamic granularity*, and (iii) *partial encryption*.

With respect to these requirements, the contribution of this paper is twofold. First, we extended the Partial RDF Encryption (PRE) approach, based on a set of modifications of the structure and semantics of its encryption policies. Second, we implemented the changes implied by these structural and semantic modifications in the PRE source code, and integrated them into a proof-of-concept version of Surprise. In light of these modifications, we analyzed the way our approach satisfies the SmarterContext's privacy and security requirements by (i) allowing users to control the access to their sensitive information by third parties in encryption policies, selectively and by means of different levels of granularity; (ii) supporting changes in these policies at runtime; and (iii) enabling partial encryption on specific elements of data, also allowing not explicitly authorized third parties to exploit users' non-sensitive data. Thus, Surprise solves the privacy and security requirements of SmarterContext, and allows its users to be confident that their data remain safe while exploiting personal context information to improve the quality of user experience in their Web interactions.

Our plans for future work include to (i) fully integrate Surprise into SmarterContext and evaluate it by analyzing different quality attributes; (ii) improve the granularity of policies to allow the partial sharing of sensitive data belonging to the same sensitive context type—for example, in case Anne wants to share only some of her payment methods; (iii) improve selectivity by allowing users to decide about the sensitivity of context data. Currently, sensitive context types are defined statically in the SmarterContext ontology. Finally, we plan to investigate the application of SmarterContext and Surprise to other problem domains such as health care.

Acknowledgments

This work was funded in part by the National Sciences and Engineering Research Council (NSERC) of Canada under the NSERC Strategic Research Network for Smart Applications on Virtual Infrastructure[3] (SAVI - NETGP 397724-10) and Collaborative Research and Development program (CRDPJ 320529-04 and CRDPJ 356154-07), IBM Corporation, University of Victoria (Canada), and Icesi University (Colombia).

[3]http://www.nsercpartnerships.ca/How-Comment/Networks-Reseaux/SAVI_AIIV-eng.asp

About the Authors

Juan C. Muñoz is a Master student, Department of Information and Communications Technology, at Icesi University, Cali, Colombia. His research focuses on context management and semantic web technologies for the implementation of user-centric smarter commerce platforms. He is a lecturer in the Faculty of Engineering, at Icesi University since 2006, where he received a Diploma Degree in Systems Engineering in 2005.

Gabriel Tamura is a Professor of the Department of Information and Communication Technologies at Icesi University, Cali, Colombia. He is a Visiting Scientist at the INRIA Lille Nord Europe, ADAM Team/Project, France. Dr. Tamura's research interests include component-based software engineering, self-(re)configuring and self-managing systems, and context-aware systems. He received a Diploma Degree in Systems and Computing Engineering in 1992 from Javeriana University, Cali, Colombia, an MSc Degree in 1996 from University of Los Andes, Bogota, Colombia, and a co-supervised PhD Degree in Computer Science in 2012 from University of Los Andes and University of Lille I, Lille, France.

Norha M. Villegas is a PhD candidate under the supervision of Dr. Hausi A. Müller, Department of Computer Science, University of Victoria, Canada. She is a CAS student at the Center for Advanced Studies at the IBM Toronto Laboratory. Her dissertation focuses on the application of dynamic context management techniques for the optimization of self-adaptive software systems. She received a Diploma Degree in Systems Engineering and a Graduate Degree in Organizational Informatics Management in 2002 and 2004, from Icesi University, Cali, Colombia.

Hausi A. Müller is a Professor of the Department of Computer Science and Associate Dean of Research, Faculty of Engineering at University of Victoria, Canada. He is a Visiting Scientist at the Center for Advanced Studies at the IBM Toronto Laboratory (CAS), CA Canada Inc., and the Carnegie Mellon Software Engineering Institute (SEI). Dr. Müller's research interests include software engineering, self-adaptive and self-managing systems, context-aware systems, and service-oriented systems. He serves on the Editorial Board of Software Maintenance and Evolution and Software Process: Improvement and Practice (JSME). He is Chair of the IEEE Technical Council on Software Engineering (TCSE). Dr. Müller received a Diploma Degree in Electrical Engineering in 1979 from the Swiss Federal Institute of Technology (ETH), Zürich, Switzerland and MSc and PhD Degrees in Computer Science in 1984 and 1986 from Rice University in Houston, Texas, USA.

References

[1] M. Barbacci, M. H. Klein, T. A. Longstaff, and C. B. Weinstock. Quality Attributes. Technical report CMU/SEI-95-TR-021, Software Engineering Institute, 1995.

[2] M. Benantar. The Internet Public Key Infrastructure. *IBM Systems Journal*, 40(3):648–665, 2001.

[3] C. Bizer, T. Heath, and T. Berners-Lee. Linked Data — The Story So Far. *International Journal on Semantic Web and Information Systems*, 5(3):1–22, 2009.

[4] J. J. Carroll, I. Dickinson, C. Dollin, A. Seaborne, K. Wilkinson, and D. Reynolds. Jena: Implementing the Semantic Web Recommendations. In *13th International World Wide Web Conference (WWW 2004)*, pages 74–83, 2004.

[5] D. E. Eastlake, J. M. Reagle, T. Imamura, B. Dillaway, and E. Simon. Xml encryption syntax and processing. World Wide Web Consortium, Recommendation REC-xmlenc-core-20021210, December 2002.

[6] S. Gerbracht. Untersuchung von Möglichkeiten zur Datensicherheit in RDF. Student research project, Dresden University of Technology, 2005.

[7] M. Giereth. Partial Encryption of RDF Graphs. In *The Semantic Web ISWC2005*, Heidelberg, 2005. Springer-Verlag.

[8] M. Giereth. PRE4J - A Partial RDF Encryption API for Jena. *Academic Medicine*, 70(3):216–223, 2006.

[9] F. Manola and E. Miller. RDF Primer. World Wide Web Consortium, Recommendation REC-rdf-primer-20040210, February 2004.

[10] K. Scarfone. Guide to Storage Encryption Technologies for End User Devices. Special Publication 800-111, November 2007.

[11] The World Wide Web Consortium (W3C). OWL Web Ontology Language Reference. http://www.w3.org/TR/owl-ref/, 2004.

[12] The World Wide Web Consortium (W3C). RDF Vocabulary Description Language 1.0: RDF Schema. http://www.w3.org/TR/rdf-schema/, 2004.

[13] N. M. Villegas and H. A. Müller. The SmarterContext Ontology and its Application to the Smart Internet: A Smarter Commerce Case Study. In M. Chignell, J. Cordy, J. Ng, and Y. Yesha, editors, *Second Book on the Personal Web*, LNCS. Springer, 2012 (in evaluation).

[14] N. M. Villegas, H. A. Müller, J. C. Muñoz, A. Lau, J. Ng, and C. Brealey. A Dynamic Context Management Infrastructure for Supporting User-driven Web Integration in the Personal Web. In *Proceedings 2011 Conference of the Center for Advanced Studies on Collaborative Research (CASCON 2011)*, pages 200–214, Markham, ON, Canada, 2011. IBM Corp.