Privacy-Preserving Reengineering of Model-View-Controller Application Architectures Using Linked Data

Juan Manuel Dodero*, Mercedes Rodriguez-Garcia, Iván Ruiz-Rube and Manuel Palomo-Duarte

School of Engineering, University of Cadiz, Av. de la Universidad 10, 11519 Puerto Real, Cádiz, Spain
E-mail: juanma.dodero@uca.es, mercedes.rodriguez@uca.es, ivan.ruiz@uca.es, manuel.palomo@uca.es
**Corresponding Author*

Received 18 March 2019; Accepted 22 November 2019;
Publication 03 December 2019

Abstract

When a legacy system's software architecture cannot be redesigned, implementing additional privacy requirements is often complex, unreliable and costly to maintain. This paper presents a privacy-by-design approach to reengineer web applications as linked data-enabled and implement access control and privacy preservation properties. The method is based on the knowledge of the application architecture, which for the Web of data is commonly designed on the basis of a model-view-controller pattern. Whereas wrapping techniques commonly used to link data of web applications duplicate the security source code, the new approach allows for the controlled disclosure of an application's data, while preserving non-functional properties such as privacy preservation. The solution has been implemented and compared with existing linked data frameworks in terms of reliability, maintainability and complexity.

Keywords: Privacy by design, Web of data, Software architecture, Model-View-Controller.

Journal of Web Engineering, Vol. 18_7, 695–728.
doi: 10.13052/jwe1540-9589.1875

1 Introduction

In the realm of a software system, confidentiality and privacy are non-functional properties aimed at protecting the system's information resources. Such properties are especially relevant in the Web of data, largely concerned with procuring web applications that can publicly display their data and information, such that entities from heterogeneous information systems can be connected [20]. Reengineering an existing application for the Web of data must consider how to fulfil privacy preservation properties.

The challenge concerns Privacy by Design (PbD) principles, which consider privacy as an essential property to be considered during the design phase and throughout the entire engineering lifecycle [22]. Engineers have to go beyond functional requirements and be especially responsive to PbD aspects when engineering software artefacts that deal with personal data [4]. Since confidentiality concerns ethical and legal aspects that are beyond the scope of this paper, in the following we focus on the technical aspects related with data privacy preservation of software systems.

Adding non-functional properties, such as privacy preservation, to an already built software system is generally more expensive than taking proper measures while designing its architecture [49]. How expensive it is to take security measures on an existing software system depends on a number of factors, such as: the number, type and scope of security properties; the size and complexity of the information system architecture; and the affordances and constraints of the methods and software technologies used to engineer the reconstruction. Regarding the latter, existing methods to reengineer a legacy application for the Web of data are based on the thorough analysis of the documented specifications, design diagrams and software manuals that describe its architecture [53]. They overlook, however, a first-class element of software architecture and design, which is the source code.

Source code can be used to gain valuable insights about the design and architecture of a legacy web application. The hypothesis of this work is that *reengineering for privacy a web application architecture at the level of source code can provide an advantage in terms of reliability and maintainability* when obtaining an extended version of the application that considers PbD properties, which might have been overlooked in the original version.

Source code-level interventions to improve an application's security and privacy are inspired by and framed in the Security by Design (SbD) and PbD principles, which have significant expressions in Role-Based Access

Control (RBAC) and Privacy-Preserving Data Publishing (PPDP), among other techniques [15].

When it comes to adding PbD protections, it is difficult to imagine a broad-spectrum solution to tame the size, complexity and architectural diversity of web applications. Nonetheless, the large number of web applications based on the Model-View-Controller (MVC) pattern architecture and its derivatives [8] is a Pareto argument in favor of limiting the scope. Therefore, it is reasonable to focus on MVC-based applications in the Web of data as the target application architectures that can be hopefully reengineered.

The realm of such applications has to be focused in software technologies that have been successfully used to publish data and information resources in the Web of data. In this vein, Linked Data (LD) methods have proven as useful to prepare web applications with standard vocabularies and schemata in order to publish and link the application data and resources [23]. Best practices for publishing linked data define how data can be published and linked by means of standard technologies such as RDF and JSON-LD. Providing an existing application with such capabilities encompasses to define a metadata schema, to compile and link the application data, and to provide an Application Programming Interface (API) that enable third parties to browse the application resources based on public metadata [18]. Numerous semantic methods and software frameworks have been used to engineer LD-enabled versions of web applications and information systems [53]. Recently, PbD and privacy preservation are prevalent concerns in the LD research field [27].

Our contribution is a new PbD approach based on LD technologies, used to reengineer MVC-based web applications that improve reliability and maintainability as relevant quality features in the reengineered application, after incorporating the confidentiality and data privacy preservation requirements.

To investigate the former hypothesis, we have followed a design-and-creation information systems research methodology [35]. It involves the steps of awareness, suggestion, development, evaluation and conclusion [52], which constitute the structure of the rest of the paper. After PbD issues are described in Section 2 as part of the awareness step, Section 3 analyzes the existing LD architectures and frameworks in order to suggest the reengineering for privacy PbD strategy. At the end of the suggestion step, we propose a first contribution, consisting in an original classification of current LD reengineering strategies. Section 4 develops, as the main contribution, a general linked data reengineering framework, named EasyData, which is applied to provide legacy MVC-based web applications with privacy preservation properties. Section 5 includes the evaluation of EasyData against other comparable frameworks, along with a discussion of the research results and their limitations. Finally, Section 6 presents some conclusions and future lines of research.

2 Data Privacy by Design

When data about individuals are involved, special care must be taken to avoid privacy violations. Data privacy by design [6] implies that sanitization approaches, based on removing identifiers, are not enough to preserve individuals' privacy, because certain combinations of non-identifying personal data, known as *quasi-identifiers* (QI) [9], may be linked with other information sources to re-identify them [44]. Nowadays, the amount of available information and data sources along with the increasing computational power facilitate to conduct such re-identifications. Because re-identification constitutes a real privacy threat and the protection of individuals' privacy is a fundamental right, legal regulations [14, 51] have set out the need for adequately protecting *personally identifiable information* (PII) [34], which is any information about an individual that can be used to distinguish or trace her identity (e.g., name or birth date) and any other information that is linked or linkable to her identity (e.g., medical, educational, financial and employment data).

To secure PII confidentiality, PPDP techniques are used that generate a transformed version of data that changes the PII it contains, while at the same time offering data that is valid for statistical analysis [21]. In order to address the current obligations for PII protection and, thus, offer ex ante privacy guarantees against identity disclosure, the design of a web application that publishes individuals' data (e.g. a healthcare company web application used by their clients) must consider the diversity of PbD methods and techniques as a first-class requirement. Data privacy preservation can be implemented in legacy web applications when transforming their architecture to a LD-enabled one. This usually involves extending the legacy application with added middleware components [19], which have to duplicate the implementation of diverse non-functional properties like security. A lot of LD techniques and software tools exist to map relational data sources [48], interlinking datasets [54] and exposing LD APIs as middleware [17]. As for the general software systems, these approaches have proven costly and not absent of significant risk [27, 53].

2.1 A Motivating Example

Despite the policies that legally regulate the use of web data sources [6], and in spite of the fact that data items must be anonymized before publishing an application's data, one cannot impede someone from knowing sensitive information [40]. This is especially worrisome in the light of linked data applications.

For instance, let's suppose DS_1 is the dataset of a tax registry web source, having the attributes $address$, $birthdate$, sex, $postcode$, $name$ and $taxes$; and DS_2 is the dataset from another web source to consult energy consumption, containing the attributes $birthdate$, sex, $postcode$, $electricityConsumption$ and $gasConsumption$. Even removing explicit identifiers, an individual's $name$ in DS_1 can be linked with another record in DS_2 through the combination of $postcode$, $birthdate$ and sex attributes. Each attribute value does not uniquely identify a record owner, but linking data from both applications forms a QI that might point to a unique or small number of records. The attacker can thus notice that one house at a certain address might be unoccupied because its $electricity$ and gas consumption are almost nil. This can pose a threat about burglary, but it can be also a tool for tax agencies to investigate occupied rental houses that might have unpaid taxes from the lessor.

Even anonymizing the combination of datasets by means of generalization techniques on the QIs, there is a possibility that QIs are split in two datasets after linking them for a given analysis. For instance, let the DS_1 schema be ($userId$, sex, $postalAddress$, $defaultRisk$), and let DS_2 schema be ($userId$, $occupation$, $defaultRisk$, $electricityConsumption$, $gasConsumption$), as shown in Table 1. Assuming that a data analyst needs to combine DS_1 and DS_2 to predict, let's say, the risk of finance default, then DS_1 and DS_2 can be linked and merged by matching the $userId$ field in a new dataset DS that is then anonymized. Then the sex and occupation attributes form a new QI, which was not included in each dataset separately, so a linking attack is still possible on such fields of DS. After integrating the tables of both datasets, the $(Female, Carpenter)$ individual on the $(sex, occupation)$ attribute pair becomes unique and vulnerable to link sensitive information, such as $postalAddress$ and $energyConsumption$.

Because the ultimate motivation of data releasing is to conduct data-driven analyses, sanitizing and anonymization should be done in a way that the protected data still retain as much analytical utility as possible; that is, the conclusions extracted from the analysis of the anonymized dataset should be

Table 1 Linked data items of an example linking the datasets from a tax registry application and an energy consumption application

$\in DS_1 \cap DS_2$		$\in DS_1 \backslash DS_2$			$\in DS_2 \backslash DS_1$	
	default		postal		electricity	gas
userId	Risk	sex	Address	occupation	Consumption	Consumption
1–3	0y3n	M	A1	Sales	18	17
4–7	0y4n	M	A2	Ceramist	24	8
8–12	2y3n	M	A3	Plumber	25	10
13–16	3y1n	F	A4	Webmaster	20	17
17–22	4y2n	F	A5	Animator	31	11
23–25	3y0n	F	A6	Animator	34	10
26–28	3y0n	M	A7	Carver	32	12
29–31	3y0n	F	A8	Carver	30	14
32–33	2y0n	M	A9	Carpenter	33	11
34	1y0n	F	A10	Carpenter	29	15

similar to those of the original dataset. With the goal of balancing privacy and utility preservation, PPDP methods [15] have been used to sanitize published datasets by modifying the original QI attributes while preserving certain statistical features.

2.2 Reengineering for Privacy Preservation

Different privacy models can be considered to define the sanitizing conditions. One of the most widely used anonymization models is *k*-anonymity [21]. The idea underlying *k*-anonymity [45] is to homogenize the QI attributes to make them indistinguishable in groups of at least k records, thus limiting to $1/k$ the probability of re-identification. Two distortion methods can be used to enforce *k*-anonymity, i.e. generalization and microaggregation. The generalization method [45] homogenizes the quasi-identifiers with the most specific superclass of the *k*-record group, while the microaggregation method [32] homogenizes the quasi-identifiers with the average of the *k*-record group. In previous works, PPDP methods have been improved to exploit the semantics of nominal values and replace them by concepts in an ontology [41,42].

All these are semantic privacy-preserving techniques that, usefully implemented into a linked data application, facilitate the fulfillment of privacy properties. The issue here is how to engineer privacy properties into an existing web application, i.e. reengineering for privacy preservation. For instance, let's consider for the first example of the previous section the development of a privacy-preserving version that follows a layered security architecture [50]. The security controls are usually implemented on top of the data model to provide linked versions of user identifiers and other PII, potentially QIs, such as $postcode$, $birthdate$ and sex. Implementing a new controller operation to link a user ID with its PII, however, might impose a restriction related with data privacy. The logic for the k-anonymity privacy preservation model, for instance, is normally implemented in the controller component. To be fair, it should be duplicated in the database mapping code as well as in the database stored procedures. Code duplication in different architectural layers overly reduces the reusability and maintainability of an application. Implementing the privacy restrictions only at one layer can pose a design-level impediment, since third party applications (e.g. a mobile app) do not necessarily access to the same controller components. Therefore, some browsers might overlook the data privacy-preserving controls. In general, duplicating the controller logic to implement changeable security properties is not a good practice.

3 Reengineering MVC-based Applications

The redesign of a web application to include SbD or PbD properties is more expensive than considering such requirements from scratch, but often it is unavoidable. Regular web applications' architecture can be tackled at any of the three layers of the MVC pattern, namely the data binding *model*, the web *view* and the *controller* logic. Interventions at the view level, known as web *scraping* or *harvesting*, consist in directly accessing the application HTTP interface to extract the data that is published as HTML. It usually requires some type of license agreement with the application owner, but the discussion on this is out of scope of our research. Therefore, we constrain the discussion to the controller and data binding layers of the MVC architecture.

Confidentiality requirements, such as access authorization or privacy preservation, are often implemented as part of the application business logic. Web applications are not usually designed to implement their business logic in the data layer (for instance, as stored procedures of the database), but in intermediate controller components instead. The controller and data binding components of an existing MVC application have been largely explored as alternative points where to provide data access [5, 10, 17]. Security requirements can be implemented in the controller layer, as some frameworks do—e.g. Spring Security[1] maps permissions and access authorizations to each controller function. On the other hand, we can grant RBAC permissions over an application's data at the data binding or the database level. Then,

[1] https://spring.io/projects/spring-security

new controller operations that need to access and render model data will not be protected against unauthorized access. For instance, consider that mobile apps usually include a separate controller layer implementation in the overall architecture. Access control code has to be duplicated in the potentially multiple implementations of controllers, as well as in the data binding layer or even in the database logic. For layered security requirements, RBAC grants and permissioned stored procedures should be implemented also in the database, which might be a source of code duplication. In sum, SbD and PbD are concerns that involve the overall MVC architecture of the application.

3.1 Analysis of Linked Data Architectures and Frameworks

The architecture of linked data applications or Linked Data Application Architecture (LDAA) is a means to structure the components a LD software system comprises [19]. An extension to the LDAA has been implemented upon a linked data API layer [17] on top of a data access layer (see Figure 1) to mediate between consumer applications and the data sources. As described in [19], the most widespread LDAA is the crawling pattern, which is suitable for implementing linked data applications over a growing set of resources that are crawled and discovered. On the top layer, the crawling architecture has a pipeline of modules, i.e. web access, vocabulary mapping, identity resolution and quality evaluation. An RDF API or SPARQL endpoint is served by such modules, which form the data access, integration and storage layer.

Pipelining all the functional modules of the data access and integration layer eventually leads to the integrated database, which feeds the SPARQL endpoint or API mediator module. In the bottom, the publication layer usually implements wrapper modules that, either by web scraping [38] or enriching [26], add semantics to existing resources and data. Setting up a middleware module is a common strategy to reengineer existing sources, which can range from HTML pages to structured data to web APIs [36]. Other approaches harvest semi-structured HTML content and automatically convert it into structured linked data instances [29]. Scraping and data wrapping techniques either convert data to linked data or provide an API to access data [23].

Thanks to an LD framework, reengineering of a legacy application can be implemented at the data binding *model*, the web *view* or the *controller* layers of its MVC architecture. Next, we analyze where in the MVC layers each framework operates.

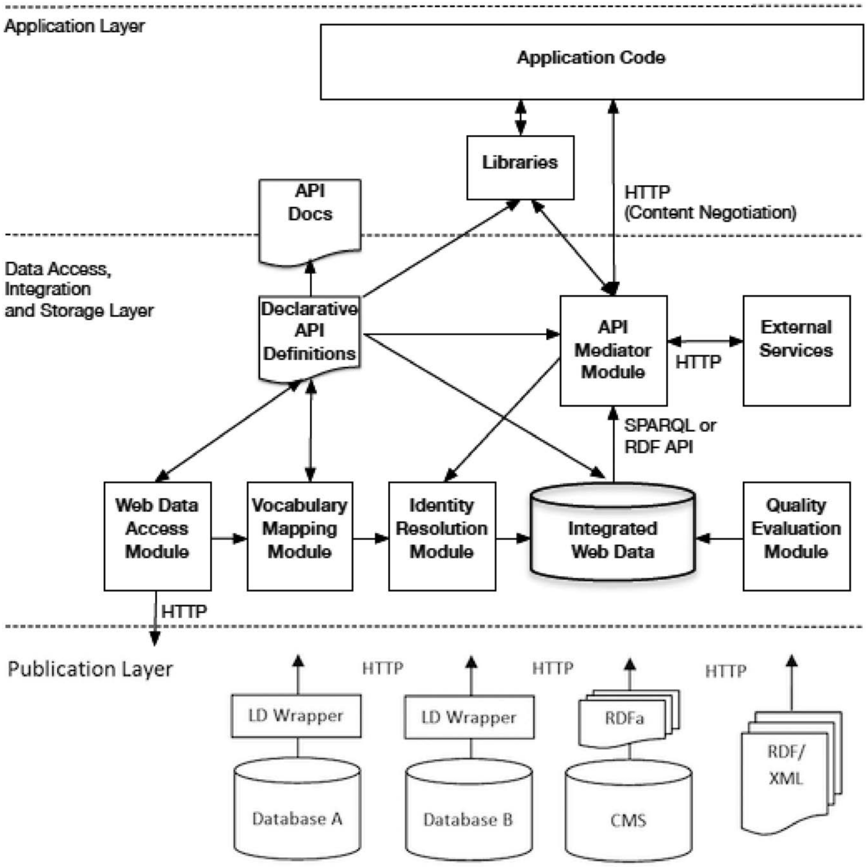

Figure 1 Extension to the original LDAA [19] with additional functions for the data, integration and storage layers [17].

- Apache Stanbol, KIM and SDArch [26] are examples of semantic enriching procedures that operate at the view level.
- The D2Rq server [5], Triplify [3] and Virtuoso RDF Views [12] are useful approaches to build wrappers at the data binding level. ActiveRDF can be used to align an application Object-Relational Mapping (ORM) component with a given RDF schema [37].
- Middleware implementations, such as Virtuoso sponger [13] and Pubby, work at the controller level. Hydra [28] is a middleware implementation that also provides clients with JSON-LD descriptions of a new vocabulary, able to express common concepts of Web APIs. Other solutions,

such as the Datalift platform [46] rely upon existing tools such as Silk [24] to provide interlinked RDF datasets.

3.2 LDAA Reengineering Strategy

Reengineering an MVC-based application at the source code level can be an advantage to provide a reliable and maintainable extension that incorporates additional properties. Clearly, this approach becomes feasible as long as the application source code is available. It has also some constraints and limitations that will be discussed later.

Before articulating the suggestion phase of the research methodology, we have participated in the development of linked open data systems for a number of disciplines, such as Information Science (IS) [25] and Software Process Management (SPM) [43], in which we used the LD tools and frameworks analyzed above. As a consequence, a number of methodological and practical considerations for LDAA reengineering have emerged and influenced the proposed methodology.

4 Proposed Methodology for LDAA Reengineering

We have defined a LDAA reengineering methodology that considers a number of application features, in order to decide the applicable reengineering practices. Such features are: (i) the availability of source code, (ii) the provision of APIs or built-in information exposure services, and (iii) the concealment level for enclosed data. The effort required by the reengineering practices range from a seamless API-based integration of LD-enabled applications, to costly adaptions for those that might not use machine-friendly data formats and protocols. The reengineering methodology is graphically summarized in Figure 2.

4.1 Reengineering Methodology

The methodological aspects have to consider the application architecture. In this vein, the reengineering strategies have been classified as scraping, wrapping, and extension, as depicted to Figure 3. Such strategies can be applied either at the data level or the API level. The following classification is considered as a first contribution of the paper, emerging from a thorough analysis of existing LD frameworks and the prior experience using them to build LD-enabled applications.

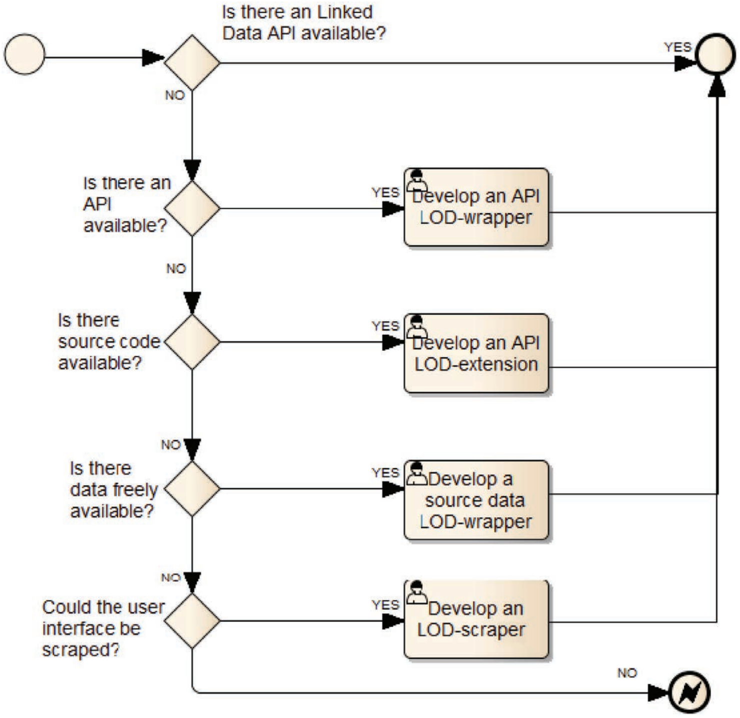
Is there an Linked Data API available?
YES
NO
Is there an API available?
YES
Develop an API LOD-wrapper
NO
Is there source code available?
YES
Develop an API LOD-extension
NO
Is there data freely available?
YES
Develop a source data LOD-wrapper
NO
Could the user interface be scraped?
YES
Develop an LOD-scraper
NO

Figure 2 Methodology to decide on LDAA reengineering.

- *LDAA data scraping* [26,38]: This strategy applies if the web application source code and internal data storage are not available at all, probably because the application was not initially designed for third-party reusing. Information retrieval, web scraping and harvesting techniques are the practicable reengineering alternatives.
- *LDAA data wrapping* [3, 5, 12, 46]: Sometimes the application's source code is not available, but data is available in an open format. Then, adapters or data wrappers can transform LD requests into queries to the application data storage. Depending on the kind of storage, queries can be issued to database systems, structured files or any other data storage system used by the application.
- *LDAA API wrapping* [1, 2, 10]: This is a practical choice when the application already provides an external API for reusing data and information. Then a proxy, wrapper or middleware component is implemented, so that LD requests are formatted for the API and issued

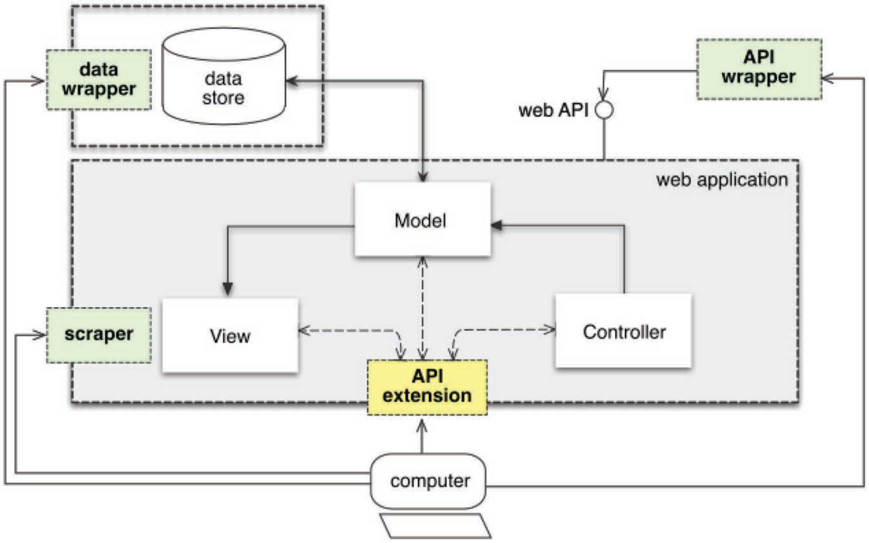

Figure 3 Reengineering strategies for MVC-based web application architectures.

forth and back. The wrapper or middleware can implement some data transformations and adaptations on top of the original API operations.

- *LDAA API extension*: If the application does not provide an external API, but its source code is available, a software add-on can be implemented to provide the LD API. In this case, data and business models can be discovered from source code analysis of the MVC implementation. On one hand, if applied at the *model* layer, the extension strategy generates a LD schema from the internal data model implementation. The schema and data instances can be revealed through an external API. The local namespace for the schema generated in this way can initially reflect the application's internal data model. Yet it can be aligned with standard LD vocabularies through user-defined configurations before publishing, as in the LDAA data wrapping case. On the other hand, if applied at the *controller* level, the API extension strategy can use the existing implementation in order to avoid code duplication. Extending the API does not consist only in wrapping the existing controller implementation (i.e., the internal API) to make it public as a functionally equivalent API, because the external API requirements might not coincide with the internal one's.

The LDAA API extension strategy makes it easier to implement extended features as part of an enriched API. This is an opportunity to include a set of additional, either functional or non-functional features. For instance, the internal API of a legacy application can implement some *finder* methods that return all objects of a given type. The external API, however, may require to define additional *findBy* methods that return only the objects that fulfill a given filtering condition. The latter is simply a functional extension of the existing API. On the other hand, different security privileges can be granted for the *find* and *findBy* methods, or for diverse executions of the same method, depending on the calling user's role. Even the data output from a method call can be sanitized after applying a custom PPDP policy. In the following we will focus on how our LDAA extension strategy is developed to include such privacy preservation properties.

4.2 Privacy-preserving LDAA Extension

Whereas data scraping and wrapping techniques are commonly used to add semantics to existing web applications, we propose a new extension approach that can be used to expose the internal structure and data model of a legacy app as linked data in a controlled and privacy-preserving way.

EasyData is the name of a new LDAA extension approach to reengineer legacy MVC-based web applications so as to provide them with additional non-functional properties. It has been used to implement privacy preservation requirements as a type of security property. The reengineering cycle consists of a number of functional steps, which can be mapped to regular LDAA modules [17] as explained next.

1. *Revealing* the underlying application data model: A linked data model equivalent to the application ORM schema is generated and published as RDF. In addition, upon a web application's request, RDFa and microdata annotations are generated and embedded into the response HTML view. To facilitate external linking with standard vocabularies, metadata mappings can be configured. In this stage, the functionalities of the LDAA *web access* and *vocabulary mapping* modules are developed.
2. *Linking* the application data instances: The linked datasets retrieved from the application's internal data storage can be processed and linked. Internal data items can be directly linked. Afterwards, an external interlinking module can be used to link external resources [54]. A complementary study on how interlinking tools can help data publishers to connect their resources to the Web of data can be found elsewhere [39].

In combination with a proper interlinking tool, this phase develops the functionalities of the LDAA *identity resolution* module.

3. *Controlling* the target non-functional quality properties for the application. Considering the scope of our work, security and privacy preservation of data and information resources are such quality aspects. In this phase, the PPDP techniques described above can be seamlessly applied for each published data item and data type. This phase is part of the LDAA *quality evaluation*.

4.3 Implementing the PbD Interventions

Two different prototypes have been implemented to illustrate and test the EasyData LDAA extension strategy. Each prototype enables the procedure to be applied with two different development languages and open source frameworks, which underpin the architecture of a considerable number of MVC-based web applications. The first is a ruby *gem*[2] used to reengineer ruby-on-rails web applications following the LDAA extension strategy; the second is a python add-on[3] used to deliver LDAA extensions of applications built with the Django framework.

Next, we show how the steps of the EasyData reengineering strategy can be performed using one of the EasyData implementation tools. The Redmine[4] open source project management application is used as a frame example to illustrate how the process can be carried out on a legacy web application.

4.3.1 Revealing the Application Data Model

The first step is to generate and publish an RDF model equivalent to the application's data model. A simplified schema of Redmine data is formed by the *Project*, *Issue*, *User* and *TimeEntry* classes, as illustrated in the Rails implementation of Figure 4. EasyData can render the RDF model from the web application source code, as shown in Figure 4. The `set_rdf_model_name` configuration option defines the alignment of the application data model elements with concepts and properties of a standard RDF schema. In this example, the Redmine `Project` objects are mapped to DOAP *project*s, the Redmine `User` objects are mapped to FOAF *person*s, and the Redmine `TimeEntry` objects are mapped to OWL-Time *duration*s. Redmine `Issue`

[2] https://github.com/dodero/EasyData_Rails

[3] https://github.com/dodero/EasyData_Django

[4] http://www.redmine.org/

```
Namespace.register(
   :doap, "http://usefulinc.com/ns/doap#")
Namespace.register(
   :foaf, "http://xmlns.com/foaf/0.1/")
Namespace.register(
   :time, "http://www.w3.org/2006/time#")
class Project < ActiveRecord::Base
   has_many :issues
   set_rdf_model_name "doap:Project"
end
class Issue < ActiveRecord::Base
  @status = IssueStatus::OPEN
  belongs_to :project
  set_rdf_model_name "xmlns:Issue"
end
class TimeEntry < ActiveRecord::Base
  @spent  = 0
  set_rdf_model_name "time:DurationDescription"
end
class ProjectTimeEntry < TimeEntry
  set_rdf_property_name "xmlns:MemberFor"
end
class IssueTimeEntry < TimeEntry
  set_rdf_property_name "xmlns:AssignedFor"
end
class User < ActiveRecord::Base
  has_many :projects,
           :through => :projectTimeEntries
  has_many :issues,
           :through => :issueTimeEntries
  set_rdf_model_name "foaf:Person"
end
```

Figure 4 Specifying an application's data model with EasyData in the Ruby implementation of Model components.

objects are not mapped to elements from an external vocabulary, since programmers could not find a standard vocabulary defining what a tracking issue is.

4.3.2 Linking Application Data Instances

External linking targets can be added to the application by means of template tags. Instead of linking to the inner application model entities revealed in the previous step, the application view can be provided with links to other entities discovered by an external interlinking tool. For example, an interlinking process configured to match the Redmine data revealed by EasyData with a DBPedia dataset can map Redmine's `Project` with DBPedia's *Project* resource type, `Issue` with DBPedia's *Issue_tracking_system* and `User` with DBPedia's *User_(computing)*. EasyData template tags can be used to include links to such DBPedia entities to configure this mapping.

4.3.3 Controlling Authorized Access

This is the first part of the controlling phase, which is applied twice: one for security access control and another for privacy preservation. Access control grants can be configured for data items, data types and service operations generated in the previous steps. Figure 5 is an example of how the MVC controllers are configured with the `has_permission_on` and `filter_access_to` options. That permits access to `Project` and `Issue` resources as well as the `getIssues` and `getAssignedIssues` operations in a Redmine instance. The example defines access permissions for specific user roles (e.g. `admin` and `analyst`) and operations (e.g., `create`, `read`, `update` and `delete`). Note that this security configuration is a simple extension of the available Rails configuration and does not need to be repeated elsewhere in external LD wrappers, thus reducing the dispensable code smells.

4.3.4 Controlling Data Privacy Preservation

This is the second part of the controlling phase, aimed at privacy-preserving data publishing. The datasets retrieved from the web application database can be configured to be sanitized before release, thereby offering privacy guarantees against identity disclosure. The guarantees are achieved in the example by setting certain privacy requirements to yield *k*-anonymous datasets. Figure 6 shows an example of how the MVC controllers are configured with new sanitizing rules and queries. The `k_anonymity` symbol defined in Ruby specifies the PPDP method that yields *k*-anonymous the data records from a query, with `generalize` and `microaggregate` the available value options. The `k_arg` option specifies the desired value of *k*, which determines the privacy degree of the resulting data records. The higher the *k*, the higher the privacy degree of the result, but the lower its analytic utility will be. Finally, the set of QI attributes to be sanitized is defined with the `quasi_id` option.

The example defines a configuration to sanitize the output data records from the `getAssignedIssues`[5] controller function, which has also an access

[5] The `getAssignedIssues` function actually returns the results of a query that joins a set of attributes from the Projects, Issues and User tables

```
# controllers
class ProjectController < ApplicationController
   filter_access_to :all
   filter_access_to :getIssues,
                    :require => :read
   def getIssues
      @issues = Issue.find(:all)
   end
   def getAssignedIssues
      @issues = Issue.find(:all)
   end
   def getProjects
      @projects = Project.find(:all)
   end
end
class UserController < ApplicationController
   filter_access_to :all
   filter_access_to :getAssignedIssues,
                    :require => :read
   def getAssignedIssues
      @issues = Issue.findByUser(:current_user)
   end
   def getIssues
      @issues = Issue.findByUser(:all)
   end
   def getProjects
     @projects = Project.findByUser(:all)
   end
end

# authorization_rules.rb
authorization do
   role :admin do
      has_permission_on :projects,
        :to => [:create, :read, :update, :delete]
      has_permission_on :issues,
        :to => [:create, :read, :update, :delete]
   end
   role :analyst do
      has_permission_on :projects,
        :to => [:read]
      has_permission_on :issues,
        :to => [:read]
   end
end
```

Figure 5 Integrating security access features with EasyData in the Ruby implementation of Controller components.

```
# controllers
class UserController < ApplicationController
   filter_access_to :all
   filter_access_to :getAssignedIssues,
                    :require => :read
   sanitize_query :k_anonymity => :generalize
                  :k_arg => 4,
                  :quasi_id => [:project_name, :issue_name,
                                :organization, :start_date]
   # ...
end

# k–anonymity_rules.rb
sanitize do
   #only k–anonymity algorithms
   sanitize_rule :microaggregate do
      #implements microaggregation mechanisms
   end
   sanitize_rule :generalize do
      #implements generalization mechanisms
   end
end
```

Figure 6 Integrating privacy preservation features with EasyData in the Ruby implementation of Controller components.

control filter as specified by the `filter_access_to` option. The output dataset will be 4-anonymous via generalization of the QI formed by `project_name`, `issue_name`, `organization` and `start_date`.

5 Evaluation

In this section we evaluate the validity of the EasyData PbD reengineering method to test the hypothesis that it can provide an advantage in terms of reliability and maintainability over other LD solutions. Therefore, we compare EasyData with a baseline of five LD frameworks, which were analyzed in Section 3.1. A thorough inspection was carried out on the software libraries implementation, in order to filter out components that do not provide an equivalent function to our solution's, or have nothing in common between compared frameworks. To ensure a comparable scope and to avoid bias in the filtering criteria, all the library components were carefully analyzed by experts who had previously built IS [25] and SPM applications [43] using these LD frameworks.

In order to understand and analyze the advantages, a benchmark is performed on a number of static analysis metrics of reliability, maintainability and complexity. Measures have been computed with the SonarQube[6] source code static analysis tool. The compared frameworks have been selected as long as they implement LDAA software component modules for either data or API wrapping approaches, they are implemented in a language that can be statically analyzed, and the source code is openly available.

5.1 Measures

The software metrics chosen for static analysis enable to compare software reliability and security, maintainability, and size and complexity, among other features. Except for the size metric, the reliability, maintainability and complexity metrics delivered by SonarQube are language-independent, so the tools can be compared despite their implementation language. The following are the types of metrics provided:

- Size and complexity: measurements of size and complexity of the code.
    - *LOC*: physical Lines of Code; physical LOCs are a simple, source code-dependent measure of the program size.
    - *Statements*: number of statements in the source code; SonarQube unifies this metric and make it independent of the parsed language.
    - *Functions*: number of functions in the source code.
    - *CC*: Cyclomatic Complexity (CC), computed based on the number of control flow paths through the code [33]. SonarQube varies slightly the standard calculation, depending on the implementation language.
    - *CC Density (CCD)*: density measured as the average CC per statement in the source code; it provides a program size-independent measurement of complexity, which is demonstrated to be a useful predictor of software maintenance productivity [16].
- Maintainability and code duplication: amount of code involved in duplications.
    - *Code smells*: the number of code smells, as symptoms in the source code that may indicate a deeper problem.
    - *Technical debt (TD)*: the effort to fix all maintainability issues, measured as *hours* of required work to remediate the issues, or

[6] `https://www.sonarqube.org/`

the *ratio* between the cost to develop the software and the cost to fix it. This ratio is computed as the remediation cost divided by the development cost. The development cost is estimated as 0.06 days (i.e., nearly 30 minutes) per line of code [7].

  - *Lines*: number of duplicated lines in the target language.
  - *Blocks*: number of duplicated blocks of lines in the target language.
  - *Density*: a measurement of the density of code duplication (i.e., number of duplicated lines / overall LOC).

- Reliability and security: measurements of reliability and security of the source code.
  - *Bugs*: the number of bugs, as a measure of software reliability, and a estimated amount of hours for remediation.
  - *Vulner*: the number of known vulnerabilities found, as a measure of software security, and a estimated amount of hours for remediation.

These metrics are not completely independent from each other. For example, size and complexity metrics are a clear indicator of software maintainability [16], while code duplication is a kind of code smell known as *dispensable* code, meaning a portion of unnecessary code that, if properly removed, would make the code cleaner, more efficient or easier to understand. Code dispensability is related with the technical debt. The reason to disclose such metrics separately is to check the reliability and maintainability of the different software solutions due to different causes that might be improved.

The more complex software frameworks are, the more functions they implement. Consequently, the entire source code of software frameworks should not be analyzed. The source code analysis should only cover the software modules of each framework concerned with the wrapping and linked data conversion functions that are common to the LDAA. Some frameworks or software tools are small and only perform such functions. Therefore, the source code of larger frameworks has been inspected in detail to filter out modules that implement non-comparable functions. The excluded modules have been those implementing certain functions of the data access, integration and storage layer (e.g., Sesame SPARQL implementations and Silk interlinking libraries, among others) as well as tool-specific functions that are not related with the rest of tools (e.g., Stanbol's semantic enrichment of contents). The list of modules that have been included in the analysis of each tool can be examined in the appendix.

5.2 Results

As shown in Tables 2 and 3, the EasyData implementation considerably reduces CC and TD values. Since such measurements are dependent on the program size, it is more accurate to observe the TD ratio and CC density to compare different solutions. In this vein, the TD ratio and CCD are lower for EasyData than for other solutions. The reduced TD has an influence in the maintainability of the solution.

On the other hand, all solutions present a smaller code duplication density than EasyData (see code duplication metrics in Table 3). Some frameworks, such as Hydra and Triplify, also present a better reliability and security remediation cost, measured as remediation hours required to fix bugs and vulnerabilities. That means a need for improvement of the EasyData implementation. Yet the number of vulnerabilities (see Table 4) is fewer for EasyData and HydraBundle, mainly because they make a less intensive use of existing libraries and components that might add security issues.

With respect to the size and complexity (see Table 2), HydraBundle, EasyData and Triplify have lesser complex implementations than other frameworks. Tools like D2Rq and Datalift add an extra complexity, because

Table 2 Size and complexity of NAME compared to other LD frameworks

	Size & complexity metrics				
	Size			Complexity	
Tool	LOC	#statements	#functions	CC	CCD
D2Rq	14,108	6,473	1,516	3,239	0.50
Stanbol	4,701	1,887	352	724	0.38
HydraBundle	2,354	1,098	187	476	0.43
Triplify	1,352	818	79	398	0.49
Datalift	16,037	7,009	1,349	3,043	0.43
NAME	3,773	2,195	133	478	0.22

Table 3 Maintainability and code duplication of NAME compared to other LD frameworks

	Maintainability & code duplication metrics					
		TD		Code duplication		
Tool	#code smells	hours	ratio	#lines	#blocks	density
D2Rq	805	84.5	1.25%	555	31	3.93%
Stanbol	0	44.9	1.99%	229	16	4.87%
HydraBundle	107	15.2	1.35%	369	4	15.68%
Triplify	166	18.5	2.85%	0	0	0.00%
Datalift	1,028	112.3	1.46%	2382	53	14.85%
NAME	21	9.1	0.50%	687	79	18.21%

Table 4 Reliability and security of NAME compared to other LD frameworks

	Reliability & security metrics				
			Remediation (h)		
Tool	#bugs	#vulner	bugs	vulner	Remediation effort (h)
D2Rq	19	205	17	240	445
Stanbol	28	415	22	260	675
HydraBundle	14	280	0	0	280
Triplify	5	100	1	30	130
Datalift	51	430	96	1,175	1,605
NAME	39	335	0	0	335

they make use of a lot of handful libraries that have to be properly integrated and managed in the source code. This criterion is less relevant for EasyData, since the source code of the eventually extended LD application is automatically generated by the framework, so the need to manage code complexity issues, which have to do with the programmers' difficulty for code maintenance, is less relevant.

5.3 Discussion

The EasyData LDAA extension strategy can be exploited to fulfill non-functional features that might be defined for an existing web application. Its aim is not to define a new technique for linking heterogeneous linked data schemata and LD datasets (i.e. interlinking). One reason is that including an interlinking feature in EasyData might constrain the evolution of generated RDF models, whilst the interlinking function can be carried out through readily available tools, as explained elsewhere [54].

As opposed to the black-box wrapping approaches for adding linked data or converting an application's output to linked data, the white-box extension approach of EasyData enables to modify the application components that are needed. When doing that, diverse non-functional requirements can be readily implemented. Thus, this white-box strategy is essential to fulfill security and privacy preservation requirements. Data records that may constitute a publicly relevant dataset are never disclosed from the underlying application implementation without being in the first place secured and privacy-preserved, with considerable savings in complexity and reliability.

The EasyData reengineering method combines the LDAA data wrapping and LDAA API extension in an integrated approach, which exposes the application data model and generates a new LD API, providing also a hook to implement diverse non-functional requirements. In its current implementation, EasyData can provide a unified security access control layer, similar to other LD frameworks, additionally to privacy preservation measures, which are not as common. Thus, external agents can be properly authorized to browsing and accessing an application's resources under a set of privacy restrictions.

Although the privacy preservation rules defined in EasyData are focused on the *k*-anonymity model, these can be extended to other privacy models, such as probabilistic *k*-anonymity [47] or ε-differential privacy [11], without changing the core of the application. It would only be necessary to adjust the current set of sanitization options to a new set of PPDP methods that make it possible to achieve the requirements of a new privacy model. For instance, in an attempt to achieve probabilistic *k*-anonymity, the current sanitization options should be extended to generalization, microaggregation and rank swapping [42]. However, to achieve differentially private datasets, the sanitization option should be replaced by noise addition [41]. On the other hand, to provide protection against attribute disclosure (besides identity disclosure), and thus, offer a stronger privacy guarantee, EasyData could be adapted to combine *k*-anonymity with other privacy models, such as, *l*-diversity [31] or *t*-closeness [30].

5.4 Threats to Validity and Limitations

A major concern of the EasyData white-box approach is that the described LDAA extension strategy is deeply integrated with the internal data model and logic of the legacy web application. This tight coupling eliminates the separation of concerns in a separate linked data layer and implies that changes to the inner workings of the web application may affect the EasyData plugin implementation. In practical terms, this can make maintenance more difficult if the original web application source code is not under the control of the LDAA extension developer. For instance, in the Redmine example it is not straightforward to migrate to new Redmine versions without breaking the EasyData plugin. Black-box approaches, on the other hand, would only require the web application to provide a stable API, which is not always easy though. In this vein, the Hydra approach improves the decoupling of linked data consumer and provider by means of a core vocabulary that can be used to describe and document generic web APIs. The EasyData approach should follow a similar approach to be more general.

Compared to other solutions as Datalift, EasyData does not provide a powerful interlinking technique to map heterogeneous metadata from different web sources. The links to external vocabularies and resources must be discovered by means of an external interlinking tool, and then used to modify the RDF model generated by EasyData. This flexible approach enables to evolve the interlinking result without parsing the model again, but we must rely on an interlinking tool to ensure completeness of the RDF model.

6 Conclusion

The EasyData approach presented in this paper makes it flexible to implement security and privacy properties in a legacy web application using linked data technologies. The LDAA extension approach can be practiced at diverse layers of the architectural components of a web application. In this paper, we have described the application to the controller layer of a regular MVC-based architecture. The EasyData privacy by design procedure is constrained to MVC-based web application architectures as well as the availability of source code. The LD model of a legacy web application can be disclosed and published with privacy preservation through any controller operation. The overall process is not based on adding middleware components, wrappers or adaptors, which can reduce the reliability and maintainability while increasing the complexity of the overall software architecture. A number of internal configurations can make it also possible to prepare for interlinking and alignment of the legacy application data model with external RDF sources. Configuring the application with external models and schemata beyond the legacy application data model is highly recommendable to interlink heterogeneous entities in the Web of data. As a future work, the EasyData implementation is planned to be augmented to connect the generated LDAA extensions with existing interlinking tools.

Overall, our results are subject to the scope of MVC-based application architectures, the use of linked data development frameworks and the implementation of non-functional confidentiality and privacy restrictions. Other web architectures, development technologies and intended non-functional properties should be further evaluated, though EasyData is a promising LDAA extension approach for other realms.

Appendix

Public Evaluation Data

The SonarQube analysis on all the tools and frameworks of this paper are publicly available in sonarcloud.io[7]. The analysis was executed with SonarQube scanner[8]. To extract the relevant metrics for this paper, the sonarcloud.io Web API[9] was used. A JSON output is obtained by means of simple scripts like that of Figure 7. Then the JSON output is converted to CSV[10] to do the analysis on a regular spreadsheet.

All the software modules and packages that are included in the analysis for the more complex linked data frameworks are listed in Figure 8 (Stanbol)

```
HASH='...'
URL='https://sonarqube.com/api/measures/component?componentKey'
COMP=$1
METRICS='cognitive_complexity,functions,sqale_index,vulnerabilities,
    duplicated_lines,bugs,reliability_remediation_effort,
    class_complexity,complexity,statements,ncloc,duplicated_blocks,
    security_remediation_effort,function_complexity,code_smells'
while read COMPKEY; do
 curl -s -u $HASH: $URL=${COMPKEY}\&pageSize=-1\&metricKeys=$METRICS|
 jq '.component | { id, key, qualifier, path, measure: .measures[] }'
done < $COMP.txt
```

Figure 7 Script to query the sonarcloud.io Web API to obtain relevant SonarQube metrics.

```
apache-stanbol-data
org.apache.stanbol.commons.owl
org.apache.stanbol.commons.security.core
org.apache.stanbol.commons.security.usermanagement
org.apache.stanbol.commons.web.base
org.apache.stanbol.commons.web.base.jersey
org.apache.stanbol.commons.web.home
org.apache.stanbol.commons.web.rdfviewable.writer
org.apache.stanbol.commons.web.resources
org.apache.stanbol.commons.web.viewable
org.apache.stanbol.commons.web.viewable.writer
```

Figure 8 Stanbol modules included in the source code analysis

[7] https://sonarcloud.io/organizations/dodero-github/projects

[8] https://docs.sonarqube.org/display/SCAN/Analyzing+Source+Code

[9] https://sonarcloud.io/web_api/

[10] https://konklone.io/json/

```
allegrograph-connector/src/java/org/datalift/allegrograph
core/src/java/org/datalift/core
core/src/java/org/datalift/core/i18n/jersey
core/src/java/org/datalift/core/i18n/web
core/src/java/org/datalift/core/rdf
core/src/java/org/datalift/core/security
core/src/java/org/datalift/core/security/shiro
core/src/java/org/datalift/core/util
core/src/java/org/datalift/core/util/web
data2ontology/src/java/org/datalift/owl
data2ontology/src/java/org/datalift/owl/mapper
database-directmapper/src/java/net/antidot/semantic/rdf/model/tools
database-directmapper/src/java/net/antidot/semantic/rdf/rdb2rdf/
    commons
database-directmapper/src/java/net/antidot/semantic/rdf/rdb2rdf/dm/
    core
database-directmapper/src/java/net/antidot/semantic/rdf/rdb2rdf/main
database-directmapper/src/java/net/antidot/semantic/xmls/xsd
database-directmapper/src/java/net/antidot/sql/model/core
database-directmapper/src/java/net/antidot/sql/model/db
database-directmapper/src/java/net/antidot/sql/model/tools
database-directmapper/src/java/net/antidot/sql/model/type
database-directmapper/src/java/org/datalift/converter/dbdirectmapper
framework/src/java/org/datalift/fwk
framework/src/java/org/datalift/fwk/rdf
framework/src/java/org/datalift/fwk/rdf/json
framework/src/java/org/datalift/fwk/rdf/rio/rdfxml
framework/src/java/org/datalift/fwk/security
framework/src/java/org/datalift/fwk/util
framework/src/java/org/datalift/fwk/util/io
framework/src/java/org/datalift/fwk/util/web
framework/src/java/org/datalift/fwk/view
framework/tests/src/java/org/datalift/fwk/util
s4ac/src/java/org/datalift/s4ac
s4ac/src/java/org/datalift/s4ac/services
s4ac/src/java/org/datalift/s4ac/utils
stringtouri/src/java/org/datalift/stringtouri
```

Figure 9 Datalift modules included in the source code analysis

and Figure 9 (Datalift). As for the simpler tools, such as Triplify, HydraBundle and EasyData, all modules were included in the analysis. In the D2Rq case, all modules were included except `src/de/fuberlin/wiwiss/d2rq/server`.

Acknowledgements

This work was supported by the Spanish Ministry of Economy, Industry and Competitiveness under grants with ref. TIN2017-85797-R (VISAIGLE project) and TIN2016-80250-R (Sec-MCloud project).

List of Abbreviations

API	Application Programming Interface
CC	Cyclomatic Complexity
CCD	Cyclomatic Complexity Density
IS	Information Science
LD	Linked Data
LDAA	Linked Data Application Architecture
LOC	Lines Of Code
LOD	Linked Open Data
MVC	Model-View-Controller
ORM	Object-Relational Mapping
PII	Personally Identifiable Information
PbD	Privacy by Design
PPDP	Privacy-Preserving Data Publishing
QI	Quasi-Identifier
RBAC	Role-Based Access Control
SbD	Security by Design
SPM	Software Process Management
TD	Technical Debt

References

[1] A. Aksac, O. Ozturk, and E. Dogdu. A novel semantic web browser for user centric information retrieval: PERSON. *Expert Systems with Applications*, 39(15):12001–12013, 2012.

[2] M. Amundsen. APIs to affordances: A new paradigm for services on the web. In C. Pautasso, E. Wilde, and R. Alarcon, editors, *REST: Advanced Research Topics and Practical Applications*, pages 91–106. Springer, 2014.

[3] S. Auer, S. Dietzold, J. Lehmann, S. Hellmann, and D. Aumueller. Triplify: Light-weight linked data publication from relational databases. In *Proc. of the 18th Int. Conf. on World Wide Web*, pages 621–630, 2009.

[4] K. Bednar, S. Spiekermann, and M. Langheinrich. Engineering privacy by design: Are engineers ready to live up to the challenge? *The Information Society*, 35(3):122–142, 2019.

[5] C. Bizer and R. Cyganiak. D2R Server – Publishing Relational Databases on the Semantic Web. In *Proc. of the 5th International Semantic Web Conference*, Athens, Georgia, USA, 2006.

[6] M. E. Bonfanti. Enhancing cybersecurity by safeguarding information privacy: The European Union and the implementation of the "data protection by design" approach. In *Proc. of the 13th International Conference on Availability, Reliability and Security*, pages 64:1–64:6, 2018.

[7] F. P. Brooks. *The Mythical Man-Month: Essays on Software Engineering, 20th Anniversary Edition*. Addison-Wesley Professional, 1995.

[8] R. D. Caytiles and S. Lee. A review of MVC framework based software development. *Int. Journal of Software Engineering and its Applications*, 8(10):213–220, 2014.

[9] V. Ciriani, S. de Capitani di Vimercati, S. Foresti, and P. Samarati. Microdata protection. In T. Yu and S. Jajodia, editors, *Secure Data Management in Decentralized Systems*, pages 291–321. Springer-Verlag, 2007.

[10] F. Dotsika. Semantic APIs: Scaling up towards the Semantic Web. *Int. Journal of Information Management*, 30(4):335–342, August 2010.

[11] C. Dwork. Differential privacy. In *Proc. of the 33rd Int. Conf. on Automata, Languages and Programming – Volume Part II*, pages 1–12, Berlin, Heidelberg, 2006. Springer-Verlag.

[12] O. Erling. Declaring RDF views of SQL data. In *W3C Workshop on RDF Access to Relational Databases*, 2007.

[13] O. Erling and I. Mikhailov. RDF support in the Virtuoso DBMS. In T. Pellegrini, S. Auer, K. Tochtermann, and S. Schaffert, editors, *Networked Knowledge – Networked Media*, volume 221 of *Studies in Computational Intelligence*, pages 8–24. Springer, 2009.

[14] EU. Regulation (EU) 2016/679 of the European Parliament and of the Council of 27 april 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing directive 95/46/EC (General Data Protection Regulation), 2016.

[15] B. C. M. Fung, K. Wang, R. Chen, and P. S. Yu. Privacy-preserving data publishing: A survey of recent developments. *ACM Computing Surveys*, 42(4):14:1–14:53, 2010.

[16] G. K. Gill and C. F. Kemerer. Cyclomatic complexity density and software maintenance productivity. *IEEE Transactions on Software Engineering*, 17(12):1284–1288, 1991.

[17] P. Groth, A. Loizou, A. J. G. Gray, C. Goble, L. Harland, and S. Pettifer. API-centric Linked Data integration: The OpenPHACTS Discovery

Platform case study. *Web Semantics: Science, Services and Agents on the World Wide Web*, 29:12–18, 2014.

[18] M. Hausenblas. Exploiting Linked Data to Build Web Applications. *IEEE Internet Computing*, 13(4):68–73, 2009.

[19] T. Heath and C. Bizer. *Linked Data: Evolving the Web into a Global Data Space*. Morgan & Claypool, 2011.

[20] J. Hendler, N. Shadbolt, W. Hall, T. Berners-Lee, and D. Weitzner. Web Science: an interdisciplinary approach to understanding the Web. *Communications of the ACM*, 51(7):60–69, 2008.

[21] A. Hundepool, J. Domingo-Ferrer, L. Franconi, S. Giessing, E. S. Nordholt, K. Spicer, and P. de Wolf. *Statistical Disclosure Control*. Wiley, 2012.

[22] P. Hustinx. Privacy by design: delivering the promises. *Identity in the Information Society*, 3(2):253–255, 2010.

[23] B. Hyland, G. Atemezing, and B. Villazón-Terrazas. Best practices for publishing linked data. Technical Report TR/LDP, W3C, January 2014.

[24] A. Jentzsch, R. Isele, and C. Bizer. Silk – generating RDF links while publishing or consuming linked data. In *Proc. of the ISWC, Posters & Demonstrations Track*, volume 658, pages 53–56, 2010.

[25] B. Jöerg, I. Ruiz-Rube, M.A. Sicilia, J. Dvořák, K. Jeffery, T. Hoellrigl, H. S. Rasmussen, A. Engfer, T. Vestdam, and E. García-Barriocanal. Connecting closed world research information systems through the linked open data web. *International Journal of Software Engineering and Knowledge Engineering*, 22(3):345–364, 2012.

[26] S. Joksimovic, J. Jovanovic, D. Gasevic, A. Zouaq, and Z. Jeremic. An empirical evaluation of ontology-based semantic annotators. In *Proc. of the 7th Int. Conf. on Knowledge Capture*, pages 109–112. ACM, 2013.

[27] S. Kirrane, S. Villata, and M. d'Aquin. Privacy, security and policies: A review of problems and solutions with semantic web technologies. *Semantic Web*, 9(2):153–161, 2018.

[28] M. Lanthaler. Creating 3rd Generation Web APIs with Hydra. In *Proc. of the 22nd Int. Conf. on World Wide Web*, pages 35–38, 2013.

[29] J. Lehmann, R. Isele, M. Jakob, A. Jentzsch, D. Kontokostas, P. N. Mendes, S. Hellmann, M. Morsey, P. van Kleef, S. Auer, and C. Bizer. DBpedia – A Large-scale, Multilingual Knowledge Base Extracted from Wikipedia. *Semantic Web – Interoperability, Usability, Applicability*, 6(2):167–195, 2015.

[30] N. Li, T. Li, and S. Venkatasubramanian. t-Closeness: Privacy Beyond k-Anonymity and l-Diversity. In *23rd Int. Conf. on Data Engineering*, pages 106–115. IEEE, 2007.

[31] A. Machanavajjhala, D. Kifer, J. Gehrke, and M. Venkitasubramaniam. L-diversity: Privacy beyond k-anonymity. *ACM Transactions on Knowledge Discovery from Data*, 1(1), March 2007.

[32] S. Martínez, D. Sánchez, and A. Valls. A semantic framework to protect the privacy of electronic health records with non-numerical attributes. *Journal of Biomedical Informatics*, 46(2):294–303, 2013.

[33] T. J. McCabe and C. W. Butler. Design complexity measurement and testing. *Communications of the ACM*, 32(12):1415–1425, 1989.

[34] E. McCallister, T. Grance, and K. A. Scarfone. Guide to protecting the confidentiality of Personally Identifiable Information (PII). Technical Report SP 800-122, NIST, 2010.

[35] B. J. Oates. *Researching Information Systems and Computing*. Sage, 2005.

[36] E. Oren, R. Delbru, M. Catasta, R. Cyganiak, H. Stenzhorn, and G. Tummarello. Sindice.com: a document-oriented lookup index for open linked data. *Int. Journal of Metadata, Semantics and Ontologies*, 3(1):37–52, 2008.

[37] E. Oren, B. Heitmann, and S. Decker. ActiveRDF: Embedding semantic web data into object-oriented languages. *Web Semantics: Science, Services and Agents on the World Wide Web*, pages 191–202, 2008.

[38] K. Pol, N. Patil, S. Patankar, and C. Das. A survey on web content mining and extraction of structured and semistructured data. In *Emerging Trends in Engineering and Technology*, pages 543–546, 2008.

[39] E. Rajabi, M. A. Sicilia, and S. Sanchez-Alonso. An empirical study on the evaluation of interlinking tools on the web of data. *Journal of Information Science*, 40(5):637–648, 2014.

[40] L. Rocher, J. M. Hendrickx, and Y. de Montjoye. Estimating the success of re-identifications in incomplete datasets using generative models. *Nature Communications*, 10(3069), 2019.

[41] M. Rodriguez-Garcia, M. Batet, and D. Sánchez. A semantic framework for noise addition with nominal data. *Knowledge-Based Systems*, 122:103–118, 2017.

[42] M. Rodriguez-Garcia, M. Batet, and D. Sánchez. Utility-preserving privacy protection of nominal data sets via semantic rank swapping. *Information Fusion*, 45:282–295, 2019.

[43] I. Ruiz-Rube, J. M. Dodero, and R. Colomo-Palacios. A framework for software process deployment and evaluation. *Information and Software Technology*, 59(3):205–221, 2015.

[44] P. Samarati. Protecting respondents' identities in microdata release. *IEEE Transactions on Knowledge and Data Engineering*, 13(6):1010–1027, November 2001.

[45] P. Samarati and L. Sweeney. Protecting privacy when disclosing information: k-anonymity and its enforcement through generalization and suppression. Technical Report SRI-CSL-98-04, SRI International, 1998.

[46] F. Scharffe, G. Atemezing, R. Troncy, F. Gandon, S. Villata, B. Bucher, F. Hamdi, L. Bihanic, G. Képéklian, F. Cotton, J. Euzenat, Z. Fan, P.-Y. Vandenbussche, and B. Vatant. Enabling linked data publication with the Datalift platform. In *AAAI Workshop on the 26th Conference on Artificial Intelligence*, pages 25–30. AAAI Publications, 2012.

[47] J. Soria-Comas and J. Domingo-Ferrer. Probabilistic k-anonymity through microaggregation and data swapping. In *IEEE International Conference on Fuzzy Systems*, 2012.

[48] D.-E. Spanos, P. Stavrou, and N. Mitrou. Bringing relational databases into the semantic web: A survey. *Semantic Web – Interoperability, Usability, Applicability*, 3(2):169–209, 2010.

[49] R. N. Taylor, N. Medvidović, and E. M. Dashofy. *Software Architecture. Foundations, Theory, and Practice*. John Wiley & Sons, 2010.

[50] H. Tillwick and M. S. Olivier. A layered security architecture blueprint. In *Proc. of the 4th Annual Information Security South Africa Conference*, 2004.

[51] US. HIPAA. Health Insurance Portability and Accountability Act, 2002.

[52] V. K Vaishnavi and W. Kuechler. *Design Science Research Methods and Patterns*. CRC Press, 2nd edition, 2015.

[53] H. H. Wang, D. Damljanovic, T. Payne, N. Gibbins, and K. Bontcheva. Transition of Legacy Systems to Semantic Enabled Application: TAO Method and Tools. *Semantic Web – Interoperability, Usability, Applicability*, 3(2):157–168, 2012.

[54] S. Wölger, K. Siorpaes, T. Bürger, E. Simperl, S. Thaler, and C. Hofer. A survey on data interlinking methods. Technical Report 2011-03-31, Semantic Technology Institute, march 2011.

Biographies

Juan Manuel Dodero obtained the BSc and MSc degrees in computer science from the Polytechnic University of Madrid, and a PhD in computer science and engineering from the Carlos III University of Madrid. He has been a Research & Development Engineer in a number of ICT companies. He is currently a Full Professor with the Department of Informatics and Engineering of the University of Cádiz, Spain. His current research interests are related with creative computing, technology-enhanced learning and computational thinking.

Mercedes Rodriguez-Garcia received a BSc degree in computer science from the University of Cádiz, Spain, a MSc degree in ICT security from the Open University of Catalonia, Spain, and a PhD degree in computer science and mathematics of security from the Rovira i Virgili University, Tarragona, Spain. She is currently an Assistant Lecturer with the Department of Automation Engineering, Electronics and Computer Architecture of the University of Cádiz, Spain. Her research and teaching interests include data privacy, computer network security, and reverse engineering and secure architectures.

Iván Ruiz-Rube received his MSc degree in software engineering from the University of Seville and a PhD degree in computer science from the University of Cádiz. He has been a Development Engineer with Everis and Sadiel ICT consulting companies. He is currently an assistant lecturer with the University of Cádiz, Spain. His fields of research are software process improvement, linked open data and technology-enhanced learning.

Manuel Palomo-Duarte received the MSc degree in computer science from the University of Seville, Spain and the PhD degree from the University of Cádiz, Spain. Since 2005 he works as a lecturer in the University of Cádiz. He is the author of three book chapters, 20 papers published in indexed journals and more than 30 contributions to international academic conferences. His main research interests are learning technologies and collaborative web. He was a board member in Wikimedia Spain from 2012 to 2016.