HAL
open science

Privacy Conscious Web Apps Composition

Aurélien Faravelon, Éric Céret

To cite this version:

Aurélien Faravelon, Éric Céret. Privacy Conscious Web Apps Composition. IEEE Research Challenges in Computer Science (RCIS) 2014, May 2014, Marrakech, Morocco. pp.1-1. hal-01003871

HAL Id: hal-01003871

https://hal.science/hal-01003871v1

Submitted on 10 Jun 2014

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Privacy Conscious Web Apps Composition

Aurélien Faravelon
SIGMA team
Grenoble Informatics Lab – 41 rue des Mathématiques
38041 Grenoble Cedex 9, France
aurelien.faravelon@imag.fr

Eric Céret
Grenoble Institute of technology
Grenoble Informatics Lab – 41 rue des Mathématiques
38041 Grenoble Cedex 9, France
eric.ceret@imag.fr

***Abstract*—So called "apps" are widespread today on the Internet. Most of them allow users to extend the range of functionalities their websites offer. However, they potentially jeopardize the privacy of users. Indeed, they collect, store and process personal pieces of information. Recent studies show that users feel they lack control over information. They also show that users distrust apps providers and would rather turn to their friends or family when they choose apps. In this paper we propose a model-driven approach to empower end-users with an extended control over their information. Our work is implemented as a web-based tool to compose apps and manage end-users privacy requirements. Our work showcases the unexploited possibilities of current web protocols and technologies in terms of privacy management.**

Keywords—Privacy, Model-driven Engineering, Web Applications, end-user programming

I. Introduction

Web applications are part of our lives. With more than 1.2 billions of users, Facebook, for instance, is everywhere. Most web applications, be them social networks, musics services or software code repositories – are "social". They allow users to interact and enhance their experience with the one of other users. Most of these applications also allow users to install third party applications. For instance, *Candy Crush Saga*, a well-known online game, is available on Facebook and gathers more than 150 millions of users.

Social features and third party applications are important factors in the success of current web applications, a flourishing economic sector. However, they also generate fears. Recent surveys show that users feel they are loosing control over their data [5][7]. As a result, they do not trust applications providers when it comes to managing their data, especially when they are sensitive. Such feelings are a brake to the acceptance of new applications. They may result in less sharing among users or suing and boycott of some applications. They demand extra attention to privacy management.

Tools are missing to address these feelings. In this paper we address this lack. We focus specifically on privacy management in third-party applications use. We diagnose two causes to the feeling of lack of control that users express. First of all, users cannot freely chose the third party apps they want to work with. They can only select the applications developed for a specific platform – such as Facebook. Secondly, when users select an application, they can usually access the application's privacy policy. However, this policy is often unclear or incomplete. Thus, users cannot actually make an informed consent when it comes to selecting a third-party application.

This problem is at the center of many current debates. For instance, in august 2013, groups of private life defenders sued Facebook. The company agreed to pay 20 millions dollars for using members "likes" as endorsements for advertising, and also to modify its rules to "give members greater control when it comes to how their information is used"[1]. In France, at the same time, a consumer organization says about Google, Twitter and Facebbok that they record, store and process all private information of their users[2]. In january 2014, Google was ordered to pay 150,000 euros by a french government office for its policy about data confidentiality[3]. In United Kingdom, according to a survey, 92% of adult users feel uncomfortable when sharing personal data on social networks, while 55% are feeling more comfortable when sharing these data with governmental institutions, showing that the problem really is about private sector companies[4]. Researchers also notice an increasing reluctance to share personal data[7].

As a result, empowering users with means to better control their data diffusion is essential to foster the acceptance of web applications. Users must be provided with relevant information – they must now who uses the application for instance – and be able to express fine-grained access control requirements. It's challenging. It demands to retrieve a large amount of data and to accommodate the lack of end-users' technical expertise. Most current works on privacy management in application composition are dedicated to expert users and are thus irrelevant to our problem. On the other hand, end-user programming is a promising way of research but has not be applied to privacy management.

In order to leverage this difficulty, we propose a model-driven approach divided in two levels:

---

1 http://tinyurl.com/kqdvmg8
2 http://tinyurl.com/lj5788o
3 http://tinyurl.com/ln5dcpp
4 http://tinyurl.com/l3z5vrj

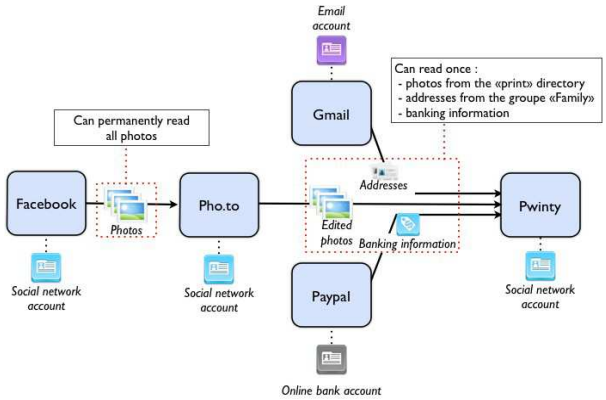

Fig. 1. Composing web applications to edit online pictures

- The **configuration level** allows end-users to describe the composition of a set of applications and the access rights to their data. Users rely on information provided by the developers of the applications. The specification level provides a metamodel divided in two parts. The technical part provides developers with vocabulary to describe their applications. The nontechnical part frees end-users from technical details.
- The **implementation level** allows to run the composition of application. It relies on code generation to produce the composition and the access control code. Automated code generation addresses the lack of end-users technical expertise: they do not have to write code.

Our main contributions are twofold. First, we bring a metamodel to specify a privacy-conscious composition of web applications. Then, we propose an execution platform for this metamodel to execute the specifications of end-users while addressing their lack of technical expertise.

Our work is implemented as a website which allows to compose web applications and specify access rights. This website is realized with the same technologies as the ones most current web application use. It aims at showcasing that these technologies allow to manage privacy more accurately than what is currently done. We apply our website to a real use case in order to demonstrate our approach's feasibility.

The paper is organized as follows. Section 2 presents our use case. In Section 3, we draw a state of the art in order to identify the lacks in current works. Section 4 introduces a global outlook over our approach. In Section 5, we detail our metamodel, its layers and the way we configure web applications to compose them. Section 6 outlines our execution level. Eventually, we presentation our results and our work's validation in Section 7 before discussing our contribution in Section 8.

II. Use Case

In this paper, we posit that a user wants to edit and print pictures from his/her social network profile. Applications to do so already exist. Their composition is represented as a process on Figure 1. Users may edit pictures stored on a social network (here Facebook) with an online editor (here Pho.to) and print them (here with Pwenty) while paying for this service with an online bank account (here Paypal) and send the pictures to an online address book (here Gmail). However current web application do not allow end users to run this process while protecting their privacy for the following reasons:

- **Each application is produced by a specific provider**. Users have to authenticate each time they interact with an application either with an account issued by the application or associated to it. For instance, users can login to the printing application with their social network account. This feature is called “single sign on” (users sign on multiple applications with a single account).
- Applications rely on the same types of data. For instance, three out of the applications we consider process photos. However, **they represent and describe these data types with specific vocabulary**. A common vocabulary to connect applications which process pictures – for instance – is missing.
- **The composition processes sensitive data**. Photos can display intimate scenes and banking information are especially sensitive. When an application gains access to these data, users loose control over what the application will do. For instance, it may sell the data to its business partners. As a result, users should be able to finely control the datasets they share – by restricting the range of the set – and the life time of the access rights. It is currently impossible.
- Binding the applications requires code. This code is currently realized by developers each time they want to connect two applications. **End-users cannot connect two applications if such code does not exist**. For instance, there is no connection between Facebook and Pho.to. Thus, users cannot process their Facebook pictures with Pho.to.

In our work, we strive to answer these difficulties in order to allow the end-users to compose web applications while protecting their privacy of end-users.

TABLE I. COMPARISON OF THIRD PARTY APPLICATION CONTROL MEANS IN FIVE WEB APPLICATIONS

Application	Deezer	Facebook	Google	Twiiter	GitHub
Authentication	Oauth 2.0				
Documentation	http://tinyurl.com/888a82e	http://tinyurl.com/phcf7kg	http://tinyurl.com/phcf7kg	http://tinyurl.com/888a82e	http://tinyurl.com/nbp3zyr
Users must consent to use third-party application	Yes				
Third-party application Identity	Name, logo				
Third-party application description	No				Optional
Provider contact	Non		eEmail	Website	
Customization of access rights	No				
Permission description	List of accessed data		List of what the application can and can't do		List of what the application can do
Level of detail in the description	Low. No list of what the application can do with data. Vague terms.		High	High	Low. No description of the accessed data
Other information	No	Privacy policy	No		

III. STATE OF THE ART

This section presents how composing web applications while enforcing privacy is presented in literature.

A. Composing web applications

Mainstream web applications are often realized as Representational State Transfer (REST) services [13]. REST presents itself as a software architecture style whose structure is an abstraction of the world wide web. It sees this structure as a distributed resource management system. Anything with a name – a picture, a person or a bank account – can be a resource [13]. REST services are web services. However they are distinct from traditional web services defined by [10]. Traditional service-oriented architectures focus on the functionalities of services and their selection. REST, on the contrary focuses on searching and processing resources.

As a result, there is no mainstream composition language for REST services. Traditional service composition languages, such as the Business Process Execution Language (BEPL), needs extensions to accomodate REST services [1][11][13]. Actually, BEPL relies on languages such as the Web Service Description Language (WSDL) or the Simple Object Access Protocol (SOAP) which are different from the Hypertext Transfer Protocol which the REST architecture uses.

Extensions to BEPL are only suitable for expert users. Indeed, end-users are accustomed to web pages and rich interfaces. Thus, REST services composition is often displayed as a mash up. [8] shows that the composition of web services relies on API and resources identified by Universal Resource Identifiers (URI). Authors underline that such a composition is *ad hoc*. Developers write code to bind specific applications but a generic framework to connect applications is missing.

Eventually, user-centered tools, such as Really Simple Syndication (RSS) feeds aggregator – such as Yahoo!Pipes – allow end-users to compose RSS feeds produced by a set of applications. However, such tools do not allow users to modify the feeds by publishing a new message or uploading a new picture for instance.

B. Protecting privacy in web applications

Composing web applications implies sharing sensitive pieces of data. When applications gain access to such data, it becomes impossible to control what they do. They may sell data for instance. As a result, access control is crucial. In REST there exist security protocols which permit to authenticate users and applications and control and applications permissions. Oauth[5] is the most used protocol. Most of the time, when a user wants to share his/her data with a third party application, a pop up appears and asks the user to review the third-party application's permissions. Figure 2 displays an example of such a pop up for Gmail[6].

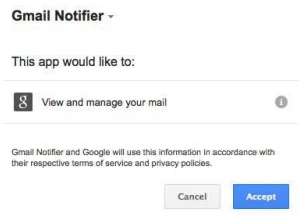

Fig. 2. Pop up to accept the use of a third party application.

Yet, the implementation of protocols such Oauth may turn them inefficient [14]. Sometimes they allow hackers to steal the identity of users. Moreover, the confirmation phase is crucial. It allows users to review the pieces of data the

[5] http://oauth.net/2/

[6] http://gmail.com

application want to access and the effect of such a sharing. However, each web application implements this confirmation phase in its own way. Users may not get all the information they need to make an informed consent or this information may be unclear or imprecise. Table 1 compares the information five web applications provide users with when they want to use third party applications.

The five applications use Oauth 2 a security protocol. However, Oauth is only a specification. Each application has its own implementation of the protocol. All the application allows users to review the permissions the third-party application they want to use requests. No application allows users to restrict the set of data they grant access to. Eventually, some descriptions are vague. For instance, users are warned they grand access to their Facebook "public profile" without the definition of this profile being outlined. Eventually, some pieces of information are missing. Surveys [5][7] show that users rely on people close to them to trust third party applications. No application provide users with information about who uses the third party application they want to use.

C. *Model-driven security, a promising solution*

Web applications process the same type of resources. For instance, most social networks store instant messages. However, each application has a specific representation of these resources. Furthermore, each application relies on a specific implementation of security and implementation protocols. Most of the time, the binding between two web applications is ad hoc: developers create code for specific applications and have to realize this for each application they want to bind their application to.

In Service-Oriented Applications, abstract languages allow to focus on services functionalities and not on their implementation to compose them through a model-driven approach [2]. A metamodel provides the vocabulary necessary to describe the composition of a set of functionalities. Model to code transformations allow to generate the code necessary to run the service composition. Model-driven engineering also allows to specify at an abstract level security features such as access control and generate the code to enforce an access control policy [4], [16]. Such a policy is efficient to protect priacy [4].

However, these approaches are dedicated to expert users. [3] shows that process languages are too complex for end-users. The author shows that identifying abstraction levels allows to gather people with different levels of expertise, such as end-users or developers. Eventually, the author shows that a concrete syntax is necessary to help end-users capture a process.

Modeling is at the heart of end-user programming. However, an approach which reconciles end-user specifications and technical details in order to run a privacy aware composition of web applications is missing.

D. *Conclusion to the state of the art*

Composing Rest services remains complex and end-users cannot do it. Furthermore, users lack tools to configure their data diffusion. As a result, users feel they lack control over their data. Existing solutions do not address these fears. In the rest of this paper, we present a model-driven approach to empower users with control over their data in web applications

IV. Global Approach

Our goal is to allow end-users to process their data with any web application while protecting their privacy. We want to allow users to share their data between a set of applications while controlling their diffusion and the access rights of each application. To do so, we propose a model-driven approach whose global structure is presented on Figure 3. This approach allows end-users to specify the applications they want to use and the datasets they allow these applications to access. End-users do not have to deal with technical complexities. We provide them with high level descriptions of applications and access rights. This description is extracted from an application's description. This description is written by the application's developer. It describes the application's functionalities. It also specifies the data the application needs to access. Such a description often contains numerous technical details and its vocabulary is not adapted for end-users. Thus, the viewpoint of end-users and the view point of developers must be reconciled.

To achieve this goal, our approach is generative and divided in two levels:

- At the configuration level, end-users specify the resources they want to process and how they want to process them. By doing so, they implicitly specify a composition of web applications. They also specify the access rights to their data. In order to guide end-users, we propose a metamodel which provides the necessary vocabulary to express without technical details the resources the user wants to share, the actions he/she wants to perform on it and the access rights to the resources. Our metamodel is divided in two views, a resource view and an access control one. As this metamodel must be available to end-users and developers also and as they do not have the same expertise, we design two abstraction layers for our metamodel. The most abstract level does not contain technical details. End-users can use this layer of abstraction. The technical layer of our metamodel allows developers to describe their applications. As end-users are used to rich interfaces, we pay a great deal of attention to the concrete syntax of our metamodel. This syntax is inspired by web applications in order to remain coherent with the usual experience of end-users.
- The implementation level of our work permits to bind the applications and process the relevant data while enforcing privacy protection. This level relies on composition code which invokes the different applications and ensures their communication. It also relies on access control code with describes the access rights of an application and its properties. All the code is automatically generated from the specifications of end-users. Thus, end-users do not have to write software code.

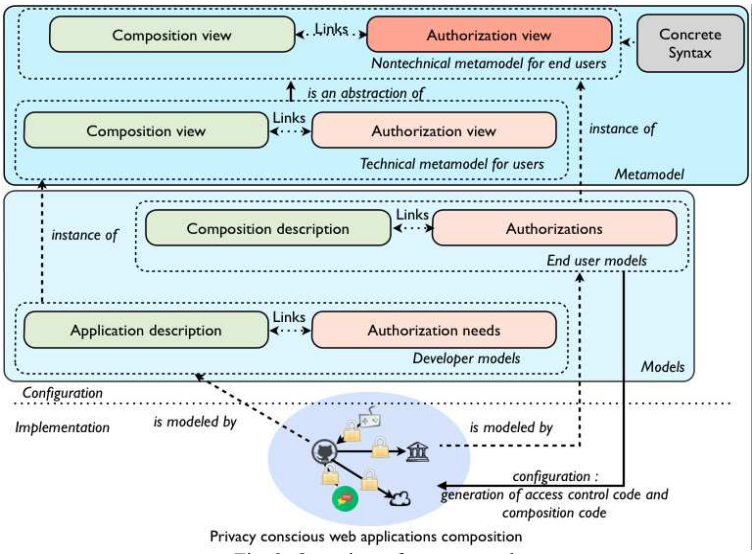

Fig. 3. Overview of our approach

V. Detailed Approach

In this part, we enter into the details of our work. We present our metamodel and its two levels of abstraction: a non-technical levl for end-users and a technical one for developers.

A. Non technical level: metamodel for end-users

End-users do not have technical expertise. We propose an abstraction level of our metamodel which matches the vocabulary they are used to on the web. This level is displayed on Figure 4.

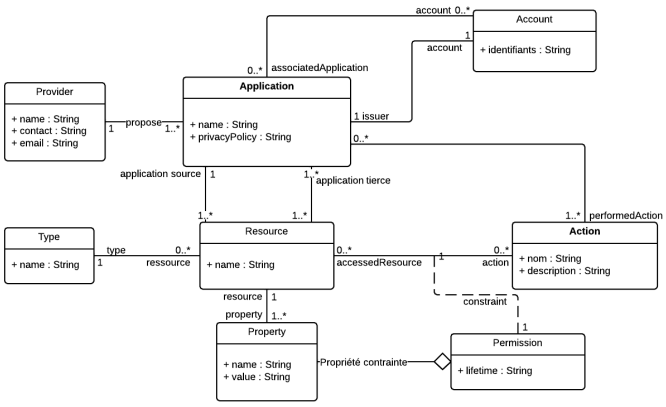

Fig. 4. Non technical metamodel

1) Metamodel entities

The metamodel gathers the following entities:

- A resource is anything which can be named. A picture, an email or a user are example of resources. Resources are identified by their name.
- Each resource as a type. A type has a name. User, Picture or Message are examples of types.
- A resource possesses a set of properties which permits to describe it. For instance, a picture has a name and it may belong to an album.
- A resource is associated to actions: it can be created, updated or deleted.
- An application is a software which stores, creates, processes and modifies resources. A social networking site or an online media player are examples of applications. For a given resource, an application can be a source – it stores resource – or a third-party application: it seeks to retrieve and process the data. Sources and third-party applications have a privacy policy. This policy is a string which describes data it stores and the way it processes this data.
- An application is created and made available by a provider. A provider possesses a name, an email and contact information such as a website.
- In order to use an application, a user must possess an account which stores information such as a login and a password. An account may be issued by an application and used to connect to another application in order to allow single sign on. For instance, a user may use a photo sharing service with his/her social network account.
- When the application plays the role of third-party application towards an resource, it must gain the necessary permissions to consume the resource it needs, *i.e.* to perform actions on the resource it needs. For instance, deleting a picture on a user social network profile may require special privileges. Permissions can be constrained according to a resource's properties. For instance, a user may grant a permission to update only the pictures in the family album.

2) Concrete syntax

Users are accustomed to graphic syntax and rich interface. In order to foster the acceptance of our work, we build a concrete syntax. Table II displays this concrete syntax.

Table II Metamodel's concrete syntax

Metamodel entity	Concrete Syntax	Example
Application	Logo	
Provider	Text or icon	
Account	Text	Bank account
Resource	Text or icon	
Action	Text	Get
Functionality	Text	Edit photo
Permission	Text	Can read

B. Application to our use case

We now instantiate our metamodel for our use case from the perspective of end-users. We add the following constraints to the communication between the applications:

- To retrieve pictures from the social network, an application must have the "read pictures" permission.
- In order to access a set of addresses, an application must have the "read addresses" permission.
- In order to request for a payment, an application must possess the "get payment" permission.

These permissions are inspired from real-world applications. For instance, the Facebook application demands an application to possess permissions to access datasets. For instance, accessing a user's private messages requires the "read_mailbox" permission. Users grant these permissions when they review the data a third-party application accesses.

Table 3 gives an overview of the instances of our metamodel's entities that we identify in our use case.

Table III Instances of our metamodel in our use case

Metamodel entity	Use Case Instance
Application	Social network
	Online photo editor
	Address book
	Photo editing service
Provider	Facebook
	Pho.to
	Pwinty
	Paypal
	Gmail
Account	Social network account
	Email account
	Bank account
Resource	Photo
	Edited photo
	Address
	Baking informations
Action	Get photos
	Get baking information
	Get addresses
Functionality	Photo edition
	Photo printing
Permission	Can permanently read photos
	Can read photos from the "family" album
	Can read once addresses from the group "family"
	Can read once banking information
	Can read pictures
	Can read addresses
	Can get payment

Figure 4 displays a graphical representation of our use case with our concrete syntax.

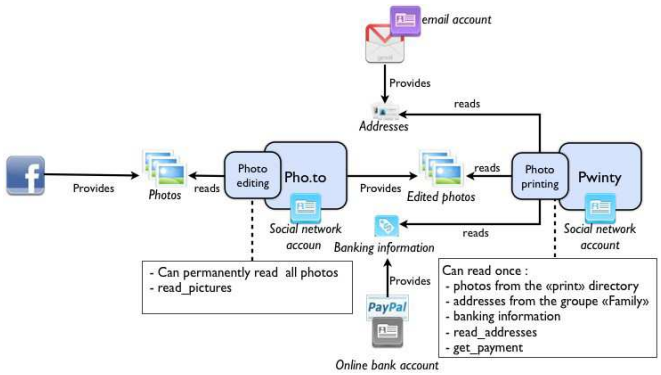

Fig. 4 Graphical representation of our use case

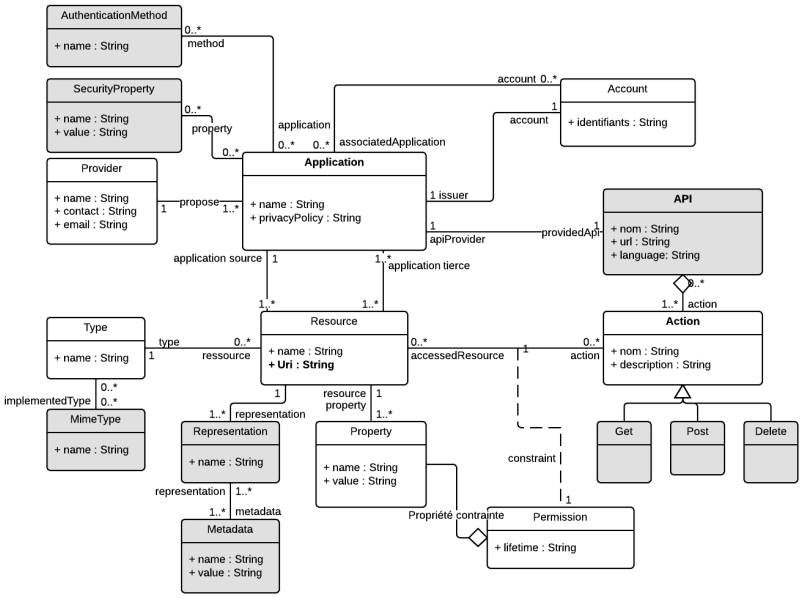

Fig. 5. Technical metamodel

C. *Technical level: metamodel for developers*

In order to generate the code to link a set of applications and the access control code, we need technical information. This code allows to invoke the functionalities of applications and to retrieve and update the data they hold. All the information necessary to produce this code is provided by developers. Currently, such information lies in Application Programming Interfaces. For instance, the Graph Api – Facebook's programming interface, describes Facebook's available functionalities.

The technical model refines the nontechnical level. Figure 5 displays the technical metamodel. It adds technical information to the concepts we have already identified. We refine our metamodel with concepts from Rest architectures. Such information can be found in [13] for instance. We also add information for access control from single sign on protocols such as Oauth. All the technical concepts are displayed in grey.

1) *Metamodel entities*

We now detail the information we add to our metamodel.

In order to access a resource, we need identifiers. We thus add Universal Resource Identifiers (URI) to resources.

According to Rest architectures, resources have several representations. For instance, a picture may have several versions with several sizes and resolutions. Each representation is described by metadata. A metadata provides information about a resource. For instance, the “resolution” metadata gives, for a picture, its resolution.

Web technologies rely on “Internet media types” (Mime types) which define the types of data applications handle. Mime types include image, text or video. We add Mime types as implemented types to our “Type” class.

In Rest architectures, actions are basic HTTP actions. We thus add three subclasses to our Action class, their definitions is inspired by the HTTP protocol's specification:

- the Get action allows to retrieve any resource identified by its URI. For Instance, “Get 012456” will return the resource identified by the number Get/012456.
- the Post action permits to publish data to the target URI. For instance, “Post 012456” adds the resource identified by the number 012456 to the target application. This method allows, for instance, to publish new messages to a social network or upload pictures.
- the Delete actions allows to delete a resource identified by its URI. For instance, “Delete 012456” deletes the resource identified by the number 012456 from the target application.

We refine the association between an application and an action to add the notion of Application Programming Interface (API). An action is part of an API. An application provides an API. This API is used by third-party developers to discover the application's functionalities. Each API has a Universal Ressource Link (URL) and is written in a language such as Java or Php.

Each application accommodates one or several authentication methods. For instance, an application may require a login and a password or allows single sign on protocols such as oAuth.

Eventually, an application requires security properties. Security properties allow an application to connect to another application. For instance, Facebook requires third-party applications to possess an “application number” and a “shared secret”. Such information are necessary to identify and authenticate third-party applications. Third party application

must possess the security properties relevant to connect to the application it targets.

2) Application to our use case

We now apply our technical metamodel to our use case. In most cases, technical information are provided as textual API by developers. We thus do not provide a graphic syntax to this level of our metamodel. We stick to a textual syntax. For each application in our use case, we present the instances of our metamodel's concepts. We only present the technical information for brevity reasons.

Table IV displays our metamodel's instance for Facebook. We only deal with getting pictures on Facebook. We extract data from Facebook's API[7].

Table IV Metamodel's instance for Facebook

Metamodel entity	Use Case Instance
API	API's url: http://tinyurl.com/qf5n7gx
	Languages: Javascript, PHP, etc.
Action	Get/PhotoID
Resource: URI	A number. Example : 119778565
Authentication method	oAuth
Required Security Properties	Access token (permissions granted to an application)
	Application secret (allows Facebook to identify an application)
	Application Id (allows Facebook to authenticate an application)

Table V displays our metamodel's instance for Pho.to. We extract all the information from Pho.to's Api[8].

Table V Metamodel's instance for Pho.to

Metamodel entity	Use Case Instance
API	API's url: http://tinyurl.com/o27cu89
	Languages: HTTP
Action	Get/use_auto_red_eye
Resource: URI	An Image URL
Authentication method	Key issued by Pho.to.
Required Security Properties	Key (allows Pho.to to authenticate an application).

Eventually, Table VI presents our metamodel's instance for Pwinty. We extract all the information from Pwinty's Api[9].

[7] https://developers.facebook.com/docs/graph-api/

[8] http://files.pho.to/documentation/editor-platform/online_photo_enhancement_platform_api.htm#item_4_2

[9] http://www.pwinty.com/Overview

Table VI Metamodel's instance for Pwinty

Metamodel entity	Use Case Instance
API	API's url: http://tinyurl.com/oodrcox
	Languages: HTTP
Action	Post/Order (creates an order)
	Post/Orders/{orderId}/Photos (adds photos to an order)
	Post/Orders/{orderId}/Status (validates the order)
Resource: URI	A number. Example: 1605 is an order's URI.
Authentication method	oAuth
Required Security Properties	X-Pwinty-MerchantId
	X-Pwinty-REST-API-Key

All this information allows to call each application and its functionalities when they are relevant to our use case. As we want to automatically produce the code necessary to call each application, we now describe our code generation process.

D. *Request templates for code generation*

Code generation is responsible for producing the code necessary to invoke each application. The technical information allows to define request templates. A request templates describes the generic form of a call to an application. templates are fed with actual information in order to invoke an application.

Figure 6 displays the request template which describes the generic request to retrieve a picture from Facebook:

```
Get /{photoId}&ApplicationId={applicationId}&Accesstoken={accesstoken}
```

Fig. 6. A request template

When a specific application wants to retrieve a photo from Facebook, it feeds the template with the relevant pieces of information. For instance, in order to retrieve the picture with URI “1265873”, the application with the application id “178656” and the access token “867567509” will issue the request displayed on Figure 7.

```
Get /1265873&ApplicationId=178656&Accesstoken=867567509
```

Fig. 7. Actual request to retrieve a picture on Facebook

At the end of the configuration process, we have captured a user's needs in terms of the composition of applications. We have also captured the user's privacy preferences in terms of data access.

We now present our implementation level, which is responsible for executing the captured requirements and feeding our code templates with actual information.

VI. EXECUTION LEVEL

In our state of the art, we have identified the several lacks in current web applications:

- Users can only select a limited range of third party applications.
- Users have limited control over data third party applications access.
- Users are not provided with all the information necessary to make an informed consent.

As we cannot modify mainstream web applications such as Facebook in order to address these lacks, we propose a dedicated architecture to execute a privacy conscious web application composition. This phase relies on code generation from the specification of users.

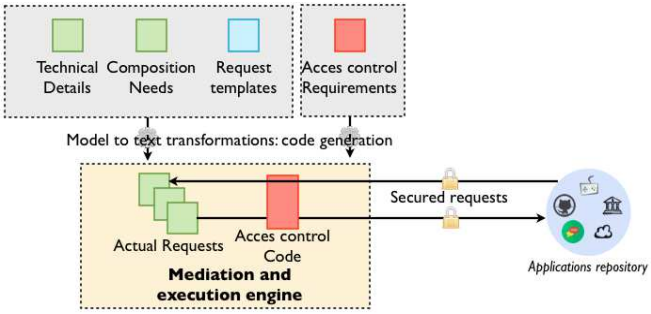

Fig. 8. Overview of our architecture

Figure 8 gives an overview of our architecture. We can identify three main components:

- Model to text transformations allow to turn user's requirements into code. They are responsible for linking the non-technical models of an end-user to technical information necessary to run the desired composition.
- The mediation and execution engine executes the privacy aware composition according to the generated code.
- The applications repository stores the description of the APIs of all the available applications.

We present each component in the rest of this section.

1) Access control code generation process

According to a user's access control requirements we generate an access control token for an application. An access control token is a piece of code which contains all the access rights of an application for a specific set of resources.

```
[Token]
 {userId}
 {appplicationId}
 {privilege}
 {lifetime}
[End]
```

Fig. 9. Access token template

In order to generate the token, we rely on an access control token template which describes the generic form of a token. Figure 9 presents our access control token template.

Our token contains the following elements:

- A user id: the identifier of the current user.
- An application id: the identifier of the third party application.
- Privileges: a set of pairs : <{property, value}, resource> which describes a resource a third party application is granted access to and a set of values its properties should have.
- Each token is associated with a lifetime which states how many times the token is valid.

The token template is fed with information from an API description end-user's access control requirements. Figure 9 displays a token to restrict Pwinty's access to the pictures in the “Family” album.

```
[Token]
 userId=87876
 applicationId=Pwinty
 privileges=<{album,family},photo>
 lifetime=once
[End]
```

Fig. 9. Composition code template

2) Composition code generation process

Composing two applications necessitates to call a source application to retrieve its data and transmit the retrieved pieces of data to a third party application. This transmission is itself a request to the third party application. Thus each composition phase relies on two request templates: one for a source application, one for the third party application. Requests to retrieve data can only bear on allowed sets of resources.

In order to link requests, we use variables: data necessary retrieved from a source application as stored in a variable before their transmission to the third party application.

```
[Composition]
 {MimeType} {variablename} = {request}
 {request {variablename}
 ...
[End]
```

Fig. 10. Composition code template

Once again, this phase relies on templates. Figure 10 displays the composition template. This templates is used at each step of the composition.

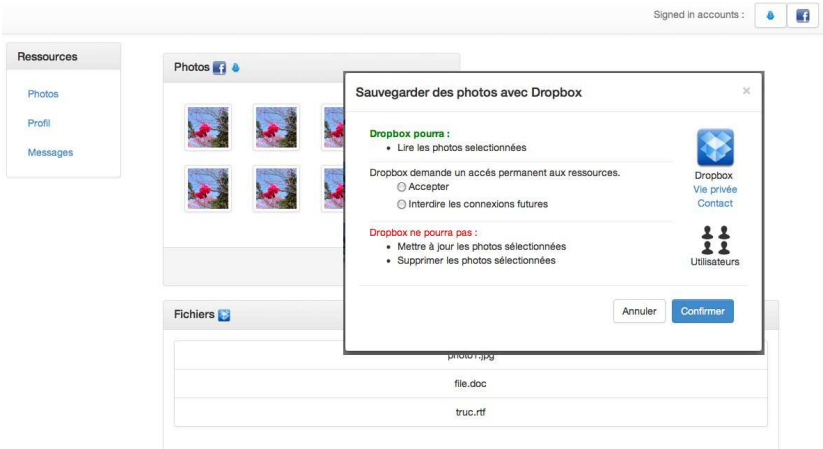

Fig. 12. Interface of our prototype

This template is fed with information from APIs descriptions. It is also fed with an application access token in order to ensure that a third party application only accesses allowed sets of resources. Figure 10 displays a part of the composition code corresponding to our use case.

3) Composition execution and access control enforcement: the mediation and execution engine

The mediation and execution engine is equivalent to an orchestrator in Service-oriented architectures. It permits to compose the applications while enforcing access control. To do so, it calls the templates we have described, feeds them with relevant information and issues requests to the applications. Implementation

We have developed a prototype to allow users to specify the composition of a set of applications and configure their access control preferences for each application. This prototype consists in a mash up of the applications an end-user actually uses. It has two main parts, a web page, which provides an interface to interact with end-users, and a mediation and execution engine which runs the composition.

Figure 12 displays the interface of our prototype. The interface contains the following elements:

- a menu presents the resources the applications hold. Such resources include pictures or friends. A contextual menu allows, for each resource to perform an action with a dedicated application. For instance, a user may chose to synchronize a set of pictures with his/her dropbox or edit this set.
- When the user wants to transfer data from one application to the other, a pop up asks the user to review the access rights he/she grants the application. The user may also restrict the datasets an application has access to.

Our prototype is implemented with mainstream web technologies. The client-side are plain Javascript and HTML.

The mediation and execution engine is written in Javascript. All the requests to the application are regular HTTP requests rewritten to enforce access control. The mediation and execution engine also stores a repository of all the applications available to end-users.

Our prototype aims at showcasing that access control can be more finely enforced in web application. We bring two new abilities to end-users. First, en users can restrict an application's privileges. They can either select a subset of data an application gets access to, or restrict the lifetime of a permission. Then, it is customary in marketing to identify groups of similar users – i.e. users who share the same interests and the same sociological profile. Identify these group allows to predict a user interests: users are likely to like and do what people who look like them like and do. This sociological phenomenon is called “assortativity”.

Surveys shows that the same stands for applications use. end-users do not trust service providers. However they trust people they feel are close to them and look like them to select an application. However, as we have said, no web application provides information about who uses a third-party application when end-users have to review the access rights they grant to this application. We answer this lack by selecting users who use the third party application, are close to the current end-user and share the same interests and close profile. We then present the end-user with the list of the selected users.

The selection phase follows two steps:

- Close users selection. Most users are connected to a large number of other users. Even if they call them “friends” on Facebook, they may not actually know or trust them. Some social networks, such as tweeter, encourage week links – users can “follow” each other or share each other's messages. However, these links may not denote a strong connection between users. Thus, we select users close to a end-user by selecting the set of users the end-users interacts and shares the most with, be it by posting content to their profiles or exchanging private messages. All interactions may not have the same significance. For instance “poking” on Facebook is rather neutral and meaningless when frequent private messages show an actual relationship between users. Given a set of interactions I defined by $I = \{I_1,...I_n\}$, each interaction is associated to a weight w, a positive

integer. The degree of interaction between users is given by the weighted sum of all the interactions between users.

- Similar users selection. Among the users close to the end-users, some users share the same tastes. These users are especially important as the end-user is likely to use the same applications as them. Given a set of interests – for instance music or applications – and set of features – age, location, etc. – the degree of similarity between users is given by the number of similar interests and features they share.

VII. Validation

A. Relevance of our prototype

We have tested our close users selection process on real-world Facebook data. To do so, we have developed a Facebook application which analyzes a user's friends list and retrieve the user's 8 closest and most similar friends. To do so, the application first selects the users a user interacts the most with. To do so, the application accesses the user's inbox and analyzes the messages the user posts to the profiles of other users. Then, the application selects the users who share the more interests with the current user.

In order to validate our work, we have selected 8 Facebook users. Each of them is connected to more than 500 other users. We have asked each user to use our application and to answer following questions:

- How many of the height selected people would you call “close”?
- If these people used an application, would it help you to trust the application?

At most, we were able to guess 8 actual close friends. At worse, we guessed 5. All the users have answered that knowing they close friends use an application helps them to trust the application.

These results have helped use to attune the way we select close friends. Even though we have only submitted our approach to a limited amount of users, their answer confirms what other surveys state and let us think that presenting users with social information when they install third party applications is necessary.

B. Relevance of the proposed metamodel

The approach presented in this paper is an adaptation of the approach we have already proposed in [4]. In this previous work, the authors described how to protect privacy in service oriented architecture : privacy is defined at design level and a service orchestration is generated to offer a secured and privacy respectfull application. This first approach has been validated on two main facets: response times and data protection. It has been shown that response times are increased by only 1%, and that it is not possible to access data without being controlled by the proposed mechanism.

The approach proposed in this paper being very close to the previous one. It relies on the same principles. Data security is the same. For instance, private pictures are protected because all requests which seek to retrieve pictures are rewritten in order to enforce access control.

However, the response times might be influenced by the languages we use in this new proposal and most of all by the dependency to social networks performances. The tests we have driven show that Facebook requests are quite slow and it takes 5 seconds to get all the needed data.

VIII. Discussion and Future Works

Web apps users feel they lack control over their data. Most of them do not trust web app providers when it comes to managing their data. Eventually, users lack tools and relevant pieces of information to control the diffusion of their data. We have tackled these issues in this paper. We brought in a model-driven approach which allows user to compose web apps while tuning an app's access rights to a user's data. Our approach empowers users with two level of control over their data.

First of all, users can invoke any app. Nowadays, users can only select apps registered with a provider, such as *Facebook*. Our approach allows user to select any app implemented as a Restful service – the widest used type of app. Users, which do not possess technical expertise, do not have to produce code to make their apps interact. We rely on code generation to produce the necessary code. Technical information can already be harvested from APIs. In order to show the instantiation of our metamodel, we have extracted the information from each API by hand. However, works such as [9] underlines that APIs can be easily turned self-descriptive and automatically processed. Such works ensure that our automatic generation process is actually feasible.

Our access control token generation allows to generate finer access control than current access control tokens in web applications

Then, we allow users to assess an app's trust and finely tune its access rights to a dataset. This is currently impossible in all the most famous web applications – such as social networks – which allow users to install third-party applications. When users install third-party applications, they are not provided with information – such as who uses the application. Furthermore, the least privilege principle – which is fundamental in access control – is not enforced on the web. Third-party apps request access rights aend-users cannot but accept them if they want to use the app. Our approach enables users to restrict the dataset they want to share with an application and configure the lifetime of an app's access rights.

Our approach is implemented as a website to share a set of resources – pictures, messages, etc. - among web applications. This website demonstrates the feasibility of our work. As a preliminary validation step, we have presented our work to a set of users. They all confirm that the pieces of information we provide them is relevant and useful to trust web apps.

We do not aim at developing yet another web application. We want to showcase what web technologies allows in terms of privacy management. Our work shows that bringing together well-mastered concepts such as model-driven engineering and access control and current web technologies empowers users with new means to control their data. As such, our work shows that privacy requirements elicitation and management can be processed more effectively with widely used technologies. Our work could thus be easily integrated to an already existing application.

Our approach is realistic to the extent that their already exists web apps repositories such as programmableweb.com. Users could select and invoke pretty much any app from such repositories. Furthermore, enhancing existing applications with privacy-conscious features is a growing trend. Blackphone.ch, for instance, extends Android with communication cyphering capabilities when secret.ly proposes to share without communicating one's identity.

Our future works will deal with integrating our work with existing apps and extending the range of privacy features we offer. We will also test our work with a larger set of users.

REFERENCES

[1] W. Budan, L. Rongheng, C. Junliang. "Integrating RESTful Service into BPEL Business Process on Service Generation System". Proceedings of the IEEE Service Computing Conference, pp. 527-534, 2013.

[2] S. Chollet,, P. Lalanda. Security Specification at Process Level. IEEE Service Computing Conference, pp. 165-172, 2008.

[3] M. Cortes Cornax. "Service choreographies through a graphical notation based on abstraction layers and viewpoints". IEEE RCIS 2011. p. 1-11..

[4] A. Faravelon, S. Chollet, C. Verdier, A. Front, . Configuring Private Data Management as Access Restrictions: From Design to Enforcement. *International Conference on Service Oriented Computing*, pp. 344-358, 2012.

[5] F-Secure. "Digital lifestyle survey", 2013. http://www.f-secure.com

[6] C. A. Hill, E. Dean, J. Murphy, Social Media, Sociality, and Survey Research, John Wiley & Sons, 2013.

[7] Microsoft. "Data privacy day privacy survey", 2013. http://tinyurl.com/lcvoqaw

[8] H. Meng-Yen, L. Hua-YiL. Kuan-Ching Li. A web-based travel system using mashup in the RESTful design. International Journal of Computational Science and Engineering. Vol. 6, n°3, p. 185-191.

[9] L. Panziera, F. Paoli. "A framework for self-descriptive RESTful services" WWW (Companion Volume), pp. 1407-1414, 2013.

[10] M. Papazoglou. Service-Oriented Computing: Concepts, Characteristics and Directions. Proceedings of the Fourth International Conference on Web Information Systems Engineering, pp. 3-12, 2003.

[11] P. Pautasso. RESTful Web service composition with BPEL for REST. Data Knowl. Eng., vol. 68, n°9, p. 851-866.

[12] R. Fielding, R. Taylor. Principled design of the modern Web architecture. ACM Trans. Internet Techn., vol. 2, n°2, p. 115-150.

[13] F. Rosenberg, F. Curbera, M. J. Duftler, R. Khalaf R. Composing RESTful Services and Collaborative Workflows: A Lightweight Approach. IEEE Internet Computing, vol. 12, n°5, p. :24-31.

[14] S. Sun, K. Beznosov. "The devil is in the (implementation) details: an empirical analysis of OAuth SSO systems". ACM Conference on Computer and Communications Security, pp. 378-390, 2012.

[15] T. Stoitsev, S. Scheidl, F. Flentge, M. Mühlhäuser. "Enabling End Users to Proactively Tailor Underspecified, Human-Centric Business Processes: "Programming by Example" of Weakly-Structured Process Models". ICEIS pp. 307-320, 2008.

[16] C. Wolter, A. Schaad. Modeling of task-based authorization constraints in BPM'07, pp. 64-69, 2007.