HAL
open science

Programmation par les utilisateurs finaux : Composition d'applications Web respectueuse de la vie privée

Aurélien Faravelon, Eric Céret, Christine Verdier

To cite this version:

Aurélien Faravelon, Eric Céret, Christine Verdier. Programmation par les utilisateurs finaux : Composition d'applications Web respectueuse de la vie privée. INFORSID 2014, May 2014, France. pp.325-362. hal-01003496

HAL Id: hal-01003496

https://hal.science/hal-01003496

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.

Programmation par les utilisateurs finaux : Composition d'applications Web respectueuse de la vie privée

Aurélien Faravelon[1], Eric Céret[2], Christine Verdier[1]

1. Laboratoire d'informatique de Grenoble, Équipe SIGMA
220 rue de la chimie, F-38400 Saint Martin d'Hères, France
2. Laboratoire d'informatique de Grenoble, Équipe IIHM
220 rue de la chimie, F-38400 Saint Martin d'Hères, France
{prenom.nom@imag.fr}

RESUME. *Les applications web comme Facebook, Gmail ou Dropbox connaissent un large succès. Chacune propose un ensemble de fonctionnalités que l'utilisateur peut parfois composer en installant dans une application source une application tierce. La composition est ainsi un moyen, pour l'utilisateur, de bénéficier de fonctionnalités étendues. Cependant, la composition se heurte à deux problèmes. L'utilisateur ne peut pas composer à sa guise les applications puisque la composition nécessite la production de code de médiation entre les applications. Les données échangées entre les applications doivent être contrôlées pour respecter la vie privée de l'utilisateur ce qui n'est pas aujourd'hui complètement possible. Dans cet article, nous cherchons à permettre aux utilisateurs finaux de composer à leur guise leurs applications web tout en configurant le respect de leur vie privée. Pour ce faire, nous proposons une approche dirigée par les modèles de configuration des applications web et de leur composition. Cette approche est implémentée sous la forme d'un environnement de spécification et d'exécution d'une composition d'applications web. Cet environnement est utilisé sur un cas d'étude tiré de la vie réelle.*

ABSTRACT. *Facebook, Gmail and Dropbox are only three examples of successful web applications. Each of them provides specific functionalities and application composition allows user to extend the range of functionalities they can benefit from. However, they rely on mediation code produced by expert developers. As a result, end users cannot freely compose web applications. They cannot precisely configure their privacy preferences either. These two constrains impair users experience. Our work seeks to deal with this lacks. We propose a model-driven approach which permits end users to specify the composition of their web applications and their privacy preferences. Our approach allows to generate the execution code so that end users do not have to rely on developers. Our work is implemented as a modeling and execution tool. We use our tool to realize a real-world composition*

MOTS-CLES : *vie privée, application web, programmation par les utilisateurs*

KEYWORDS: *privacy, web applications, end-user programming*

1. Introduction

Le web ressemble, pour ses utilisateurs, à un vaste système d'information distribué. Réservation de voyages, facturation, la plupart des activités peuvent aujourd'hui être exécutées en composant des applications web. Souvent, ces applications peuvent être connectées les unes aux autres. Ainsi est-il possible de se connecter à un site de réservation comme *hotels.com*[1] avec les identifiants d'un réseau social comme *Facebook*[1].

Dans le domaine des entreprises, l'exécution d'un processus intégrant un ensemble d'applications existantes est bien maîtrisée. Elle permet notamment de s'adapter aux applications disponibles et d'introduire de la flexibilité dans l'exécution des processus. L'intégration des « applications web » par et pour les utilisateurs finaux reste, elle, un domaine de recherche récent. Aujourd'hui, les applications sont composées de manière *ad hoc* : les développeurs créent des interfaces entre deux applications spécifiques et les rendent disponibles aux utilisateurs. Les utilisateurs finaux ne peuvent pas composer à leur guise les applications qu'ils utilisent, leurs fonctionnalités et leurs ressources. On peut identifier deux difficultés principales qui empêchent les utilisateurs de composer leurs applications :

- La composition d'applications web nécessite la compréhension d'API et la production de code informatique, deux opérations qui ne sont souvent pas à la portée des utilisateurs finaux.

- Le partage d'informations entre des applications fournies par des vendeurs variés pose de nombreux problèmes en termes de vie privée. Une fois l'accès aux données autorisé pour une application, il est quasiment impossible de vérifier que cette dernière n'utilisera pas les données d'une façon imprévue par l'utilisateur ou ne les revendra pas, par exemple. Des protocoles comme oAuth[1] existent aujourd'hui pour limiter l'accès d'applications tierces aux données d'un utilisateur. Cependant, l'implémentation actuelle de ces protocoles n'offre qu'un contrôle limité aux utilisateurs finaux : en général, les possibilités de paramétrage des droits d'accès sont restreintes. Ces difficultés à offrir aux utilisateurs la maîtrise de leurs données personnelles est, depuis 2010, au centre d'un vif débat entre des acteurs majeurs d'Internet (comme Facebook ou Google) et les associations de consommateurs, les organismes de contrôle (comme la CNIL en France) et même l'Union Européenne[2]. Les utilisateurs ont ainsi tendance à se tourner vers les applications les plus simples à configurer et qui leur donnent le plus de contrôle sur leurs données[3].

Dès lors, la composition d'applications web par et pour les utilisateurs finaux représente un défi. Elle permet potentiellement aux utilisateurs de tirer pleinement

---

[1] voir http://fr.hotels.com, https://www.facebook.com et http://oauth.net/2

[2] La justice américaine a condamné Facebook pour avoir utilisé des données personnelles dans ses publicités. Début 2014, en France, Google a été doublement condamné par la CNIL puis par le Conseil d'Etat. Voir le dossier de l'Express du 07/02/2014.

[3] Voir le récent reportage de *Libération* à ce sujet : http://tinyurl.com/pz9zvlz

parti des applications qu'ils utilisent et des données qu'ils partagent. Les recherches existantes n'apportent qu'une réponse partielle à ce domaine : l'ingénierie des processus métiers, surtout lorsqu'elle prend en compte des propriétés comme la protection de la vie privée, s'adresse à des experts, les concepteurs d'applications qui ont des connaissances techniques. Le *end-user programming* (Jones 1995) se donne, lui, pour but de permettre aux utilisateurs d'étendre leurs applications. Il représente une direction intéressante mais il manque un lien entre le *end-user programming* et le niveau technique d'implémentation des applications. Implémenter une composition d'applications respectueuse de la vie privée spécifiée par un utilisateur final nécessite ainsi de trouver un langage commun aux utilisateurs finaux et aux développeurs.

Cet article présente une adaptation de l'approche centrée sur la protection de la vie privée dans les architectures orientées services de (Faravelon *et al.* 2012). Cette approche propose de spécifier la vie privée au niveau conception et de générer la SOA qui permet d'exécuter une application sécurisée et respectueuse de la vie privée. Nous adaptons l'organisation en deux niveaux, spécification et configuration/implémentation pour permettre (a) aux développeurs de décrire l'API proposée pour leurs applications, (b) aux utilisateurs de décrire la composition des API des applications qu'ils ont choisies et les droits d'accès aux ressources associés à cette composition et (c) de générer la couche applicative d'exécution correspondante.

L'article est organisé de la manière suivante. La Section 2 introduit notre cas d'utilisation. Dans la Section 3, nous dressons un état de l'art afin de montrer les manques que l'on peut identifier dans les recherches actuelles. Nous présentons ensuite notre approche d'un point de vue global dans la Section 4 avant de détailler dans la Section 5 les métamodèles que nous proposons et les outils de configuration automatique des applications web. Enfin, dans la Section 6, nous présentons l'implémentation de notre approche et son application à notre cas d'utilisation avant de discuter notre travail et de tirer des perspectives de recherche dans la Section 7.

2. Cas d'utilisation

Nous prenons dans ce papier l'exemple d'un procédé d'impression en ligne d'un ensemble de photographies, présenté sur la Figure 1. Les applications nécessaires à sa réalisation existent déjà. Les photographies d'un utilisateur peuvent être extraites d'un réseau social comme *Facebook* et retouchées grâce à un service en ligne comme *Pho.to*[6]. Enfin, des applications d'impression en ligne comme *Pwinty*[4] permettent d'imprimer les photos et de facturer l'impression. Cependant, la réalisation du procédé se heurte aux difficultés suivantes :

- Chaque application est proposée par un fournisseur différent. L'utilisateur doit ainsi s'authentifier auprès de chaque application soit à l'aide d'un compte propre à l'application, soit avec un compte associé à une autre application.

---

[4] voir http://pho.to/editor-platform et http://www.pwinty.com/Overview

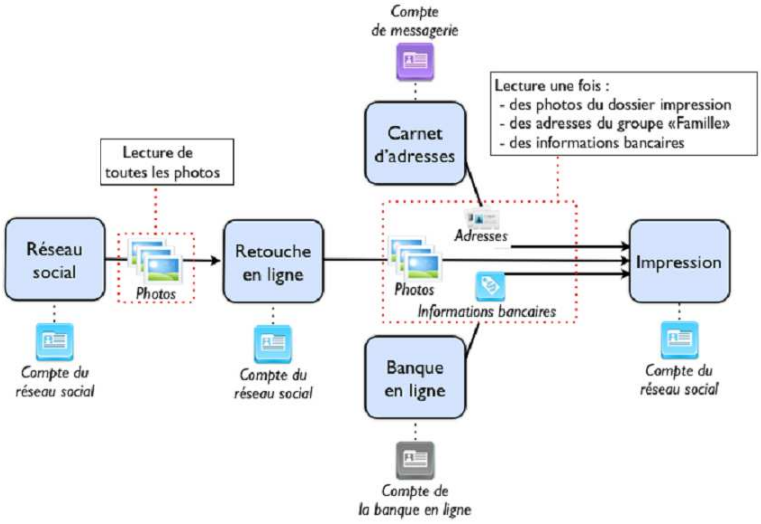

Figure 1. Un exemple de processus intégrant des applications web

- Les applications manipulent des types communs de ressources (dans cet exemple, toutes les applications traitent des images) qu'elles représentent d'une manière qui leur est propre au travers d'un vocabulaire spécifique.

- Le procédé manipule des données intimes – les photographies – et sensibles – les informations bancaires. L'utilisateur doit pouvoir contrôler précisément l'accès aux données et la durée de cet accès. Cette possibilité n'est pas offerte actuellement par les applications web,

- La liaison entre les applications est aujourd'hui réalisée de manière *ad hoc* par les développeurs. Par exemple, il n'existe pas de connecteur entre *Facebook et Pho.to*. L'utilisateur final ne peut pas lui-même connecter une application à son réseau social s'il n'existe pas de connecteur.

- Enfin, l'utilisateur est habitué à une présentation orientée ressources dans laquelle il réalise un processus par des enchaînements de tâches, sans que ce processus soit présenté explicitement sous forme de graphe d'activités.

Dans le cadre de notre travail, nous cherchons à répondre à ces quatre difficultés afin de permettre l'ingénierie de processus intégrant des applications web tout en prenant en compte la protection des ressources des utilisateurs finaux. La section suivante présente les travaux existants dans ce domaine.

3. État de l'art

Aujourd'hui, les applications web sont composées de manière *ad hoc* : les utilisateurs finaux peuvent connecter une application à une application tierce parmi celles proposées par les développeurs. Nous revenons dans cet état de l'art sur la

difficulté de composer ces applications et les enjeux de la prise en compte de la vie privée.

3.1 Composition de services web

La composition de services web repose le plus souvent sur la réalisation d'un processus spécifié à l'aide du *Business Process Modeling* Language (BPMN) ou du *Business Process Execution Language* (BEPL). Cependant, dans le cadre de l'utilisation grand public du web, les services sont réalisés sous la forme de services *Representational State Transfer* (REST) définis notamment dans (Fielding *et al.*, 2002). REST est un "style architectural" qui constitue une abstraction de la structure du World Wide Web et l'envisage comme un système distribué de gestion de ressources. Une ressource est une entité qui peut être nommée – comme une photo, une personne ou un compte bancaire (Fielding *et al.*, 2002). Si les services REST sont considérés comme des services web, l'importance de la notion de ressources dans REST le distingue des architectures orientées services (SOA) habituelles, telles qu'elles sont définies dans (Papazoglou 2003) notamment. Les SOA mettent en effet le plus souvent l'accent sur les fonctionnalités offertes par les services, leur sélection et leur invocation alors qu'une architecture REST met l'accent sur la recherche et le traitement d'un ensemble de ressources.

À cause de la différence d'architecture et d'implémentation entre REST et les SOA, l'intégration de services web et de service REST n'est pas triviale et nécessite l'extension de langages comme BEPL, comme l'ont proposé différents travaux (Wu *et al.* 2013) (Rosenberg *et al.* 2008) (Pautasso 2009). En effet, BEPL repose sur des protocoles de description des services web comme le *Web Service Description Language* (WSDL) ou de communication comme le *Simple Object Access Protocol* (SOAP) qui ne sont pas utilisés dans les architectures REST.

Enfin, ces extensions ne sont destinées qu'aux concepteurs d'applications et pas à leurs utilisateurs finaux. En effet, ces derniers sont habitués aux applications web comme un ensemble de pages qui présentent des ressources qu'ils peuvent consulter ou modifier grâce à des liens. La présentation de la composition de services REST prend ainsi souvent la forme de l'association du contenu offert par un ensemble de services présenté sous la forme d'une page web appelée *mashup*. (Hsieh *et al.* 2011) montre ainsi que la composition de services REST se fait au travers d'API et de l'utilisation de ressources identifiées par leurs URI. Les auteurs remarquent que le développement d'applications à partir de services REST est aujourd'hui faite de manière *ad hoc* : un développeur construit une application à partir d'une API existante. Contrairement aux architectures orientées services, il n'existe ainsi pas pour les architectures REST l'équivalent d'un langage abstrait de composition de services comme celui proposé par (Dami *et al.* 1998).

D'autres paradigmes d'agrégation existent, comme les agrégateurs de flux *Really Simple Syndication* (RSS). Yahoo!Pipes est un exemple de ces agrégateurs. Ces techniques ne permettent que de consulter le contenu résultant d'un ensemble de

services REST et pas de modifier ce contenu, par exemple en publiant un nouveau message ou une image.

3.2 Prise en compte de la vie privée dans les applications web

La composition de services web repose sur le partage d'informations potentiellement sensibles entre des applications fournies par des développeurs différents. Dans la mesure où il est difficile de contrôler ce que font les applications tierces des données une fois qu'elles en ont obtenu l'accès, le contrôle de l'accès aux ressources est crucial. Dans les architectures REST actuelles, ce contrôle repose sur des protocoles dédiés à la sécurité qui permettent d'authentifier les applications qui souhaitent accéder aux données d'un utilisateur et de restreindre leur accès à ces données. Oauth est le protocole le plus utilisé aujourd'hui. Le plus souvent, lorsqu'un utilisateur souhaite donner l'accès à ses données à une application tierce, une fenêtre apparaît et lui demande confirmation. La Figure 2 présente un exemple de fenêtre sur *Gmail*[5].

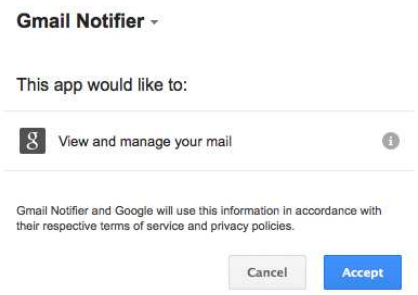

Figure 2. Un exemple de fenêtre proposée lors de l'installation d'une application tierce sur Gmail

L'efficacité de ces protocoles repose cependant sur la façon dont ils sont implémentés (Sun *et al.* 2011). En effet, certaines implémentations permettent d'usurper l'identité d'une application ou de forger des droits d'accès frauduleux. La phase de recueil du consentement est elle aussi cruciale – elle permet à l'utilisateur de savoir avec qui il partage ses données et à quelles conditions. Néanmoins, cette phase est implémentée de manière extrêmement variable. L'utilisateur ne dispose ainsi pas nécessairement de toutes les informations requises pour déterminer à quelles informations une application tierce souhaite accéder ni sa politique de conservation, de vente ou de traitement des données recueillies. Le Tableau 1 compare les informations données à l'utilisateur lors de l'installation d'applications tierces dans cinq applications web parmi les plus utilisées.

---

[5] voir http://pipes.yahoo.com/pipes, http://oauth.net/2 et http://gmail.com

Toutes les applications utilisent le même protocole d'autorisation – oAuth version 2.0. Néanmoins, oAuth 2.0 n'est qu'une spécification, chaque application l'implémente d'une manière propre. Toutes les applications permettent aux utilisateurs de les connecter à des applications tierces à l'installation desquelles l'utilisateur doit consentir, le plus souvent à partir d'une fenêtre dédiée. Aucune application ne permet aux utilisateurs de n'accepter qu'une partie des permissions demandées par l'application ou de restreindre les ressources auxquelles l'application souhaite accéder. La description de ces informations est importante pour permettre

Application	**Deezer**	**Facebook**	**Google**	**Twitter**	**GitHub**
Authentification	Oauth 2.0				
Documentation	http://tinyurl.com/888a82e	http://tinyurl.com/phcf7kg	http://tinyurl.com/phcf7kg	http://tinyurl.com/888a82e	http://tinyurl.com/nbp3zyr
Consentement de l'utilisateur nécessaire à l'installation	Oui				
Identité de l'application tierce	Nom, logo				
Affichage de la description de l'application tierce	Non				oui (au choix du développeur)
Contact du fournisseur	Non		email et site Web	Site web	
Actions possibles	Accepter / Refuser l'installation				
Énoncé des permissions	Enoncé des objets pouvant être lus.		Énoncé de ce que l'application peut faire et ne pas faire.		Énoncé de ce que l'application veut faire.
Niveau de détail dans la définition	Faible Utilisation de termes vagues, pas de description précise des ressources. Pas de précision de ce que l'app peut faire.		Fine Précision des objets auxquels l'application tierce veut accéder et des opérations qu'elle veut réaliser.	Moyenne Précision des objets mais pas des opérations.	Faible Pas de description des objets auxquels l'application veut accéder.
Autres informations	Non	Politique de vie privée	Non		

Tableau 1. Prise en compte de la vie privée dans cinq applications grand public

aux utilisateurs de choisir de connecter les applications. Le plus souvent, elle est vague. Par exemple, l'utilisateur est informé qu'une application veut accéder à son « profil public » sans se voir expliquer ce que ce profil contient. Enfin, alors que toutes ces applications sont « sociales » dans la mesure où elles reposent sur les interactions entre leurs utilisateurs, aucune ne propose lors de l'installation une évaluation de l'application issue de la communauté des personnes qui l'utilisent déjà, ce qui est connu pour augmenter la confiance des utilisateurs[6].

3.3 Ingénierie dirigé par les modèles, sécurité et abstraction dans les processus

Les applications web manipulent des types de ressources identiques. Par exemple, les réseaux sociaux et les systèmes de courrier électronique manipulent tous des messages électroniques. Cependant, chaque application maintient une représentation propre de ces ressources et repose sur une implémentation spécifique de l'authentification et du contrôle d'accès. La liaison entre les différentes applications se fait ainsi le plus souvent de manière *ad hoc* : les développeurs créent une application tierce pour une application source donnée et doivent réaliser le code nécessaire à chaque application source qu'ils souhaitent utiliser.

Dans les SOA, la composition abstraite de services, c'est-à-dire en se concentrant sur les fonctionnalités recherchées et non pas sur leur implémentation, peut être réalisée à partir d'une approche dirigée par les modèles (Chollet *et al.* 2008). Un métamodèle fournit le vocabulaire nécessaire à la capture d'un processus de manière abstraite. Des transformations de modèles permettent de générer le code nécessaire à la composition des services. L'ingénierie dirigée par les modèles permet aussi de définir à un niveau abstrait des propriétés de sécurité et une politique de contrôle d'accès et de générer à l'exécution le code nécessaire à la mise en œuvre de ces propriétés (Chollet *et al.* 2008) (Faravelon *et al.* 2012) (Wolter *et al.* 2007).

Néanmoins, toutes ces approches sont dirigées vers les concepteurs d'applications informatiques et non les utilisateurs finaux. (Cortes-Cornax 2011) montre ainsi que les langages d'expression de processus sont trop complexes pour être utilisés par des utilisateurs peu experts. L'auteur montre aussi que la distinction de plusieurs niveaux d'abstraction dans un métamodèle permet d'offrir un vocabulaire adapté à l'expertise de chaque utilisateur. Enfin, l'auteur montre que la syntaxe concrète d'un métamodèle est un élément crucial dans son utilisation. Dans le cas du *end-user programming* (Jones 1995), il faut ainsi proposer aux utilisateurs finaux une syntaxe graphique simple qui illustre un nombre de concepts restreint.

3.4 Conclusions à l'état de l'art et difficultés à résoudre

Notre état de l'art permet de dresser les conclusions suivantes : (1) la protection des données est indispensable pour l'acceptabilité de leur partage, mais sa mise en

---

[6] F-Secure. “Digital lifestyle survey”, 2013. http://www.f-secure.com

œuvre n'est pas assurée aujourd'hui de façon adéquate; (2) la composition de services REST est réalisée de manière *ad hoc* car elle repose sur des langages propres. Elle est ainsi inaccessible aux utilisateurs finaux ; (3) la composition de services REST repose sur le partage de ressources potentiellement sensibles. Néanmoins, les possibilités de contrôle d'accès des utilisateurs finaux à leurs ressources sont limitées.

Ces difficultés empêchent les utilisateurs finaux de considérer le web comme un véritable « système d'information distribué » (Fielding *et al.*, 2002) et de manipuler leurs ressources à leur guise. L'ingénierie dirigée par les modèles est un moyen efficace de composition de services et de configuration des préférences de partage mais elle n'a pas été appliquée au cas des services REST. C'est ce que nous cherchons désormais à faire.

4. Approche globale

Notre objectif est de permettre aux utilisateurs finaux des applications web de manipuler à leur guise leurs ressources. Ceci signifie permettre aux applications de partager les ressources qu'elles manipulent. Dans la mesure où ces ressources sont potentiellement sensibles, il est nécessaire de permettre aux utilisateurs finaux d'en contrôler finement l'accès.

Pour y parvenir, et afin de répondre aux difficultés identifiées dans l'état de l'art, nous proposons une approche dirigée par les modèles qui vise à permettre aux utilisateurs finaux de partager leurs ressources entre les différentes applications qu'ils utilisent sans être contraints à réaliser du code. Pour ce faire, il est nécessaire de disposer d'une description des applications à composer et de leurs besoins d'accès à un ensemble de ressources. Cette description est rédigée par les développeurs des applications. Il faut ainsi réconcilier le point de vue des utilisateurs finaux et celui des développeurs.

Afin d'atteindre ces buts, nous proposons une approche générative qui permet aux utilisateurs finaux de spécifier la composition d'un ensemble d'applications et les droits d'accès qui s'appliquent au partage de leurs ressources. La structure de notre approche est présentée sur la Figure 3. Nous fournissons un métamodèle qui donne le vocabulaire nécessaire pour exprimer de manière abstraite les ressources que l'utilisateur souhaite manipuler, les actions qu'il souhaite réaliser sur ces dernières et les droits d'accès aux ressources. Ce métamodèle est séparé en deux vues, une vue ressources et une vue autorisation.

Dans la mesure où ce métamodèle doit pouvoir être instancié par des utilisateurs finaux et des fournisseurs d'applications, nous avons défini deux niveaux d'abstraction. Le niveau d'abstraction le plus élevé est destiné aux utilisateurs finaux. Il laisse de côté les détails techniques. Le niveau d'abstraction le moins élevé est, lui, destiné aux fournisseurs d'applications. Il leur permet de décrire de manière détaillée leurs applications.

Enfin, nous avons souligné que les utilisateurs finaux étaient habitués à manipuler leurs applications au travers d'environnements graphiques orientés ressources. Afin de favoriser l'acceptabilité de notre proposition, nous définissons une syntaxe concrète graphique de notre métamodèle. Cette syntaxe concrète est inspirée des applications web afin d'être cohérente avec l'expérience habituelle des utilisateurs finaux.

Le code nécessaire à la communication entre les différentes applications et au contrôle de leur accès aux ressources est généré à partir des modèles établis par les utilisateurs finaux. L'utilisateur final n'a ainsi pas à rédiger de code informatique. Nous présentons dans les sections suivantes notre métamodèle ainsi que l'implémentation à laquelle notre travail a donné lieu.

5. Approche détaillée

Nous présentons dans cette section les détails de notre niveau spécification et de notre niveau configuration/implémentation.

5.1 Niveau d'abstraction du métamodèle pour les utilisateurs finaux

Les utilisateurs finaux n'ont pas de compétences techniques. Nous proposons un métamodèle construit à partir du vocabulaire qu'ils manipulent habituellement.

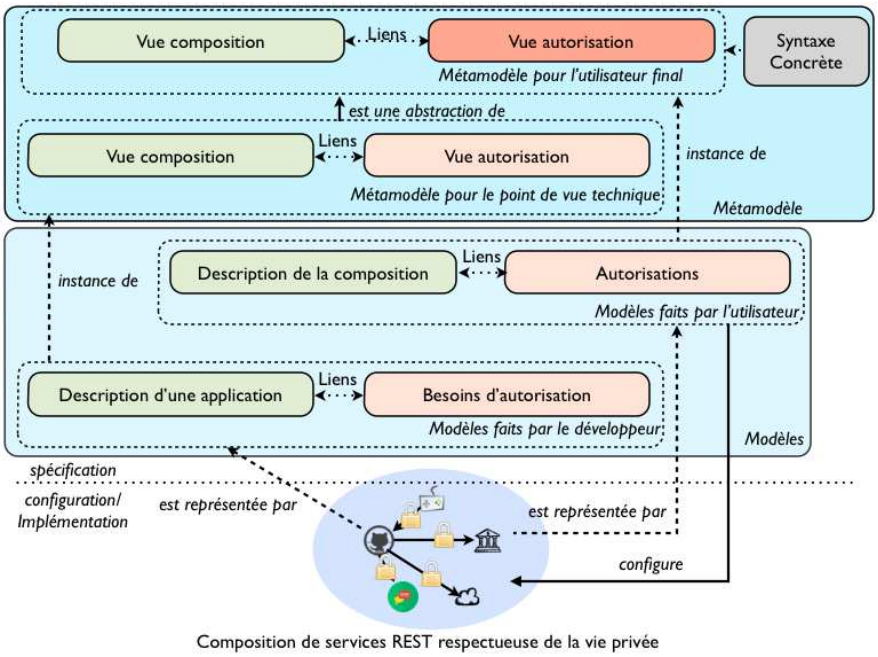

Figure 3. Approche globale

présenté sur la Figure 4.

5.1.1 Entités du métamodèle

Le métamodèle destiné aux utilisateurs, présenté sur la Figure 4, repose sur les entités suivantes : une **ressource** est un objet que l'utilisateur peut créer ou modifier. Une photographie ou un message électronique, par exemple, sont des ressources. Chaque ressource a un type. Une **application** est un programme informatique pré-existant qui permet à un utilisateur de créer ou modifier des ressources. Un site de réseau social ou un système de messagerie sont des exemples d'applications. Pour une ressource donnée, l'application qui permet de la créer est une application source. Les applications qui souhaitent accéder à cette ressource sont des applications tierces. Une **action** est un modification apportée à une ressource par une application. Des actions peuvent dépendre les unes des autres. Par exemple, l'impression d'un album photo nécessite de payer l'impression. Une application possède une politique de vie privée qui spécifie les données auxquelles elle accède et la manière dont elle les traite. Concrètement, cet attribut contient l'URL de la politique de gestion de la vie privée de l'application, ce qui permet de la télécharger dynamiquement. Une application est offerte par un **fournisseur** identifié par son nom et qui peut être contacté. Un utilisateur se connecte à une application grâce à un **compte**. Un compte est défini par un identifiant et un mot de passe. Une application peut gérer l'identification des utilisateurs pour d'autres applications.

Enfin, il est nécessaire de contrôler l'accès d'une application tierce aux ressources de l'utilisateur. Le contrôle s'effectue par la définition de **permissions** qui possèdent une durée de vie déterminée. Une permission est aussi définie par un ensemble de contraintes sur les **propriétés** des ressources considérées. Par exemple, il est possible de préciser qu'une application ne peut accéder qu'aux photos d'un album donné.

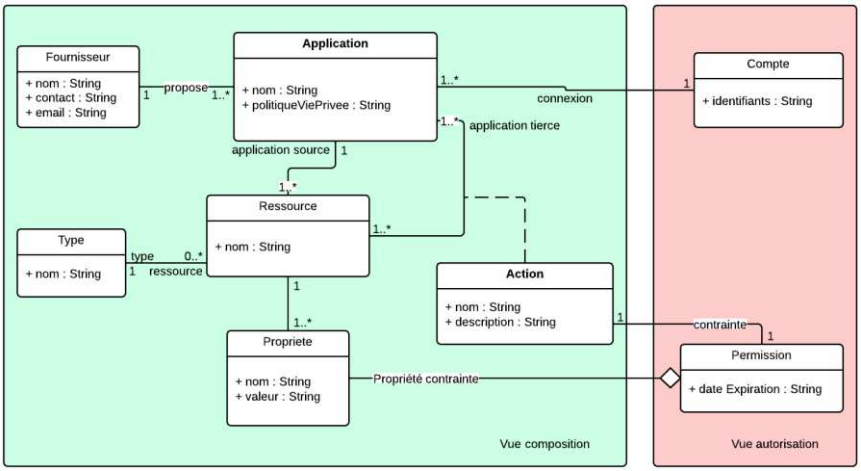

Figure 4. Métamodèle destiné aux utilisateurs finaux

5.1.2 Syntaxe concrète du métamodèle

Les utilisateurs finaux ont l'habitude d'interagir avec les applications web au travers d'environnements graphiques. Pour cette raison, il est nécessaire de définir une syntaxe concrète à notre métamodèle. Afin de favoriser l'acceptabilité de notre proposition, nous définissons cette syntaxe concrète à partir d'éléments graphiques déjà existants. Le Tableau 2 présente la syntaxe concrète que nous proposons.

5.2 Métamodèle pour les développeurs

Afin de décrire leurs applications, les développeurs ont besoin d'un vocabulaire technique. Nous reprenons ce vocabulaire dans un métamodèle présenté sur la Figure 5.

Ce métamodèle est construit en étendant le métamodèle des utilisateurs finaux avec les concepts destinés aux développeurs, qui sont présentés en gras sur la figure 5. Ces concepts sont issus des notions de l'architecture REST que l'on trouve notamment dans (Fielding *et al.* 2002). Chaque ressource possède une adresse ou URI, qui joue le rôle d'identifiant. Chaque ressource possède des **représentations**, des instances de la ressource. Par exemple, le concept d'image peut avoir plusieurs instances pour des images réelles différentes. Une représentation est décrite par un ensemble de **métadonnées**. Les types des ressources sont associés à des types MIME, ce qui permet d'identifier les équivalences entre les différents types (ex. : image et photo sont tous les deux des types MIME image). Les ressources sont regroupées dans une **API**, qui implémente une Application. Les applications pouvant avoir des niveaux de détails différents pour décrire les ressources, des modèles pivots et des adaptateurs peuvent être implémentés pour unifier ces

Entité du métamodèle	Syntaxe concrète	Exemple
Application	Logo de l'application	
Fournisseur	Texte ou icône	Crédit Agricole
Compte	Icône	
Ressource	Texte ou icône	
TypeRessource	Texte ou icône	Photos
Action	Texte ou icône	Impression
Permission	Texte	Lecture une fois

Tableau 2 - Syntaxe concrète du métamodèle pour les utilisateurs finaux

descriptions. Une action est considérée comme une action HTTP, et donc spécialisée en **Get**, **Post** ou **Delete**. Le compte d'un utilisateur peut être associé à une **autorité centrale** dans le cas où il peut être utilisé par plusieurs applications. Chaque application peut être associée à une ou plusieurs **méthodes d'authentification**. Par exemple, une application peut à la fois permettre de s'authentifier en utilisant oAuth et un login et un mot de passe.

6. Implémentation

Afin de démontrer la faisabilité de notre approche, nous avons développé un prototype de gestion de ressources distribuées entre les applications d'un utilisateur. Cet environnement permet aux utilisateurs de configurer le partage de leurs ressources et les droits d'accès à ces dernières grâce à une page web. La Figure 7 présente une capture d'écran de notre outil.

Le menu de gauche ❶ reprend la liste des ressources qu'un utilisateur peut manipuler. Cette liste est générée à partir du métamodèle pour les utilisateurs finaux. À chaque ressource est associé un menu contextuel qui permet de sélectionner sa source – par exemple un réseau social. Les ressources sélectionnées par l'utilisateur ❷ ❸ apparaissent dans l'espace central de la page web. L'utilisateur peut alors filtrer un sous-ensemble des ressources, par exemple un album auquel des photos doivent appartenir. Les ressources sélectionnées sont associées à un menu contextuel qui

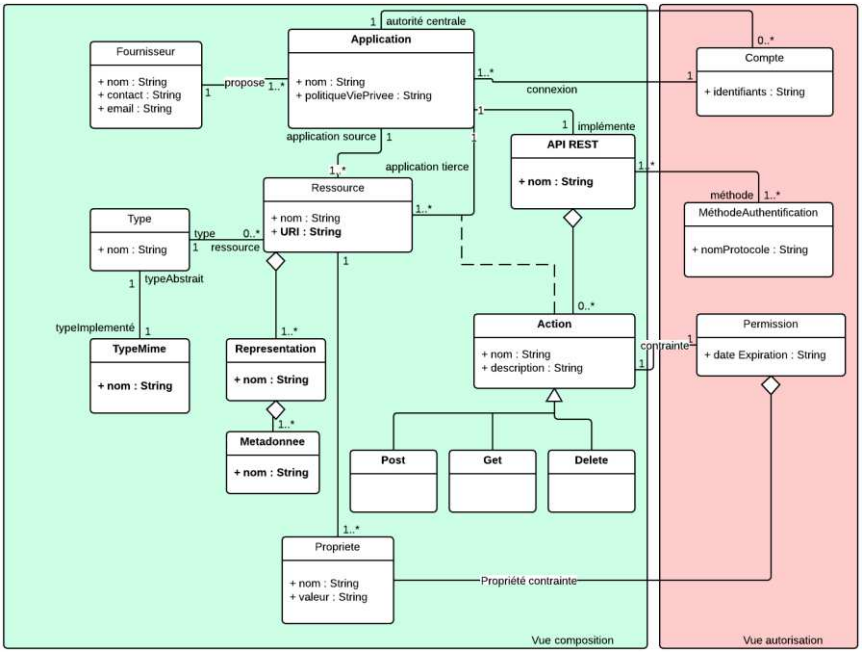

Figure 5. Métamodèle pour les développeurs

Figure 7. Capture d'écran d'une fenêtre de configuration des droits d'accès

présente les actions possibles sur un ensemble de ressources. L'enchaînement des actions permet implicitement de réaliser un processus, comme celui de notre cas d'étude.

Lorsqu'un utilisateur sélectionne une action, une boîte de dialogue ❹ de demande de confirmation de l'utilisation d'une application tierce apparaît. La boîte de dialogue présente toutes les informations spécifiant les autorisations et leurs limites ❺, ainsi que le contact chez le fournisseur de l'application tierce ❻ et les informations sur les contacts de l'utilisateur qui utilisent déjà l'application ❼. Cette boîte de dialogue est générée à partir de la description des applications fournies par les développeurs.

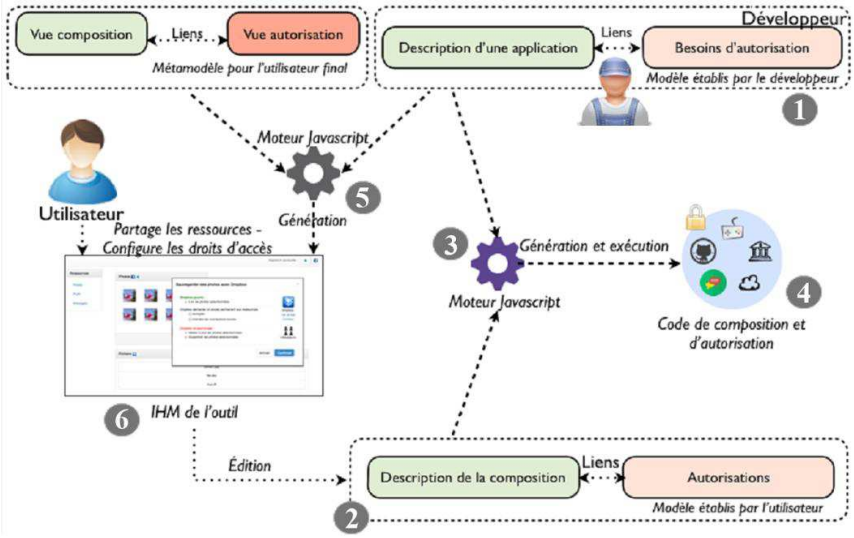

Figure 6. Principe de notre outil

La spécification des actions à effectuer sur un ensemble de ressources et des droits d'accès permet de générer le code de composition des applications et le code d'autorisation. Afin de mettre en œuvre les droits d'accès, notre prototype joue le rôle de médiateur entre les applications sources et les applications tierces : pour chaque ressource, il génère une URL vers laquelle l'application tierce pointera. Cette URL décrit les droits d'accès de l'application tierce. Avant de donner l'accès à l'application source, le prototype évalue les droits d'accès. La Figure 6 reprend le principe de notre prototype et de l'utilisation de la génération automatique. A partir des modèles des applications et des besoins d'autorisations décrits par les développeurs, un premier moteur Javascript génère l'IHM présentée à l'utilisateur final (figure 6), sur laquelle ce dernier peut définir les autorisations pour la composition concrète qu'il est en train de spécifier. A partir de cette configuration définie par l'utilisateur final, un second moteur Javascript génère le code de composition des applications et de contrôle d'accès. C'est à ce niveau qu'intervient l'originalité de notre proposition, puisque ce code est actuellement créé par les développeurs pour une composition donnée, et non généré dynamiquement.

7. Conclusion et perspectives

Les débats actuels sur la protection des données personnelles montre l'importance de la question de la vie privée. Nous avons cherché à permettre aux utilisateurs finaux des applications web de composer leur applications tout en protégeant leur vie privée. Nous entendons ici la protection de la vie privée comme le contrôle des applications tierces qui peuvent accéder aux informations d'un utilisateur et leurs droits d'accès à ces informations.

Pour y parvenir, nous avons proposé une approche dirigée par les modèles afin de configurer les applications web qu'un utilisateur souhaite composer et leurs droits d'accès à un ensemble de ressources. Cette approche permet à un utilisateur final de décrire le partage de ses ressources entre ses applications et leurs droits d'accès. Des transformations automatiques permettent de générer le code nécessaire à l'exécution de la composition et à la protection de sa vie privée.

Notre prototype se positionne comme une recommandation, dans la mesure où, pour être efficace, il faudrait qu'il soit intégré directement dans les applications Web. Dans les travaux, il conviendrait d'évaluer l'acceptabilité de cette proposition par les fournisseurs d'applications et par les utilisateurs finaux.

Notre approche est implémentée sous la forme d'un environnement de gestion des ressources d'un utilisateur. Cet environnement démontre qu'il est possible d'améliorer la prise en compte de la vie privée dans les applications web. Son fonctionnement pourrait être intégré aux plateformes existantes qui proposent déjà de partager leurs ressources avec des applications tierces. Nous prévoyons, à titre de travaux futurs, d'étendre notre approche en prenant en compte des ressources complexes, comme une structure composée d'une photo, de son auteur et de sa localisation. Nous augmenterons aussi la gestion de la cohérence des permissions de

données en définissant une sémantique formelle. Enfin, nous prévoyons d'évaluer l'utilisation de cet outil par des utilisateurs finaux.

Remerciements

Nous remercions Stéphane Frénot pour ses relectures et ses conseils.

Références

Budan Wu, Rongheng Lin, Junliang Chen (2013). Integrating RESTful Service into BPEL Business Process on Service Generation System. Proceedings of the *IEEE Service Computing Conference 2013, 2013, IEEE Computer Society.*

Cortes Cornax M. (2011). Service choreographies through a graphical notation based on abstraction layers and viewpoints. *IEEE RCIS 2011. 2011.*

Chollet, S., Lalanda, P. (2008). Security Specification at Process Level. *IEEE Service Computing Conference 2008, 2008, IEEE Computer Society.*

Dami, S., Estublier, J., Amiour, M. (1998). Apel: A Graphical Yet Executable Formalism for Process Modeling Autom. *Softw. Eng.*, vol. 5, p. 61-96.

Faravelon, A., Chollet, S., Verdier, C., Front, A. (2012). Configuring Private Data Management as Access Restrictions: From Design to Enforcement. *International Conference on Service Oriented Computing*, 2012.

Jones C. (1995). End user programming. *Computer*, vol. 28, n°.9, p.68-70

Meng-Yen Hsieh, Hua-Yi Lin, Kuan-Ching Li. (2011). 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.

Papazoglou, M. P. (2003). Service-Oriented Computing: Concepts, Characteristics and Directions. *Proceedings of the Fourth International Conference on Web Information Systems Engineering, 2003,* IEEE Computer Society.

Pautasso P. (2009). RESTful Web service composition with BPEL for REST. *Data Knowl. Eng.*, vol. 68, n°9, p. 851-866.

Rosenberg F., Curbera F., Duftler M. J., Khalaf R. (2008). Composing RESTful Services and Collaborative Workflows: A Lightweight Approach. *IEEE Internet Computing*, vol. 12, n°5, p. :24-31.

Roy T. Fielding, Richard N. Taylor (2002). Principled design of the modern Web architecture. *ACM Trans. Internet Techn.*, vol. 2, n°2, p. 115-150.

Sun S., Beznosov K. (2012). The devil is in the (implementation) details: an empirical analysis of OAuth SSO systems. *ACM Conference on Computer and Communications Security, 2012*. ACM.

Wolter, C., Schaad, A. (2007). Modeling of task-based authorization constraints in BPMN *BPM'07*, 2007, Springer-Verlag