École Polytechnique de l'Université de Tours
64, Avenue Jean Portalis
37200 TOURS, FRANCE
Tél. +33 (0)2 47 36 14 14
www.polytech.univ-tours.fr

Département Informatique
5e année
2013 - 2014

Rapport de projet de fin d'études

Interface d'annotations de page web pour le projet SWAP

Encadrant
Sébastien AUPETIT
sebastien.aupetit@univ-tours.fr

Étudiants
Vincent ROUILLÉ
vincent.rouille@etu.univ-tours.fr

Université François-Rabelais, Tours

DI5 2013 - 2014

Version du 5 mai 2014

Table des matières

Table des figures

Introduction

L'objectif de ce PFE consiste à réaliser sous forme modulaire et générique des outils d'annotations de page web. Une annotation étant des métadonnées associées à une partie de la page web. Par exemple, une page web pourrait contenir une annotation marquant l'en-tête de la page, son menu, où se situe le contenu, etc.

L'originalité de ce projet vient de l'approche par laquelle les outils d'annotations sont mis en place. Les méthodes classiques : comme créer une interface à l'aide d'un moteur de rendu web, de modifier un navigateur existant ou autre, ont tous des difficultés à être maintenu et à gérer la portabilité et l'intégration dans des logiciels déjà existants. L'approche utilisée pour ce projet consiste en la modification des pages web avant même qu'elles n'atteignent le navigateur et a injecter les outils d'annotations. Une telle méthode garantie une portabilité totale avec l'ensemble des logiciels permettant de naviguer sur internet.

Ce projet sera réalisé sous la forme d'une extension au projet SWAP. Le projet SWAP (Smart Web Accessibility Platform) développé par l'équipe HANT de Polytech'Tours a pour objectif la mise en place d'un système transformant intelligemment les pages web pour les rendre plus accessible et ce de façon automatique ou quasi automatique. Ce système se place entre le navigateur de l'utilisateur et les sites web qu'il visite. SWAP se charge de modifier le contenu des pages afin d'améliorer l'efficacité des outils déjà existant et ainsi aider l'utilisateur finale à mieux appréhender le contenu de la page.

Un des problèmes du projet SWAP est l'évaluation, c'est à dire la détermination d'informations complémentaires à une page web permettant de mieux en comprendre le sens, comme des métadonnées permettant de définir le rôle de certaines parties. Du fait de la variabilité du web, les éléments d'une page web ne peuvent être identifié de façon fiable sans intervention humaine. Il faut donc un utilisateur qui aura pour rôle de définir des informations sur une page. Par exemple, un article, un menu, etc.

Ces informations permettront alors d'améliorer les outils du projet SWAP, soit via l'utilisation des données directement pour aider l'utilisateur, soit par l'utilisation de ces données pour vérifier le comportement d'algorithme sur un grand nombre de sites.

Le projet SWAP

Ce PFE étant intimement liée au projet SWAP, une explication des technologies et méthodes qu'il utilise est nécessaire pour bien saisir l'essence du projet. SWAP est une application réalisé en Java reposant très fortement sur le framework Spring [1]. Il reprend donc les principaux concepts de Spring tel que la modularité ou l'inversion de contrôle.

Le projet est donc découpé en module, chaque module ayant un but précis et peut s'interconnecter avec les autres. Ces modules reposent tous sur la même base. On peut regrouper ces modules de façons à distinguer 3 parties :

- Coeur : Ce sont les éléments communs utilisés par les deux parties suivantes,
- Proxy : C'est la partie gérant la transformation des pages avant envoi à l'utilisateur
- Server : C'est la partie gérant le stockage et la diffusion de données à répartir à l'ensemble des utilisateurs.

De la même façon, le système d'annotation apporte trois nouveaux modules :

- expertinterface : pour la partie commune,
- expertinterfaceproxy : pour l'intégration de l'interface d'annotation,
- expertinterfaceserver : pour la gestion de la persistance des données annotées,

L'outil Apache Maven [2] est utilisé pour la gestion des dépendances entre les modules. Il permet de déclarer simplement les dépendances requises et se charge de faire le nécessaire pour les rendre fonctionnel au sein de l'application. Maven n'est pas juste un gestionnaire de paquets, il permet bien plus. Outre l'aspect modulaire, différents concepts de Spring sont repris dans SWAP.

2.1 Spring

Spring est un framework Java [3] sous licence Apache. Il a pour objectifs :

- Gérer la partie infrastructure afin de permettre au développeur de ne se préoccuper que de la partie applicative.
- D'apporter une modularité complète ainsi qu'une intégration non invasive. Par exemple pour utiliser les fonctionnalités Web de Spring, il faut utiliser le projet Spring Web.

Spring est donc un projet fortement modulaire, chaque module apportant une fonctionnalité particulière. Le projet SWAP utilise les modules suivants :

- Spring Framework : Coeur de spring apportant notamment l'inversion de contrôle et les appels de fonctions à distance,
- Spring Data JPA : Gestion simple de la persistance des données,
- Spring Web Services : Mis en place d'applications web.

1. Spring framework : http://www.springsource.org/
2. Apache Maven : http://maven.apache.org/
3. Java : http://www.java.com/

FIGURE 2.1 – Logo de SWAP

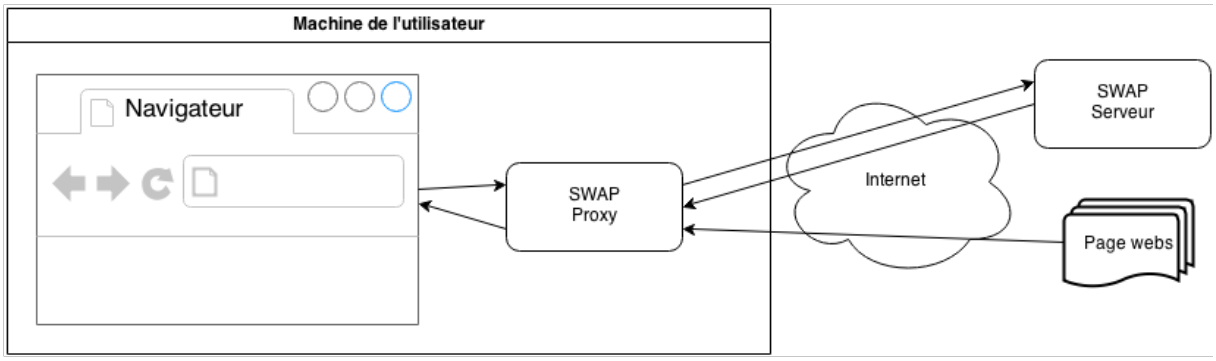

FIGURE 2.2 – Positionnement de SWAP et interactions

FIGURE 2.3 – Logo de spring

2.1.1 Inversion de contrôle

Le concept le plus important de Spring est l'inversion de contrôle. Celui-ci permet de palier aux problèmes posés par la modularité d'un projet en permettant d'accéder facilement à certains objets qui serait difficilement accessible avec les approches classiques.

Prenons un exemple simple. Une classe A contient un attribut de classe B. Une solution classique est d'instancier B lors de l'instanciation de A, ce qui est en général fait dans la classe elle-même.

L'inconvénient est que B est figé. Par exemple, si B est différent entre les tests, l'environnement de production et l'environnement de développement, la classe A devra prendre en considération ces cas. Sur un projet important, cela rends vite le code difficile à lire et à maintenir. L'approche classique rends aussi le partage de la ressource B plus difficile.

L'approche inversion de contrôle consiste à prendre le prendre dans l'autre sens. La classe A a besoin d'un attribut de classe B et c'est un moteur tel que Spring qui fera en sorte de trouver un élément qui correspond aux besoins de A et qui lui attribura. Cela permet d'accèder très facilement à différents objets et simplifie très fortement le code ainsi que la définition des classes.

Cela ne se fait pas toutefois par magiquement, il faut définir ce besoin soit via des annotations, soit via la déclaration dans un fichier xml de "beans". Un bean correspond à un objet (en général un singleton [4], mais Spring permet bien plus) utilisable pour les objets qui pourraient en avoir besoin.

Ce système apporte une grande simplicité à un projet aussi complexe que SWAP. Il facilite le codage et la maintenance.

2.1.2 Les annotations

Spring et le projet SWAP utilisent intensément les annotations Java pour l'intégration de fonctionnalités. Par exemple, pour donner la capacité à une classe de répondre à des requêtes web, l'utilisation de quelques annotations permets d'arriver à un tel résultat avec un minimum d'effort.

Les annotations permettent donc d'accroitre la productivité, mais nécessite toutefois un temps de recherche dans la documentation assez conséquent. Une fois les annotations les plus courantes bien connue,

4. Un singleton est une classe qui n'est présente que sous la forme d'une unique instance pour l'ensemble de l'application

```java
@Handler(singleton = true)
public class HtmlTransformationLoaderInjection extends HtmlDomTransformation {

  /** The environment. */
  @Autowired
  private Environment environment;

  @Autowired
  private ElementFactory elementFactory;
```

FIGURE 2.4 – Exemple d'annotation et d'inversion de contrôle

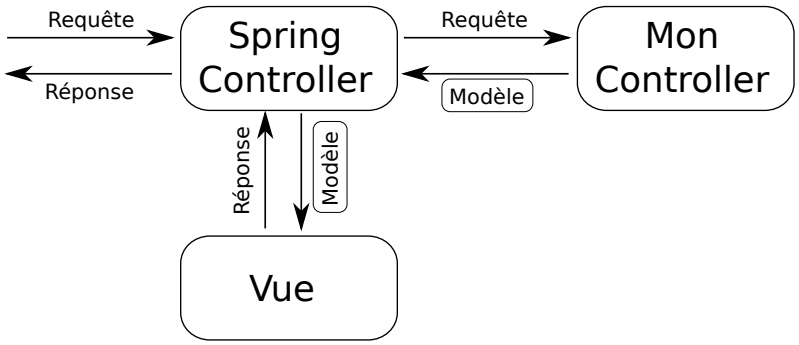

FIGURE 2.5 – Schéma de gestion d'une requête Web par Spring Web MVC

la réalisation de fonctionnalités, même complexes, est souvent un vrai jeu d'enfant.

La capture 2.4 est exemple d'utilisation des annotations venant de la classe injectant l'interface dans une page web. L'annotation `@Handler` permet de définir la classe comme un modificateur de page web. Quelque part dans le projet SWAP, cette classe sera donc utilisée pour une tâche particulière sans que le créateur de cette classe n'ait à faire quoique ce soit d'autre que d'ajouter cette annotation. Cela permet une très grande flexibilité d'utilisation et une modularité exemplaire. La seconde annotation visible est très courante dans un projet utilisant le framework Spring, *@Autowired* permet de récupérer automatique une instance de l'élément demandés. Ici, ce sera l'environnement global de l'application et une classe permettant de créer des éléments pour le DOM.

2.1.3 Web MVC

Le module Web Service [5] permet la réalisation simplifiée d'application web avec une approche MVC [6]. Il s'intègre au fonctionnement des servlets [7] Java.

Le servlet est géré par le framework Spring Web. En utilisant les principes d'inversion de contrôle, il est très simple, via l'annotation *@Controller* de créer un contrôleur [8] pour une page Web. Il en est de même pour le routage [9] des adresses qui se fait via une annotation sur les fonctions du contrôleur. La figure 2.5 permet de bien se rendre compte des étapes mise en oeuvre dans la création d'une réponse à une requête.

5. http://docs.spring.io/spring/docs/3.2.x/spring-framework-reference/html/mvc.html
6. Modèle / Vue / Controlleur
7. Un servlet est une classe Java servant à étendre les fonctionnalités d'un serveur HTTP. La définition d'un servlet est standardisée.
8. Un contrôleur, dans le modèle Web MVC, a pour but de mettre à jour le modèle de données d'après la requête.
9. Le routage permet de déterminer quel est le contrôleur qui doit se charger d'une requête à partir de paramètres tels que la forme de l'url.

2.1.4 Persistance des données

Spring Data est un module Spring permettant de faire abstraction des différentes API permet de gérer des bases de données (ex : JDBC, Hibernate, JPA, etc.) tout en fournissant des services et méthodes permettant de gérer plus facilement les transactions.

Comme pour Spring Web, l'ajout d'annotation permet de totalement changer le comportement d'une classe et ainsi de la sauvegarder simplement au sein d'une base de données. Les annotations permettent de définir le modèle de la base de données. Il est très rare de devoir créer des requêtes SQL manuellement, celle-ci étant en général automatiquement généré par le système d'annotation.

2.1.5 Appels à distance

Spring inclut un système d'appel de fonctions à distance. Celui-ci peut-être couplé à différentes méthodes de transports des informations. Dans le cas du projet SWAP, les appels distants se font sous forme de requêtes webs : *Spring's HTTP invoker*[10].

10. http://docs.spring.io/spring/docs/3.2.x/spring-framework-reference/html/remoting.html

Recherches et prototypage

Le projet SWAP apporte une base importante de fonctionnalités. Modifier une page web pour y inclure une nouvelle interface peut alors sembler simple. Dans la pratique, il faut gérer les risques de conflits et rendre l'interface inaccessible à la page web.

La première étape dans la réalisation de ce PFE fut donc d'effectuer les recherches nécessaires et de créer plusieurs prototypes permettant de choisir la meilleure méthode actuellement utilisable pour réaliser l'inclusion de l'interface d'annotation. Les recherches ont donc portés sur :

- L'isolation du JavaScript afin de le rendre inaccessible au reste de la page.
- L'isolation des éléments insérés afin de ne pas perturber le rendu de la page web.
- L'isolation du CSS pour permettre de gérer l'apparence de l'interface simplement.

3.1 Isolation

Pour comprendre les différents moyens d'isolation, il est nécessaire d'avoir une compréhension du DOM et de sa structure. Le DOM est un arbre d'objets créés pour chaque page web qui représente la structure de la page. Le DOM part d'un noeud racine qui représente la fenêtre du navigateur et s'étend jusqu'aux éléments HTML feuilles de la page web. Tout élément visible d'une page web est présent dans le DOM. Les différents scripts des pages web utilisent le DOM pour toutes les opérations qui visent à modifier l'affichage. Le DOM permet de structurer une page web et d'interagir avec les éléments de celle-ci.

Comme il n'existe pas de gestion de droits d'accès sur les éléments du DOM, la page est modifiable à la fois par le script injecté et par les scripts de la page. L'ajout d'éléments dans le DOM peut donc avoir des conséquences importantes sur le fonctionnement et l'aspect de la page.

Pour injecter l'outil d'annotation, il est nécessaire d'isoler au maximum les interactions de la page web avec les éléments de l'interface d'annotations au niveau du DOM mais aussi au niveau des styles d'affichage.

Nous avons identifié trois façons principales d'isoler l'interface du DOM de la page web avec le mécanisme d'injection de contenus (cf. Fig. 3.1). Du résultat de cette analyse découle l'isolation des styles et du Javascript.

Toutes les méthodes présentées dans ce rapport ont été testé via un prototype.

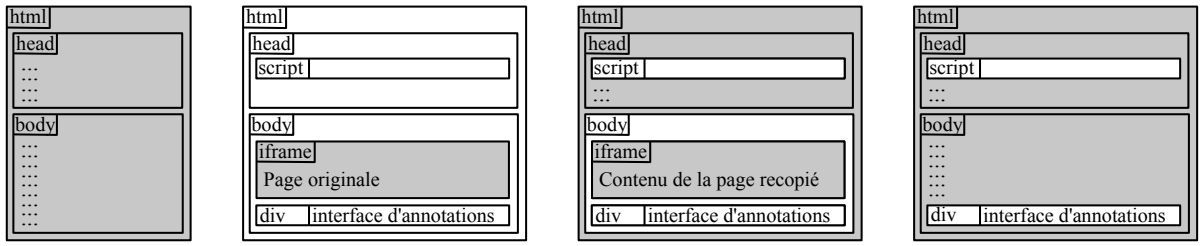

FIGURE 3.1 – Les trois manières d'isoler le DOM. La page original est grisé, les éléments injectés sont blancs. (a) DOM original, (b) Isolation par une `iframe` avec URL externe, (c) Isolation par une `iframe` avec DOM interne, (d) Isolation par ajout d'un noeud.

3.1.1 Isolation du DOM par une `iframe` avec URL externe

La balise `iframe` permet d'incorporer un document HTML (contenu) au sein d'un autre document HTML (contenant). Les interactions entre le contenant et le contenu sont fortement contraintes. Le contenant a son propre DOM indépendant et invisible du point de vue du DOM du contenu.

La première façon d'assurer l'isolation du DOM consiste en la création d'un document HTML contenant l'interface et une `iframe` contenant le document à annoter (cf. Fig. 3.1(b)).

Cette méthode garantie une isolation totale du document annoté par rapport à l'interface (DOM, styles et Javascript). Cependant, cette organisation a de lourdes conséquences. L'implémentation des interactions entre l'interface et la page web visité impose l'utilisation d'une API de communication JavaScript complexifiant le codage et la maintenance de l'outil. De plus, la gestion de l'`iframe` au sein de l'interface d'annotation entre plus ou moins en conflit avec des fonctionnalités du navigateur telles que l'ouverture dans une autre fenêtre/onglet ou la gestion des popups. Les interactions entre l'`iframe` et le navigateur peuvent provoquer un changement de la page web courante et ainsi faire disparaître l'interface d'annotations (popup sans interface d'annotations...). Une solution possible à la disparition de l'interface serait de détecter via la page web la présence ou non de l'interface d'annotation. En cas d'absence, on recharge la page avec l'interface d'annotations via une URL particulière spécifiant l'URL de la page à annoter à inclure dans l'`iframe`. Le problème est que rien ne garantit qu'un second affichage de la même page web donne le même résultat notamment, mais pas uniquement, dans le cas de requêtes POST. De façon résiduelle, l'URL affiché par le navigateur correspond au contenant (l'interface d'annotations) plutôt qu'à celle de la page web visité.

Une telle méthode reviendrait donc plus ou moins à intégrer un navigateur dans un navigateur : une tâche longue, complexe et qui ne correspond pas aux besoins.

3.1.2 Isolation du DOM par une `iframe` avec DOM interne

Une solution alternative à l'isolation avec une `iframe` insensible au rechargement consiste à ajouter à toutes pages web un code JavaScript se déclenchant après la création complète du DOM de la page. Comme on peut le voir sur la figure 3.1(c), ce script copie le DOM de la page web, supprime tous les éléments du DOM de la page, insère dans le DOM une `iframe` contenant le DOM copié (donc le DOM original).

Cette approche permet de conserver l'isolation totale du DOM du contenu, de ne pas altérer les habitudes de navigation et enfin de garantir l'affichage en toute circonstance de l'interface d'annotation. Cette solution a en théorie tous les avantages, sans les inconvénients précédents. Cependant, elle a néanmoins des conséquences importantes :

- Une latence importante est introduite car il est nécessaire d'attendre que le DOM soit complètement disponible.
- Le DOM étant déjà créé, il n'est pas possible de décharger les scripts existants de la page web. Ceux-ci provoquent souvent des échecs ou des boucles infinies pouvant aller jusqu'au plantage du navigateur.
- Le fait de créer une `iframe` et d'y insérer le contenu de la page induit une nouvelle analyse complète du contenu ce qui augmente à nouveau la latence ainsi que des problèmes de performances et de stabilité.

Cette approche semblant proche du réalisable, une modification de celle-ci a permis d'en améliorer le fonctionnement. Au lieu de charger deux fois le contenu de la page, le proxy se chargeant de l'injection de l'interface d'annotation envoi le contenu HTML de la page dans un espace non interprété par le navigateur. Le contenu de la page est alors récupéré et injecté dans une iframe. Cette petite modification a permis à un bon nombre de sites de fonctionner. Elle pose cependant beaucoup de problème d'interprétation, mettant toujours en échec les différents scripts présent dans certaines page web. Sans parler des sites intégrant eux-mêmes des iframes.

Cette solution n'est donc adaptée qu'à de très simples pages statiques, ce qui est de nos jours est de plus en plus rare. Elle fut donc rapidement écartée elle aussi.

3.1.3 Isolation du DOM par ajout d'un noeud

La dernière solution que nous avons considéré est également la plus simple. Elle consiste à ajouter un élément dans le DOM de la page web, comme le montre la figure 3.1(d). Cet élément sert de noeud racine pour tous les autres éléments de l'interface d'annotations. Les principaux avantages sont :

- une grande simplicité de mise en place
- une faible complexité de gestion des éléments
- un impact quasi nul sur les performances de navigation

L'inconvénient majeur est que le DOM de la page web est modifié. Dans nos expérimentations, aucun site web ne s'est montré incompatible avec cette méthode. Néanmoins, il est théoriquement possible qu'une page web cesse de fonctionner ou de s'afficher correctement du fait de cette modification. Pour limiter les risques, nous insérons par défaut l'élément à la fin du DOM. Insérer un élément à la fin du DOM permet d'éviter les conflits avec les différents scripts de parcours du DOM qui commence généralement par une lecture du DOM depuis le haut.

Au regard des différents avantages et inconvénients des différentes approches, nous avons considéré que cette méthode était la plus appropriée. Pour que celle-ci soit viable, il est alors nécessaire de réaliser une isolation des styles et des scripts de l'interface afin que ceux-ci ne soient pas influencés par ceux du document original.

3.2 Isolation des styles

La gestion des styles CSS est fortement liée à la forme du DOM. Afin que les éléments de l'interface insérés dans la page aient la même apparence sur toutes les pages web, les règles CSS à appliquer doivent être créées selon une méthode particulière.

L'approche la plus évidente consiste à marquer l'élément parent contenant l'interface par une classe CSS « unique ». A partir de cette classe, des règles ne s'appliquant qu'aux éléments de l'interface d'annotations peuvent être crées. Afin d'éviter tout problème de priorité entre les règles, toutes les propriétés doivent être marquées comme prioritaire. La définition de toutes ces règles devient très rapidement complexe. De plus, les propriétés CSS à définir sont nombreuses, varient en fonction des navigateurs et des évolutions des standards. Maintenir les styles devient alors rapidement un problème.

Une solution à ce problème consiste à appliquer les styles CSS aux éléments de l'interface via du code JavaScript. Pour cela, l'ensemble des propriétés CSS qui sont appliqués à un élément sont comparées avec un élément du même type non modifié. Toute propriété qui diffère est alors changée par la valeur définie par l'élément de référence. Pour obtenir cette élément de référence qui n'est autre qu'un simple élément d'une page web sans CSS, une iframe non visible est créée et un élément du même type est crée. C'est cet élément qui sert de référence. Cette approche induit un léger impact sur les performances de création de l'interface, mais le gain en termes de simplicité, de sécurité et d'évolutivité prévaut largement.

Une approche légérement plus évoluée est présentée dans la partie traitant de l'architecture finale.

3.3 Isolation du Javascript

Il reste à isoler le code JavaScript injectée de celui de la page. Le proxy de l'interface d'annotations est conçu pour insérer du JavaScript le plus haut possible dans la structure de la page. Le script ainsi inséré est donc systématiquement le premier chargé et exécuté par la page web visité. Cette approche permet d'éviter tout problème que pourrait poser un quelconque script sur la page web qui pourrait modifier le comportement de certaines API dans le contexte de la page web.

```
1 (function() {
2   // Code isolé
3 })();
```

FIGURE 3.2 – Structure du code permettant d'obtenir un nouvelle espace de nom

Le code chargé se doit de ne pas entrer en conflit avec celui de la page web. La solution est un système de module et une bonne méthodologie. Dans les différents modules on s'interdit de créer ou modifier des variables globales. Il est même possible de rendre le script injecté totalement invisible à la page. Cette technique est très simple et repose sur une façon de programmer très courante en JavaScript. Elle consiste en la création d'une fonction anonyme que l'on appelle directement au niveau de sa définition.

Cette méthode (cf. Fig. 3.2) permet d'éviter la création d'une quelconque référence dans l'espace de nom de la page et permet donc l'utilisation de variables locales. Il n'existe pas de moyen pour la page web d'accéder aux variables et fonctions qui pourrait être défini dans cette espace.

Le système de chargement de modules utilise cette méthode pour s'initialiser. Il met à disposition une interface permettant aux modules de se définir. Chaque module utilise cette interface pour définir son nom, ses dépendances et son implémentation.

Spécifications et organisation

Les recherches terminées, une étape de spécification et de planification de la suite du PFE eu lieu. Lors de cette étape, l'architecture de l'outil d'annotation fut décidée.

4.1 Spécifications

4.1.1 Objectif initial

La première contrainte liée à l'architecture du logiciel fut de suivre les méthodes utilisées dans la partie Java du projet SWAP et de Spring. Il n'était évidemment pas question de recréer Spring en JavaScript, mais plus de réaliser une base permettant d'utiliser les concepts de Spring.

Le but était donc de créer un module de base en JavaScript permettant :

- de gérer un ensemble de modules
- charger des modules et gérer les dépendances
- permettre aux modules de se déclarer simplement
- pouvoir remplacer un module par un autre facilement

Le premier objectif en termes de fonctionnalités était la possibilité de sélectionner une zone de la page, d'associer un nom à cette zone et de l'enregistrer.

4.1.2 Architecture du logiciel

Afin d'atteindre ce premier objectif avec un logiciel extensible et simple à maintenir, définir les principaux modules et leurs fonctionnalités fut important.

Premièrement, l'ensemble du système est découpable en 3 parties :

- JavaScript coté client
- Module Java pour le proxy
- Module Java pour le serveur

Le proxy et le serveur ont en commun les objets contenant les données à stocker. Le proxy devant envoyer ses objets au serveur qui se charge de les sauvegarder. L'étape de récupération des données est aussi à prendre en considération.

Il en est advenu un découpage en 3 projets Java :

- Coeur : pour les classes Java utiles aux deux autres projets,
- Proxy : pour l'intégration de l'interface et la mise à disposition des scripts,
- Serveur : pour la sauvegarde des données et la mise à disposition d'une interface de manipulation des données à distance par le Proxy.

Le diagramme de classe 4.1 réalisé au moment des spécifications, montre bien les différentes séparations entre les parties et projets. Nous retrouvons le `Loader` permettant de gérer les modules JavaScript sans attache. L'`InterfaceExpert`, qui requière différents modules et les utilises de façon à créer les fonctionnalités voulues. Le Proxy et le Serveur implémente les classes d'injection du JavaScript (`InterfaceExpertHtmlHandler`), de communication avec les scripts et le serveur (`InterfaceExpertApi`) et enfin la gestion de la persistance des données (`InterfaceExpertData`).

Bien que, comme nous le verrons dans la partie présentant le résultat finale, ce diagramme a évoluer au fil du projet, la structure et les relations principales entre les classes telles que définies lors de spécifications

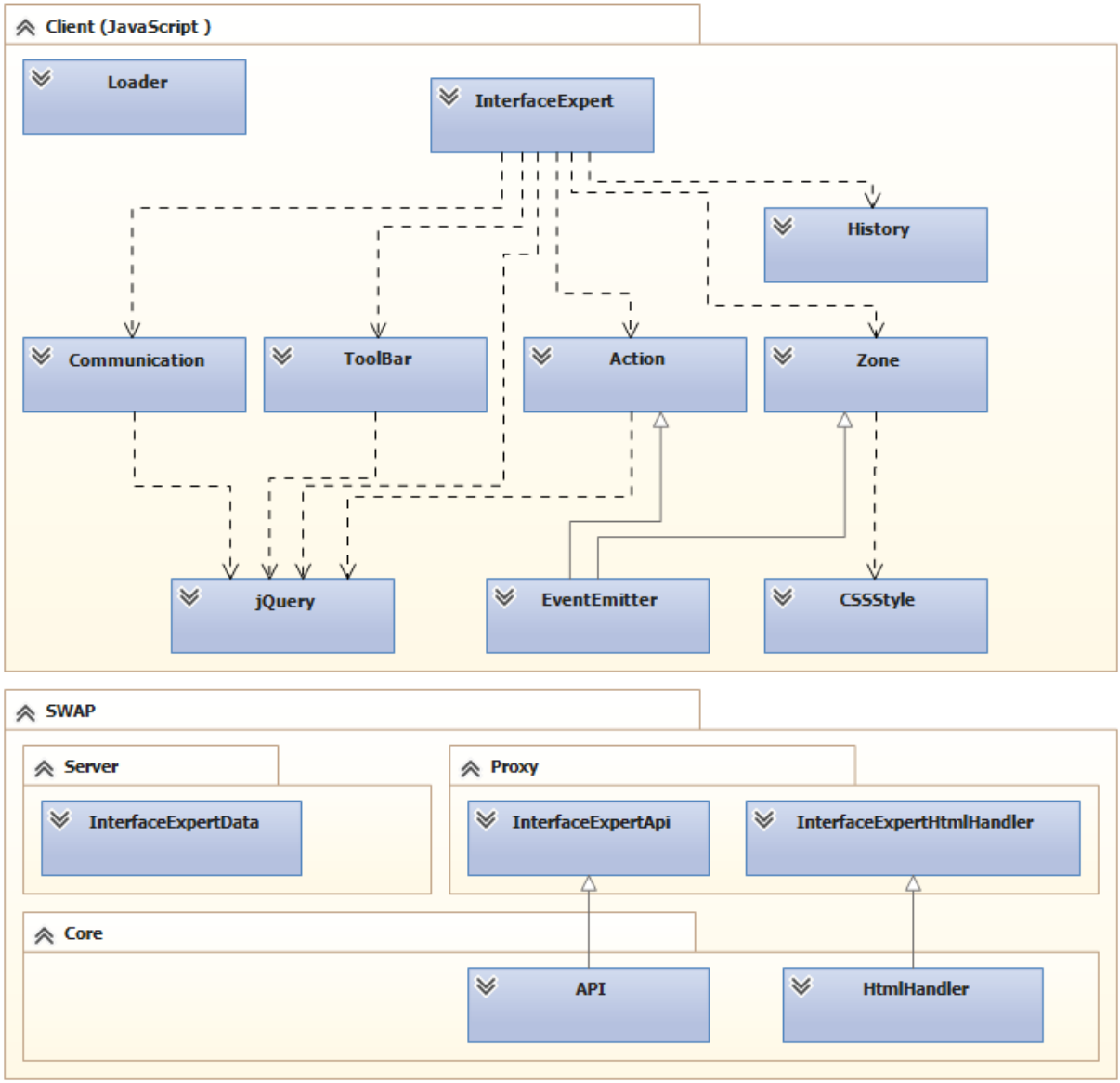

FIGURE 4.1 – Diagramme de classes au moment des spécifications

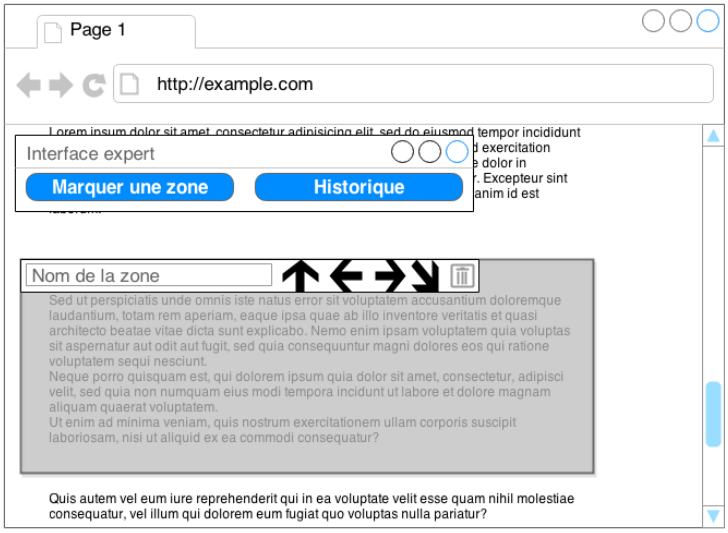

FIGURE 4.2 – Maquette principale de l'interface expert au sein du page web

sont toutefois restées. Pour certaines classes le nom de celle-ci à simplement changer pour en représenter le sens.

4.1.3 Interface utilisateur

Lors des spécifications, l'aspect de l'interface utilisateur fut décidé. L'idée étant de créer une barre d'outils permettant de gérer les zones. Cette barre étant déplaçable dans la fenêtre. Les zones sont visibles en surbrillance et une mini interface était mise à disposition. L'image 4.2 montre bien l'idée initiale.

4.2 Organisation

Afin d'atteindre les différents objectifs visés, une bonne organisation et de bons outils furent nécessaire. Avoir à sa disposition des outils que l'on connaît et qui sont aussi puissant que possible a un impact très positif sur la qualité du code et la productivité.

4.2.1 Organisation

Concernant l'organisation des tâches, j'ai défini pour tous les objectifs les grandes étapes à réaliser pour y parvenir. J'ai ensuite découpé ces étapes en tâches. Une fois le tout ordonné. Chaque grande étape correspondait en méthode Agile à un Sprint qui dans la majorité des cas représenté une semaine.

Au cours de ce PFE, un changement important dans le planning a eu lieu. Deux semaines furent réquisitionner pour l'écriture par mon encadrant et moi-même de 2 articles. L'un à destination de la conférence Handicap 2014 [1], l'autre pour la conférence ICCHP 2012 [2]. Le papier concernant la conférence ICCHP a d'ailleurs été accepté. Ces deux articles sont présents en annexes de ce rapport.

Outre ce décalage, tout au long du PFE le planning a été suivi et c'est avéré correcte, enfin jusqu'à la partie consistant à écrire ce rapport.

1. http ://ifrath.fr/handicap2014/
2. http ://www.icchp.org/

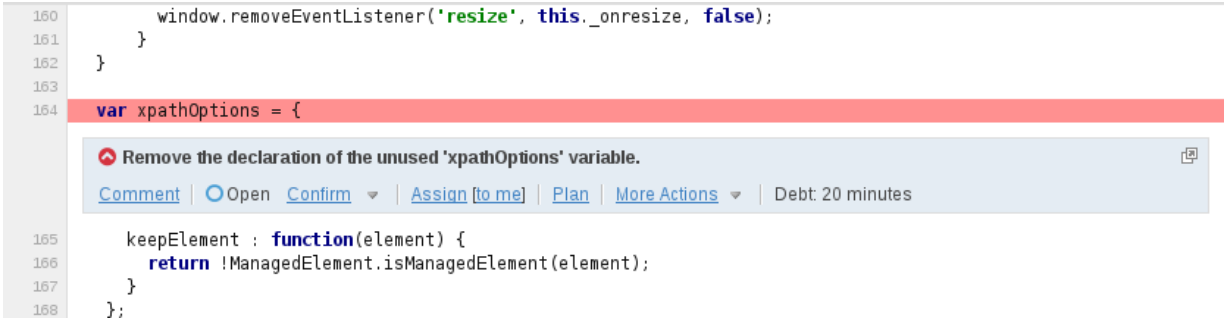

FIGURE 4.3 – Exemple de code inutilisé détecté par Sonar

4.2.2 Outils utilisés

Git

Pour la gestion de version, étant habitué à Git [3], mon encadrant à gracieusement mis à ma disposition un dépôt Git. Cet outil m'a permis tout au long du PFE de garder trace des différentes évolutions de celui-ci. L'un des avantage de Git est sa gestion aisée des branches. Cette fonctionnalité est très utile pour tester sans crainte certaines approches et d'en garder une trace même si en l'état ces modifications ne sont prêtes pour rejoindre le tronc commun.

Eclipse

Afin d'avoir une bonne aide au développement ainsi qu'une bonne intégration des différents outils Java, l'IDE eclipse [4] fut utilisé tout au long du développement. Bien que les fonctionnalités d'eclipse sur la partie JavaScript restent limitées, pour la partie Java, eclipse permet une excellente productivité.

Sabayon Linux

Comme on est généralement plus productif dans un environnement stable et agréable. Ce PFE fut exclusivement réaliser sous un système Linux. La distribution Sabayon [5] fut utilisée et a rempli son contrat : simple à configurer et à utiliser, tout en permettant d'accéder aux dernières versions des outils de développement.

Sonar

Étant donné que tout développeur laisser toujours à un moment ou à un autre des petits bogues dans le code. Un outil d'analyse statique des sources permet bien souvent d'en faire ressortir quelques-uns. J'ai utilisé l'outil Sonar [6] pour analyser à la fois le code Java et JavaScript. Tout au long du projet cet outil c'est révélé très efficace pour détecter des erreurs. Les plus fréquentes étant des erreurs de frappes, des variables inutilisés (Voir 4.3) ou encore du texte collé à cause d'un clic milieu de souris trop sensible. Cette outil m'a donc permit de nettoyer le code et le rendre plus harmonieux.

A titre d'information le projet d'annotations, c'est 17000 lignes : 11000 lignes de code et 3000 lignes de commentaires, le reste étant des lignes vides.

3. http ://git-scm.com/
4. https ://www.eclipse.org/
5. http ://www.sabayon.org/
6. http ://www.sonarqube.org/

Architecture finale

5.1 Gestion des modules : `Loader`

Le module de gestion des modules (`Loader`) fut le premier réalisé. Il se charge de chargé d'autre modules et de gérer leurs dépendances.

En interne, chaque module est décrit par :

- son nom
- son état : initialisation, chargement en cours, chargé
- les fonctions attendant son chargement. Ces fonctions sont appelées par le `Loader` une fois le module prêt. C'est d'ailleurs comme cela que sont gérer les dépendances en interne.
- la valeur retourné par la fonction d'initialisation

5.1.1 Chargement d'un module

Afin que chacun des modules chargés ne soit pas rendu disponible au reste de la page, ceux-ci sont récupérés par le biais d'une requête `XMLHttpRequest`. C'est un objet permettant d'exécuter des requêtes HTTP directement via le JavaScript. Le code des modules est donc récupéré par ce biais. Cette approche ajoute une contrainte supplémentaire par rapport à l'utilisation d'un élément `script` habituel. Du fait que les requêtes sont effectuées vers un nom de domaine différent de celui en cours, l'envoi du contenu des modules est géré par une route particulière sur le proxy afin d'ajouter un en-tête HTTP particulier à la réponse : `Access-Control-Allow-Origin`. Cet en-tête est d'ailleurs nécessaire pour toute les communications entre l'interface et le proxy afin de passer outre ces sécurités

Pour que gestionnaire de module sache quels modules il doit charger, il charge systématique le fichier `main.js` qui contiendra le code demandant les différent modules à charger via la fonction `require` du `Loader`.

5.1.2 Définition d'un module

Les modules doivent pouvoir déclarer leurs prérequis avant exécution. Pour se faire, les modules utilisent la méthode `define` du `Loader`. Cette méthode permet de définir le nom du module chargé, ses dépendances et sa fonction d'initialisation qui sera appelée une fois les dépendances chargées. Le `Loader` se charge d'appeler cette méthode d'initialisation avec pour paramètres les différents modules demandés dans l'ordre

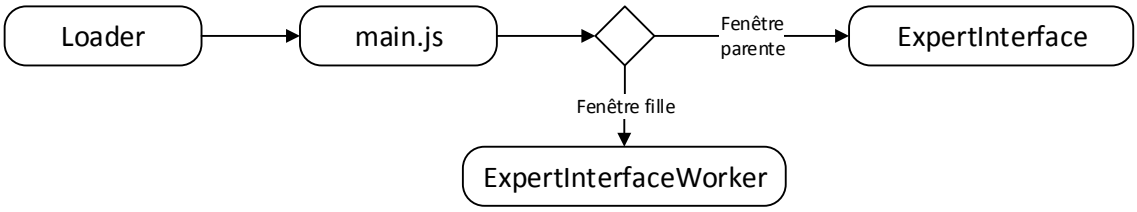

FIGURE 5.1 – Étapes de lancement du module d'annotation

```
1  (function() {
2    // Espace de nom inacessible depuis l'extérieur
3    var Loader = {};
4    (function(loaderObject, loaderEval) {
5        // Définition du loader
6        // Fonctions publiques et privées
7    })(Loader, function(jsContent) {
8      // Seulement les variable jsContent et Loader sont accessible
9      eval(jsContent);
10   });
11   Loader.require('main'); //< Chargement du module main
12 })();
```

FIGURE 5.2 – Structure du code permettant la gestion des espaces de noms du Loader

dans lequel ils sont décrits. Cette fonction doit retourner la définition du module, par exemple une classe (cf Fig. 5.3). C'est cette valeur retournée qui est fourni comme paramètre à la fonction d'initialisation des modules qui dépendent du module tout juste initialisé.

Bien que cela ne fut pas nécessaire pour le développement du module d'annotation, la classe `Loader` met aussi à disposition la méthode `defineAsync` qui a particularité de permettre les dépendances cycliques. Ce bénéfice à évidement un prix : la fonction d'initialisation est appelée immédiatement et il n'est pas possible d'utiliser les modules demandés au moment de son appel. L'unique paramètre fourni est un objet qui sera mis à jour par le `Loader` et qui permet d'accéder aux modules demandés lorsque ceux-ci seront chargés.

5.1.3 Exécution d'un module

L'objet `Loader` est donc rendu accessible aux modules lors de leur exécution. Le code des modules est exécuté via la fonction `eval` qui via une utilisation astucieuse (cf. Fig. 5.2) des espaces de noms JavaScript n'a accès qu'à l'objet `Loader` et à son propre code en plus des variables globales.

Cette méthode d'initialisation permet une isolation totale du JavaScript injecté est un contrôle fin sur les variables mises à disposition. Cependant pour le développement, l'utilisation du débogueur JavaScript présent dans le navigateur est impossible. C'est pourquoi un mode dégradé permettant de charger les modules via une méthode plus classique a été ajouté. Ainsi, lorsque le mode *debug* est actif, les modules sont chargés via la création d'un élément `script` dans le DOM de la page.

5.1.4 Exemple

La figure 5.3 est un exemple réalisé à partir de la vrai définition du module `ManagedElement`. Il met en avant les différentes fonctionnalités offertes par le système de modules. On voit clairement qu'il dépend du module `core/CSSStyle` et que lors de l'initialisation, la variable `CSSStyle` demandée en paramètre permet l'accès à la définition du module `core/CSSStyle`. Lors de l'initialisation présent dans la figure, on peut voir la méthode classiquement utilisée en JavaScript pour déclarer une classe et la faire étendre une autre.

5.2 Gestion des évenements : `EventEmitter`

JavaScript est un langage assez événementielle par nature. Les méthodes `addEventListener` et `removeEventListener` disponible sur l'ensemble des noeuds du DOM en est un bon exemple. Un modèle de gestion des événements, simple à intégrer et à utiliser fut crée dans le but de rendre les différents modules plus simple à utiliser. Cela est d'autant plus utile qu'une bonne partie des modules mettent à disposition des fonctionnalités liées à l'interface utilisateur.

```
1  Loader.define('core/ManagedElement' //< Nom du module
2      , ['core/CSSStyle'] //< Liste des dépendances
3      , function(CSSStyle) { //< Fonction initialisant le module
4
5    // Constructeur de la classe ManagedElement
6    var ManagedElement = function(element, className) {
7      //Appel au constructeur de la classe parente :
8      CSSStyle.apply(this, arguments);
9      /* code */
10   }
11
12   // Fonction membre statique de la classe
13   ManagedElement.isManagedElement = function(element) { /* code */}
14
15   // La classe ManagedElement étends la classe CSSStyle
16   ManagedElement.prototype = Object.create(CSSStyle.prototype);
17   ManagedElement.prototype.constructor = ManagedElement;
18
19   // Fonction membre de la classe ManagedElement
20   ManagedElement.prototype.remove = function() { /* code */}
21
22   return ManagedElement; //< Partage de la classe
23 });
```

FIGURE 5.3 – Exemple concret simplifié de definition du module `ManagedElement` avec héritage

Ce module nommé `EventEmitter` s'inspire donc du modèle utilisé par le DOM et du module similaire utilisé par nodejs[1]. Il est basé sur 4 fonctions principales :

- `emit(eventName, ...)` : pour émettre un événement nommé `eventName`
- `on(eventName, callme)` : pour écouter un événement. La fonction `callme` est appelée lorsque l'événement `eventName` est émis. L'ensemble des paramètres utilisés après `eventName` lors de l'appel à `emit(eventName, ...)` sont disponible lorsque que `callme` est appelée.
- `once(eventName, callme)` : pour écouter un événement une unique fois.
- `off(eventName, callme)` : pour arrêter d'écouter un événement.

Une simple fonction permet de donner la capacité à un objet de gérer des événements : `implementIn(object)`. Une fois cette fonction appelée, l'objet dispose des fonctions présentées précédemment.

5.3 Communications interframes : `CrossSiteCommunication`

Afin de supporter un maximum de page web différente, le cas assez commun des pages webs intégrant une `iframe` est particulier. Dans ce cas, il n'est plus possible d'accéder directement au DOM de l'iframe depuis la page mère. Différentes sécurités mises en place par le navigateur et nécessaire pour la sécurité de l'utilisateur étant présentes. Comme il toujours possible d'intégrer du code dans l'iframe et que l'on souhaite avoir une unique interface pour gérer l'ensemble de la page Web, un système de communication a été mis en place.

Une seule API est disponible pour ce genre de communication : `window.postMessage`[2]. Cette API permet de passer des messages à d'autres frames de façon événementielle.

Cette API impose d'avoir connaissance de la fenêtre à appeler pour être utilisable. Le moyen le plus fiable pour que la fenêtre mère dispose d'une telle informatique est que la fenêtre fille envoie une demande

1. nodejs est un logiciel permettant d'exécuter du JavaScript et ciblant particulière les applications réseaux : http ://-nodejs.org/
2. https ://developer.mozilla.org/en-US/docs/Web/API/Window.postMessage

d'enregistrement. Demande qui est émise toute les secondes tant que la fenêtre mère n'a pas répondu avec succès.

Dans le but de limiter le plus possible les capacités d'un site web à utiliser cette api pour accéder à des fonctionnalités non voulue. Toutes les communications sont cryptées avant transmission. La librairie CryptoJS [3] et l'algorithme de chiffrement AES sont utilisés à cet effet.

Afin de rendre l'API de communication la plus simple à utiliser possible, celle-ci implémenté un système événementielle via les fonctions suivantes :

- `emit(window, eventName, ...)` : pour émettre un événement nommé `eventName` vers la fenêtre `window`.
- `broadcast(eventName, ...)` : pour émettre un événement nommé `eventName` à toutes les autres fenêtres enregistrées.
- `on(eventName, callme)` : pour écouter un événement. La fonction `callme` est appelée lorsque l'événement `eventName` est émis. L'ensemble des paramètres utilisés lors de l'appel à `emit(eventName, ...)` sauf `eventName` sont disponible lorsque que `callme` est appelée. Le premier paramètre lors de l'appel est donc la fenêtre appelante.
- `once(eventName, callme)` : pour écouter un événement une unique fois.
- `off(eventName, callme)` : pour arrêter d'écouter un événement.

Ce module est donc très simple à utiliser. Cependant, il reste une contrainte liée à son utilisation : les données sont sérialisés avant transfert. Cela implique des restrictions par rapport aux transferts d'objets.

Pour utiliser ce module, il suffit de le charger partout où il est nécessaire de communiquer, il se charge automatiquement de détecter s'il est dans la fenêtre mère.

5.4 Annotations : `Zone`

La description d'une annotation dans la page est faite via la classe `Zone`, c'est la classe de base de toutes les annotations. Cette classe permet :

- l'affichage de la zone annoté,
- de sélectionner une zone de la page au curseur,
- de gérer les différentes propriétés d'une zone.

Voici une description des principales fonctions de l'API d'une zone :

- `info()` : permet d'obtenir toutes les informations nécessaires pour gérer l'affichage de l'état de la zone dans l'interface.
- `data()` : permet d'obtenir toutes les données nécessaires pour sauvegarder la zone.
- `loadData(data)` : permet de charger une zone à partir des données sauvegardées via `data()`.
- `remove()` : permet de supprimer la zone.
- `moveTo()` : permet de déplacer la zone.
- `target` : propriété spéciale décrivant l'élément actuellement ciblé par la zone. Changer sa valeur change le positionnement de la zone.
- `xpath` : propriété spéciale décrivant le XPath [4] de l'élément actuellement ciblé par la zone. Changer sa valeur change le positionnement de la zone et donc la propriété `target`

5.4.1 Gestion des iframes

Au cours de développement, la question des iframes a posé un problème complexe à résoudre. Gérer la synchronisation des zones avec l'interface était un réel casse-tête. Afin de simplifier les choses, il fut décidé de créer une classe d'abstraction : `RemoteZone`. C'est uniquement via cette classe que l'interface gère les zones. Les appels aux fonctions sont légèrement plus longs à écrire, mais il n'y a plus à gérer les cas particulier. Cette classe d'abstraction se chargeant de relayer les appels jusqu'à l'objet final.

3. https ://code.google.com/p/crypto-js/
4. Le XPath est un moyen permettant de décrire le position d'un élément dans le DOM

L'ensemble des zones créées est gérer par la classe statique `Zones`. Ce gestionnaire de zones permet de faire abstraction de la complexité induite par la gestion des iframes. C'est lui qui met à disposition les différentes `RemoteZone` via une interface événementielle.

La gestion des événements pour les zones est particulière. L'ensemble des événements émis est capturé par la classe les manageant : `Zones` du côté de la fenêtre mère, `ZonesWorker` pour les fenêtres filles. Comme le but de ces classes est de permettre une abstraction totale par rapport à la gestion des iframes, les événements capturés sont retransmis attachés à la classe d'abstraction `RemoteZone` qui met à disposition :

- la fonction `run(methodName, ...)` : permettant d'appeler la fonction `methodName` avec les autres paramètres sur la zone. La valeur retournée par l'appel à la méthode n'est cependant pas accessible. Les communications inter-frames étant asynchrones.
- la fonction `set(propertyName, value)` permet de modifier une propriété d'une zone.
- la propriété `info` est un objet crée par la fonction `info()` de la zone et est suffisant pour l'affichage de l'état de la zone par l'interface. Cette propriété est automatiquement mise à jour en cas de modification de la zone.

5.4.2 Sélection d'une zone de la page

Commençons par la sélection d'une zone dans la fenêtre actuelle, sans prendre en considération le problème des iframes.

Lors du déclenchement de l'étape de sélection, les événements de la souris : `mousemove` et `mousedown` sont écoutés de façon à pouvoir visualiser la zone sélectionnée et stopper la sélection lors d'un clic souris. A chaque fois que la souris bouge, l'élément présent sous le curseur est récupéré. Si celui-ci n'est pas admissible pour cette zone, alors c'est l'élément en dessous de celui-ci qui est testé et ainsi de suite. La fonction `isValidZoneTarget(element)` permet cette vérification. C'est cette fonction qui empêche une zone de sélectionner une autre zone.

Pour accéder à l'élément en dessous de l'élément testé, il faut tout d'abord caché l'élément testé, puis demander au navigateur de trouver l'élément sous la souris. Une fois tous les tests faits, les éléments cachés sont remis à leurs états initial. L'élément est ainsi déterminé.

L'affichage de la visualisation de la zone sélectionnée par l'utilisateur est aussi plus complexe qu'il n'y paraît. Une approche naïve serait de récupérer la taille et position de l'élément dans la fenêtre et de créer un élément positionné par rapport à cette fenêtre. Une telle approche poserait alors le problème suivant : comment garantir que la zone mis en avant correspond bien à l'élément ? Par exemple, si l'utilisateur descend une liste déroulante. Pour gérer cette problématique, l'élément affichant la zone sélectionnée est inséré en tant que dernier enfant du plus proche parent de l'élément ciblé servant à son positionnement. La détermination de ce plus proche parent se fait par un parcours depuis l'élément vers la racine en testant la propriété CSS `position`. Si celle-ci est différente de `static`, c'est que l'élément ciblé est positionné de façon relative à ce parent. Cette approche permet de ne gérer la mise à jour du positionnement que lors des redimensionnements de la fenêtre.

Maintenant que nous avons vu comment est déterminé la sélection, examinons le cas avec les iframes. Bien que le système d'abstraction soit très pratique, il ne résous pas ce problème. En effet, lors d'une sélection, l'utilisateur peut passer d'une frame à une autre. Il faut donc que le script de sélection soit lancé sur l'ensemble des frames. Enfin il faut synchroniser l'état des différentes zones gérant les sélections afin que lorsque le curseur sort d'une frame, la dernière zone sélectionné dans celle-ci soit retiré.

C'est pour cette raison que je n'ai pas décrit l'API de la classe `Zone` permettant de lancer une sélection. C'est à nouveau la classe `Zones` qui va se charger de gérer l'abstraction est de coordonner le tout. La fonction `Zones.startSelection()` permet de lancer une sélection sans avoir à se poser la question des iframes.

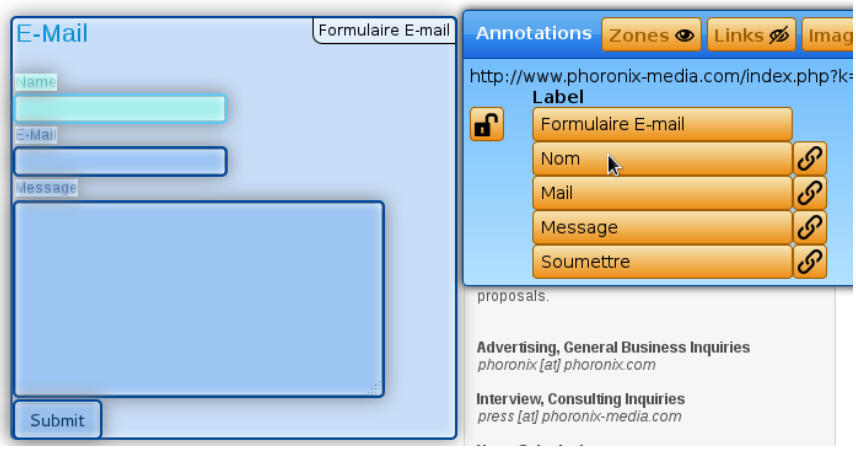

FIGURE 5.4 – Capture d'écran de l'interface d'annotation sur un formulaire. On peut voir la partie en surbrillance qui correspond à ce qui est actuellement en cours de modification.

5.4.3 Lien avec le texte

Les zones sélectionnées sont souvent cités dans différents textes et pouvoir référencer certaines parties d'un texte comme lié à un zone est donc utile.

La sélection d'une partie d'un texte ne peut se faire avec une zone, car le texte sélectionné peut concerner plusieurs éléments de façon partielle. Une classe spéciale à donc été créée pour gérer ce type de sélection : `ZoneSelection`.

Cette classe permet de lancer une sélection textuelle. Pendant cette sélection, toutes les zones sont cachées afin de permettre à l'utilisateur de sélectionner ce qu'il souhaite. Une fois la sélection faite, celle-ci est récupéré est analysé. Le navigateur gère les sélections via 4 propriétés :

- l'élément qui contient le début de la sélection
- l'offset du début de la sélection dans l'élément de début
- l'élément qui contient la fin de la sélection
- l'offset de la fin de la sélection dans l'élément de fin

A partir de ces propriétés, il faut en déduire la zone à dessiner pour afficher la sélection réalisé par l'utilisateur. Pour cela on détermine les rectangles à dessiner pour suivre la forme de la sélection ainsi que le parent commun le plus proche permettant de placer correctement la sélection affiché même lorsque celle-ci est dans un espace défilant. Ce parent commun est déterminé via la même technique que celle utilisé pour la sélection de zone.

Comme une sélection textuelle est en générale difficilement représentable par un unique rectangle, un élément intermédiaire est inséré à l'emplacement trouvé et les éléments affichant la visualisation en tant que telle sont ajouté ensuite en tant qu'enfants. Cette approche permet de simplifier les mises à jour.

5.4.4 Types de zones

Afin de donner plus de possibilité d'annotation, en fonction des éléments sélectionnés par une zone, un comportement particulier a été mis en place. Pour les liens et urls cela correspond simplement à l'ajout de données supplémentaires : l'url du lien et la source de l'image.

Concernant les tableaux. Le but est de pouvoir distinguer ceux qui servent à l'affichage de données et ceux qui servent à la décoration. Il faut aussi pouvoir sélectionner l'en-tête du tableau. Pour les formulaires le problème est encore plus complexe. Il faut que l'utilisateur puisse pour chaque entrée du formulaire définir une sélection textuelle associé ainsi que la description du champ.

Afin de gérer cette nouvelle problématique, une solution applicable aux tableaux et aux formulaires fut

choisie. Chaque zone obtient la capacité de définir des sous-zones. Ces zones sont rendus accessibles via quelques légers changements dans l'API, notamment sur la couche d'abstraction afin de donner la capacité à l'interface de communiquer avec les sous-zones. Contrairement aux zones classiques, ces sous-zones en sont pas disponible directement par l'interface. Celle-ci doit toujours passer par la zone mère pour y accéder.

5.5 Gestion du DOM et des styles

Afin de gérer les changements apportées au DOM et de permettre l'application de styles aux éléments, une couche d'abstraction a là aussi été mise en place. Le but de cet ensemble de classe est de mettre à disposition l'ensemble des API les plus courantes pour créer facilement une interface utilisateur.

5.5.1 CSSStyle

`CSSStyle` est la classe dont dépend tout élément créé par l'interface d'annotation. Elle permet de gérer un élément du DOM. Elle fournit les méthodes nécessaires au contrôle des styles appliqués à l'élément. Lorsqu'un élément est donnée à contrôlé à cette classe, c'est à dire lors de l'instanciation de celle-ci, l'élément est réinitialisé aux styles initiaux définit par le W3C.

Cette réinitialisation se fait via l'application de la valeur `initial` à l'ensemble des propriétés CSS de l'élément. Il faut faire attention, car cette réinitialisation remet l'élément dans un état différent de celui présent dans une page web vierge. En effet, la définition W3C [5] définie des valeurs particulières. Ainsi par exemple, la propriété `display` est toujours placée à `inline`. Cette approche permet tout de même d'assurer que l'élément est dans un état que l'on contrôle, même s'il faut tout de même ajouter quelques règles CSS pour retrouver un état *habituel*. A cette étape de réinitialisation, `CSSStyle` ajoute la classe `global-`*`TAGNAME`* pour permettre de contrôler plus finement l'état des éléments par nom de tag.

Gestion initiale des propriétés CSS

Lors de la première implémentation, la gestion des propriétés CSS étaient géré *à la main*. La classe `CSSStyle` était à ce moment-là une sorte de moteur CSS. C'est à dire que c'est l'implémentation qui gérée l'application des règles CSS, la mise à jour des styles, les règles de `cascading` CSS et de priorité.

Cette méthode fut utilisée pendant presque 2 mois. Elle avait comme avantage de permettre un contrôle complet sur les propriétés appliquées à un élément. Cependant la mise à jour et la déduction de valeur à appliquer aux propriétés étaient difficiles à gérer et posaient des problèmes de performances.

Gestion native des propriétés CSS

Afin de simplifier la gestion des règles CSS. L'objectif fut de trouver un moyen de rendre cette gestion au navigateur. Sans pour autant laisser apparaître des problèmes de conflits.

L'idée trouvée et implémentée fut d'utiliser une méthode hybride. Afin de conserver la compatibilité dans le temps, il n'était pas question de perdre les avantages liés à la réinitialisation des propriétés CSS de façon automatique. Le fait qu'il soit possible de créer des règles CSS via JavaScript et la possibilité d'énumérer l'ensemble des propriétés CSS disponible dans un navigateur a permis de créer une règle CSS de réinitialisation. Afin de générer un ensemble de règle CSS unique, un préfixe considéré comme unique est ajouté en début de chaque règle CSS créée.

Le nouveau système repose donc sur un ensemble de règle CSS créées et insérées dans la page via JavaScript. L'API permettant d'appliquer les styles n'a pas eu à changer d'un iota. L'implémentation fut alors fortement simplifiée et réduit à la génération de règle CSS et la détermination du nom des règles à appliquer sur un élément. Le bénéfice en termes de performance fut la aussi très net.

5. http://www.w3.org/TR/CSS21/about.html#initial-value

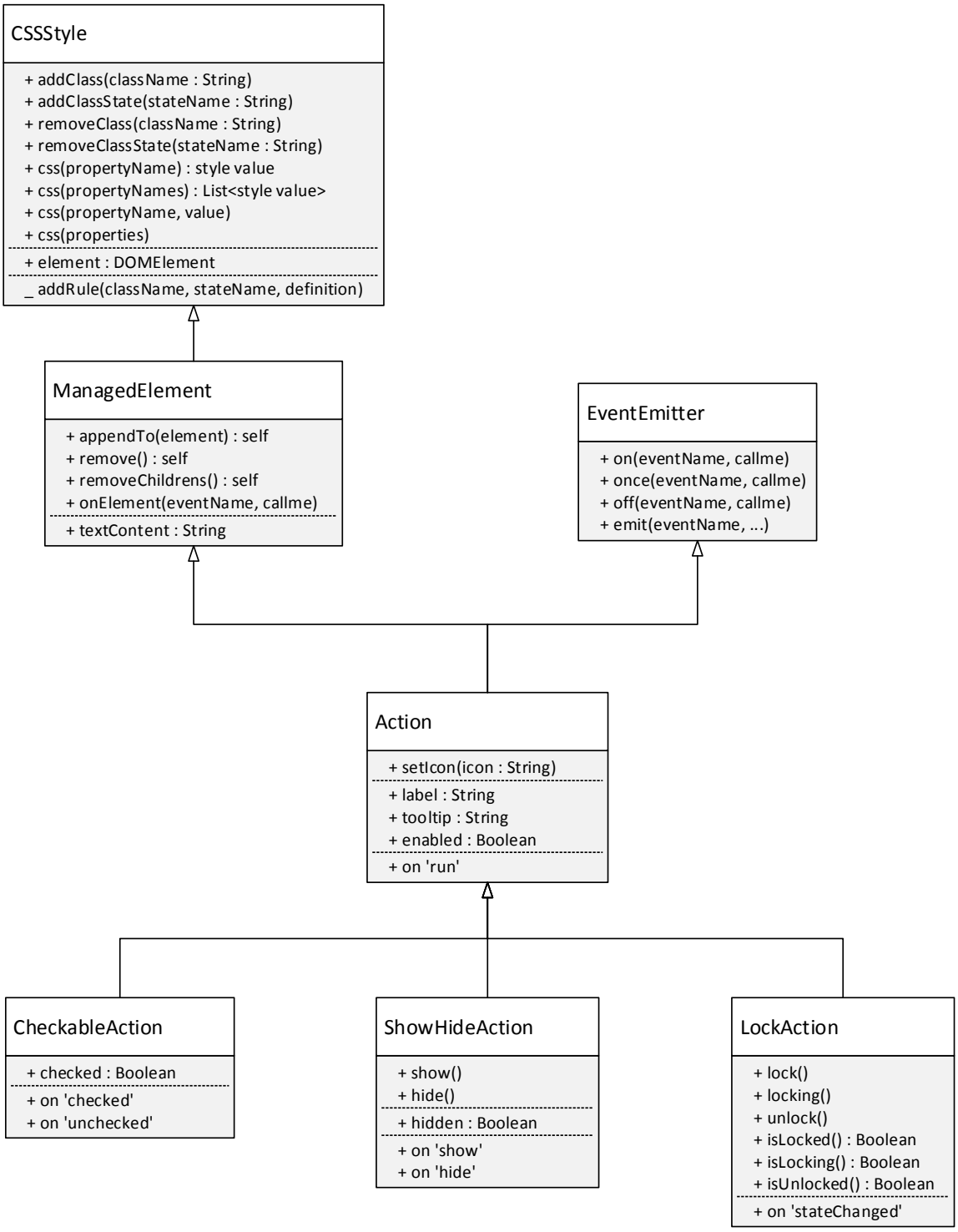

FIGURE 5.5 – Diagramme de classes des objets permettant la création de l'interface utilisateur.

5.5.2 ManagedElement

La gestion des styles sur un élément étant fait par `CSSStyle`, il fallait toujours simplifier les méthodes d'utilisation d'un élément. Pour se faire la classe `ManagedElement` met à disposition un petit nombre de fonctions permettant d'améliorer fortement la lisibilité du code ainsi que l'interaction entre éléments managés. La plupart des fonctions retournent l'objet lui-même, ce qui permet de chaîner les appels. Cette approche inspirée par la très connue librairie jQuery, permet la aussi d'améliorer la lisibilité du code. Au risque tout de même en cas d'abus du chainage d'obtenir l'effet inverse.

Par exemple, au lieu ajouter un élément fils à un élément parent via les fonctions du DOM, la fonction `appendTo(element)` accepte à la fois un élément du DOM ou un élément managés et détermine seule comment doit se réaliser l'ajout.

5.5.3 Action

Créer des éléments managés étant désormais simple, il restait à créer différentes classes permettant certaines interactions courantes. La plus utilisés dans l'interface d'annotation étant un bouton. La classe `Action` est là pour remplir ce rôle. Elle permet les fonctionnalités classiques d'un bouton avec un comportement assuré cohérent pour l'ensemble des navigateurs tout en ajoutant la possibilité de personnalisation du contenu.

La classe `Action` n'utilise pas l'élément `Button` classique, mais un élément `div` pour lequel certains événements tels que `click` sont écoutés. Cela permet à un bouton de l'interface d'avoir la forme et le contenu que l'on souhaite. Cette capacité est d'ailleurs fortement utilisée par les classes héritant d'`Action`.

La figure 5.6 montre l'ensemble des actions utilisées pour l'interface d'annotation.

CheckableAction

Cette classe définit une checkbox[6] tout en conservant les mêmes capacités qu'une action classique. La propriété `checked` permettant d'accéder et de changer l'état de la checkbox. L'affichage de l'état est géré via l'utilisation d'un icône.

LockAction

Cette classe permet de gérer les états : verrouillé, en cours de verrouillage et déverrouillé. L'interface de gestion des zones utilise cette classe pour gérer simplement l'état sauvegardé ou non d'une zone. L'aspect visuel de ce bouton correspond à un icône différent en fonction de l'état.

ShowHideAction

L'interface permettant de passer d'une zone avait besoin d'un moyen d'afficher et cacher les zones d'un mode. Une classe permettant cela fut donc créée. Elle reprend les fonctionnalités d'un bouton et ajoute un mini-bouton à droite permettant de passer de l'état affiché à l'état caché et vice versa.

5.6 Interface utilisateur

Passons maintenant à l'interface utilisateur. Celle-ci est déplaçable grâce aux capacités offertes par le module `ui/Draggable`. L'interface se compose de deux parties :

- Un en-tête composé de boutons permettant de choisir quoi faire
- Un espace permettant d'afficher les actions relatives à l'action en cours.

On peut voir l'interface utilisateur dans le mode *Zones* sur la figure 5.6.

6. Une checkbox est un élément cochable et décochable

FIGURE 5.6 – Capture d'écran de l'interface d'annotation.

5.6.1 Gestion des modes

Via les boutons disposés dans l'en-tête, l'utilisateur peut changer de mode. Pour l'interface d'annotations, un mode correspond à un type de zone à gérer. Afin de permettre à l'utilisateur de gérer facilement l'affichage des zones, les boutons classiques permettant de passer d'un mode à l'autre sont équipés d'un mini-bouton activable et désactivable qui permet d'afficher et cacher les zones gérer par le mode visé. Le passage d'un mode à l'autre désactive le mode précédent.

Pour simplifier l'écriture de l'interface, chaque interface d'un mode est décrit selon la même API. Chaque mode devant définir via cette API son apparence.

5.6.2 Détails

Afin de rendre l'interface la plus simple et efficace, les fonctionnalités suivantes ont été implémenté :

- Cliquer sur une zone affiché dans la page permet d'accéder à sa définition dans l'interface et affiche l'interface en surbrillance.
- Passer le curseur au-dessus de l'interface permettant la gestion d'une zone met en surbrillance la zone dans la page. La gestion de la surbrillance se fait aussi pour les sous-zones et les sélections textuelles liées aux zones.
- L'état des contrôles permettant de modifier une zone dans l'interface est instantanément visible et mis à jour en temps réel.

La mise en place de ces détails fut facilité par le design de l'application. Le modèle événementielle et les différentes capacités d'abstraction ont fait que la réalisation de ces fonctionnalités prends seulement quelques lignes de code.

5.7 Communication entre le proxy et l'interface

La communication entre le proxy et l'interface d'annotation est unidirectionnelle. C'est à dire que c'est toujours l'interface qui initie une demande et le proxy qui y répond.

L'implémentation du système de communication coté JavaScript repose sur la fonction `ajax` mise à disposition par la libraire `jQuery`[7]. L'utilisation de jQuery permet de s'affranchir des quelques problèmes encore présent lors de l'utilisation de l'API native sur certains navigateurs même récents (ex : Internet Explorer). Cette implémentation prend la forme d'un module (`ProxyCommunication`) mettant à disposition une classe statique définissant deux méthodes : `get` et `post`. Ces fonctions permettent d'envoyer respectivement des requêtes HTTP `GET` et HTTP `POST` au proxy.

Coté proxy, la récupération des requêtes HTTP utilise le système Web MVC de Spring sur des routes spécialement allouées et définissant l'API. La mise en place de ces routes reprend la méthode utilisé dans le projet SWAP. Au final, chaque requête arrive dans une fonction particulière du composant gérant l'API.

7. http://jquery.com/

Comme cela fut brièvement expliqué dans la section 5.1.1, pour que la communication passe les restrictions liées à la sécurité, il faut que dans la réponse fournie par le proxy, l'en-tête HTTP `Access-Control-Allow-Origin` est requis. En effet, afin d'éviter de permettre au site web d'envoyer des requêtes vers d'autre site web a des fins malveillante, cette sécurité est présente dans tous les navigateurs. Afin de simplifier l'ajout de cet en-tête l'API dispose d'une méthode permettant de préparer la réponse avant envoi et donc d'ajout l'en-tête.

5.8 Persistance des données

Pour la sauvegarde des zones dans le temps, il faut conserver les différentes méta-données décrivant la zone ainsi que la page complète. En effet, même si la page change, certains outils peuvent nécessiter un accès à la page originale pour certaines tâches. Il faut aussi pouvoir retrouver facilement la position d'un élément dans une page web.

5.8.1 Exportation du DOM

Exporter le code HTML représentant la totalité de la page ou le contenu d'un élément pourrait être très simple. Seulement, la présence de l'interface d'annotation au sein de la page et le besoin d'anonymisation de certaines pages complique quelque peu la tâche. Il faut donc avoir la capacité d'exclure des éléments de l'exportation, mais aussi transformer le contenu afin de le rendre anonyme avant transfert.

La méthode implémentée consiste à copier la totalité du DOM, puis à modifier celui-ci avant de récupérer le code HTML correspondant. Pour cela, l'API du DOM permet de facilement récupérer le code HTML d'un élément, elle permet aussi le clonage d'un élément et de tous ces enfants. Il reste donc à filtrer les éléments et à anonymiser le contenu.

Pour le filtrage, un parcours complet du DOM copié est effectué et chaque élément est testé via une fonction personnalisable déterminant s'il faut conserver ou non l'élément. La même chose est aussi possible pour filtrer les attributs des éléments. Concernant l'anonymisation, lors du parcours du DOM copié, une fonction permet de déterminer si un élément doit être anonymisé ou non. La même chose est à nouveau possible pour les attributs. Si le contenu doit être anonymisé, celui-ci est alors modifié via une fonction dédié. Cette fonction est là aussi personnalisable.

L'implémentation par défaut de la fonction d'anonymisation à pour but de conserver la forme. Elle effectue ainsi les transformations suivante :

- les chiffres sont tous transformés en `1`,
- les lettres minuscules en `a`,
- les lettres majuscules en `A`,
- les symboles sont conservés,

Afin que cette fonction soit la plus fiable possible et soit capable de gérer les différents encodages, une librairie JavaScript [8] permettant de trouver la classe d'un caractère unicode fut utilisée. Cela permet la prise en compte des accents et des langues étrangères de façon fiable. Cette librairie est nécessaire car il n'existe pas en JavaScript de moyen de gérer les caractères Unicode simplement et les expressions régulières disponibles n'intègre pas les fonctionnalités Unicode.

5.8.2 Position d'un élément

Afin de sauvegarder la position d'un élément dans la page et pouvoir retrouver celui-ci lors d'une future visite, il est nécessaire d'utiliser un moyen le permettant. Pour ce PFE, j'ai choisi le `XPath`. C'est une méthode souvent utilisé dans les documents XML. Elle permet la sauvegarde de la position de l'élément dans le DOM en se basant sur la position de l'élément par rapport à ses parents. Comme il n'existe pas de fonction permettant de trouver le XPath d'un élément, il a fallu en créer une.

8. http://inimino.org/~inimino/blog/javascript_cset

La création du XPath consiste à parcourir le DOM depuis l'élément jusqu'à la racine en passant par l'ensemble des pères. Pour chaque élément parcouru, on détermine la position de celui-ci parmi tous les enfants du même type de son parent. On obtient ainsi un résultat comme celui-ci : `/html/body/div[1]/h1[1]`.

Ecrire une fonction permettant de retrouver un élément à partir du XPath aurait pu être amusant, seulement cette fois-ci le DOM possède une fonction de l'API le permettant : `document.evaluate`[9]. Comme à son habitude Internet Explorer n'implémente pas cette méthode, enfin pas avec le même nom et les mêmes paramètres : `document.selectSingleNode`[10]

Le XPath fonctionne en général plutôt bien, même sur certaines pages au contenu fortement variable. Cependant, il n'est pas infaillible non plus. La gestion de contenu variant fortement étant la plus grosse difficulté. D'autres méthodes sont aussi possible mais elles non cependant pas étaient implémentées :

- marquage unique par le proxy des éléments : fonctionne uniquement sur les pages totalement statiques.
- utilisation d'un ensemble de propriétés de l'élément pour tenter de le retrouver par la suite dans la page par tentative d'association et l'utilisation d'une marge d'erreur.

5.8.3 Enregistrement & Récupération

Une fois les données à enregistrer générées, il faut les sauvegarder. Le proxy se charge de récupéré et vérifier les données envoyées par l'interface. Puis elle envoie ces données au serveur pour enregistrement (cf Fig. 5.7).

La gestion des données au niveau du proxy et du serveur utilise les mêmes classes (cf. Fig. 5.8). Différentes annotations ont été appliqués à ces classes et à leurs attributs afin de leurs donner les propriétés de sérialisation Json et le comportement à l'enregistrement souhaité. C'est donc via les annotations Java que la base de données est modélisée. L'utilisation des annotations est un réel avantage pour simplifier la gestion des données. Il faut toutefois faire attention au comportement de la base de données vis à vis de ceux-ci. Certains changements d'annotation pouvant causer problèmes. Mais une fois la bonne combinaison d'annotation trouvée, le système est stable. Enfin, il a toutefois fallu mettre à jour Hibernate afin de corriger un bogue bloquant empêchant l'enregistrement du contenu Html sous la forme d'un type `TEXT` et non `VARCHAR` comme c'est le cas par défaut pour les chaînes de caractères.

Le transfert du modèle Javascript au modèle Java s'est fait via une sérialisation JSON par l'utilisation du module de communication coté JavaScript et par la création d'une route coté Java. Une fois les données récupéré en Json sur le proxy, la librairie Jackson[11] à permit de mapper[12] les données sur les objets Java. Une fois les données converties au modèle Java, celui-ci est vérifié avant envoi au serveur. L'envoi du modèle au serveur ce fait via une interface. Le framework Spring entre à nouveau en jeu et par le biais d'une simple annotation, il met à disposition du proxy un objet implémentant l'interface demandé et dont les appels sont automatiquement transmis au serveur. Du côté du serveur, il suffit de déclarer l'implémentation de l'interface et de la définir comme `@Component` pour que Spring s'occupe de faire automatiquement le lien.

Maintenant que les données sont arrivées coté serveur, il reste plus qu'a les enregistrer. Pour rappel, le modèle contient déjà tout ce qu'il faut pour gérer la sauvegarde, il suffit alors de donner ce modèle à la base de données. Là encore, le framework Spring permet via l'annotation `@Repository` de mettre facilement à disposition un objet initialisé par un autre module de Swap et permettant l'enregistrement du modèle.

La récupération des données utilise exactement les mêmes principes : l'interface demande au proxy les données, le proxy relaye la demande au serveur qui effectue la recherche dans la base de données et retourne la liste des Zones. Cette liste est alors convertie en Json. Cette fois la conversion est même géré automatiquement par Spring. L'interface récupère alors le modèle qu'elle avait envoyé lors d'une précédente visite et l'utilise pour recréer les zones.

9. https://developer.mozilla.org/en-US/docs/Web/API/document.evaluate
10. http://msdn.microsoft.com/en-us/library/ms757846(v=vs.85).aspx
11. http://wiki.fasterxml.com/JacksonHome
12. Le mappage consiste à passer les données d'un objet à un autre par comparaison des attributs communs.

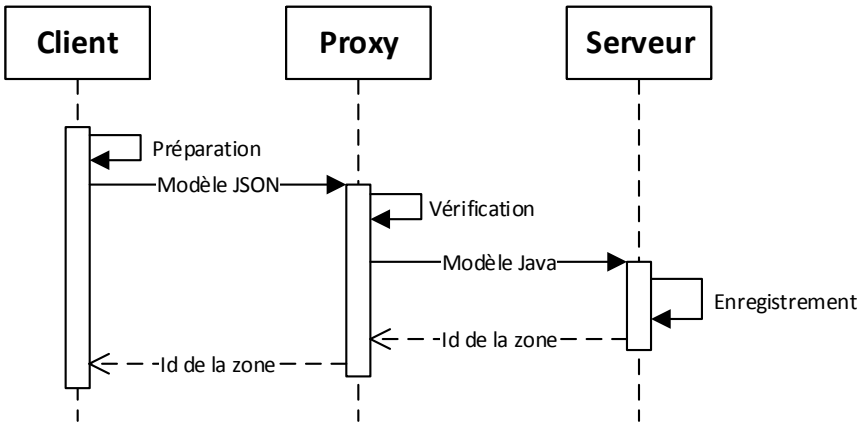

FIGURE 5.7 – Diagramme de séquence montrant le fonctionnement de la sauvegarde des données

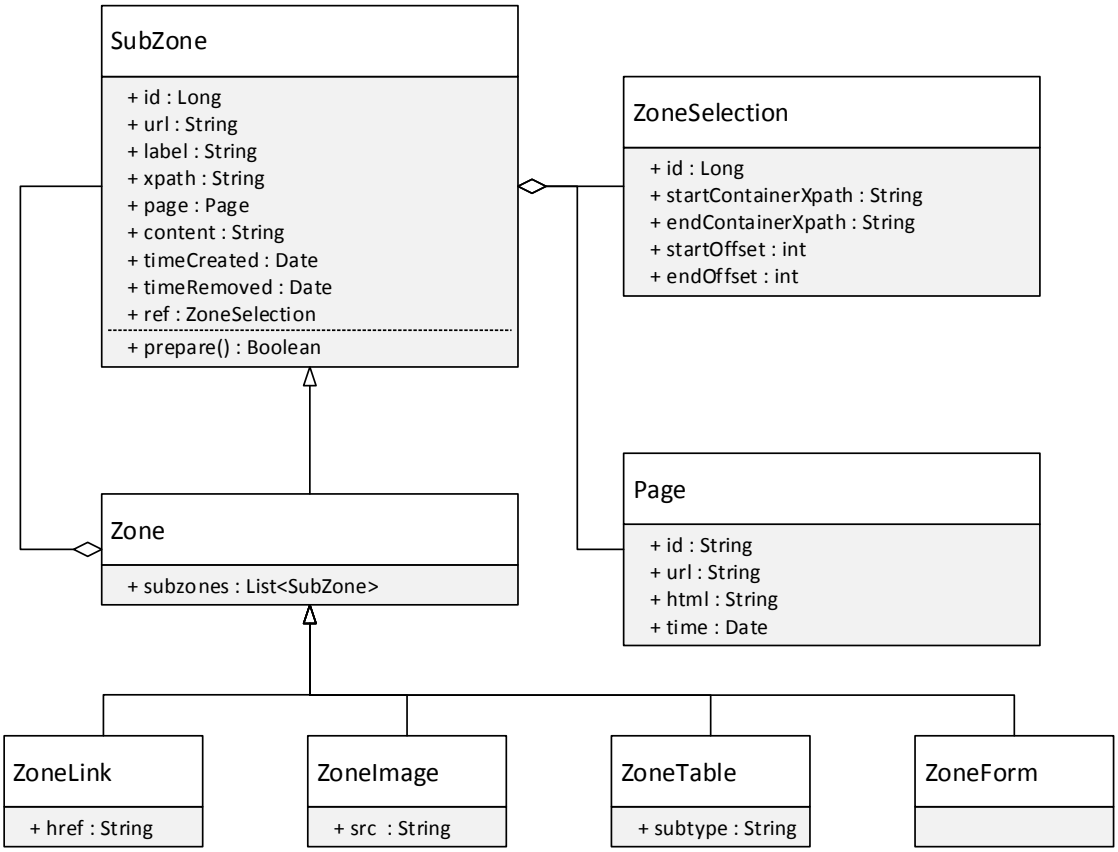

FIGURE 5.8 – Diagramme des classes modélisant les données coté proxy et serveur. Les attributs visibles ici sont en réalité accessible uniquement via des *getters* et *setters*

5.9 Sécurisation d'appels à distance

Le dernier objectif de ce PFE fut la sécurisation des communications entre le proxy et le serveur. Le but était de mettre en place une connexion SSL authentifié des deux côtés. C'est à dire que le serveur n'accepte des requêtes que s'il peut vérifier le certificat client. Et le client envoi la requête uniquement s'il peut vérifier le certificat du serveur.

La mise en place d'un tel système demande 2 choses :

- un échange de certificat avant communications
- des changements dans la manière dont les communications sont établis.

5.9.1 Gestion des certificats

Coté serveur comme coté proxy, un système de gestion des certificats a été mis en place par le biais de 2 classes. Ces classes permettent de gérer les différents magasins de clés et certificats qui seront nécessaire pour établir la communication. Des deux côtés, ces classes sont facilement accessible via l'annotation `@Autowired`. Lors de l'instanciation, une vérification du bon état des magasins est effectuée.

Gestion des magasins du serveur

Coté serveur, la communication a besoin de 3 magasins pour fonctionner :

- Key : magasin contenant la clé privée du serveur et le certificat de celui-ci.
- Trust : magasin contenant le certificat du serveur. A partir de ce certificat, le système est capable de vérifier que c'est bien ce serveur qui à signer les certificats clients.
- Crl : magasin contenant les numéros de séries des certificats clients qui ont été révoqué. Il sert à vérifier que le certificat qu'un client présente n'a pas été interdit.

A l'initialisation, si l'un des magasins est invalide, il est recréé automatiquement. Si cette création automatique est effectuée, l'utilisateur en est averti. La signature des certificats du proxy se fait via une application externe. Cette application réutilise la classe gérant les magasins pour effectuer la signature.

Gestion des magasins du proxy

Coté proxy, seulement 2 magasins sont nécessaire :

- Key : magasin contenant la clé privée du proxy et le certificat signé de celui-ci.
- Trust : magasin contenant le certificat du serveur afin de pouvoir vérifier que le proxy communique bien avec le bon serveur.

A l'initialisation, si l'un des magasins est invalide, il est recréé automatiquement. Si le certificat du serveur est manquant, une boite de dialogue demande à l'utilisateur de lui indiquer où se trouve le certificat du serveur. Si le certificat du client n'est pas signer par le certificat du serveur, une boite de dialogue demande ou enregistrer la demande de signature. Puis une fois cela fait, une autre boite de dialogue demande alors le certificat signée. Ces interactions avec l'utilisateur permettent de facilement configurer les magasins du proxy.

Le transfert entre le proxy et le serveur des certificats n'est pas géré par l'implémentation. C'est donc à l'utilisateur de s'assurer qu'il récupère bien les certificats de la bonne personne et non des faux.

5.9.2 Etablissement des communications

Si la configuration des différents magasins coté proxy et serveur pouvait sembler complexe, notamment du fait des différents éléments cryptographique mis en jeu, l'établissement des communications fut bien plus complexe. Déboguer l'établissement de connexions crypter n'est pas le plus simple. Après plusieurs heures (pour ne pas dire jours) à analyser les logs et à utiliser le débogueur, j'ai finalement trouvé une

solution fonctionnel qui après refactoring [13] s'avère assez simple dans la mise en oeuvre. L'argument Java `-Djavax.net.debug=all` s'est d'ailleurs révélé être très pratique pour vérifier le bon fonctionnement des communications, malgré le fait que cela affiche vraiment beaucoup d'informations dans la console.

Coté serveur la configuration fut bonne dès le début. Lors de la construction du servlet servant à attendre l'arrivée de connexion, il est facilement possible de lui fournir la configuration du socket SSL via la classe `SSLSocketFactory`.

Coté proxy, les choses sont plus complexes, il n'est pas possible d'utiliser les moyens classiques mis à disposition par Spring puisque ceux-ci ne permettent pas de configurer simplement la méthode d'établissement de la connexion. Les différents exemples visible sur internet utilisant une connexion SSL combinée à un nom d'utilisateur et un mot de passe. La solution trouvée consiste à modifier l'invocateur des appels à distance, pour que celui-ci utilise la configuration voulu. Après analyse du code source de Spring de l'invocateur utilisé jusqu'à présent, il est apparu qu'au moment de l'instantiation de celui-ci, il était possible d'accéder à l'objet `HttpClient`. Permettant de remonter jusqu'au manager de connexion et enfin jusqu'au gestionnaire de protocole utilisé pour l'invocation des appels à distance. A partir de là, la suite fut assez simple et reprend une partie du travail effectué coté serveur. On configure le contexte SSL à utiliser pour le protocole `swaphttps` via le gestionnaire de protocol dont on vient juste d'accéder. Il ne reste plus qu'à changer le protocol définit dans l'url utilisée pour les appels à distance en `swaphttps`. On test et ça marche.

13. Le refactoring est le fait de changer l'organisation d'un code de façon à le rendre plus clair et maintenable.

Conclusion

Ce projet de fin d'étude fut l'occasion idéale pour la mise en pratique de l'ensemble des compétences acquises au cours de mon cursus. Recherche, prototypage, spécifications, architecture logiciel, méthodes de programmation, toute ces éléments décrivent les grandes lignes de ce projet. La recherche d'une solution permettant de gérer les appels à distance fut un réel défi. Au vu du résultat final, je pense pouvoir dire sans me tromper que l'objectif initial a été atteint et qu'il en est de même des objectifs secondaires.

Au cours de ce projet j'ai pu découvrir de nouvelles méthodes de programmation avec notamment le principe d'inversion de contrôle mis en avant par le framework Spring. Mes compétences JavaScript et Java se sont fortement améliorer tout au long des étapes de développement.

Je tiens à remercier mon encadrant Sébastien Aupetit pour son implication dans le projet, ses conseils toujours avisés et l'opportunité qu'il m'a offert de co-écriture deux articles liées aux recherches effectués dans le cadre de ce PFE.

Annotation Tool for the Smart Web Accessibility Platform

Sébastien Aupetit and Vincent Rouillé

Université François Rabelais Tours, Laboratoire Informatique (EA6300) 64, avenue Jean Portalis, 37200 Tours, France
aupetit@univ-tours.fr

Abstract. Active and passive accessibility are two manner to improve web accessibility. While active accessibility mostly relies on norms and recommendations, it is practically proved that it is not sufficient. Passive accessibility is achieved by *a posteriori* content transformations. The Smart Web Accessibility Platform (SWAP) is a set of open source tools designed to tackle the passive accessibility problem of web contents. This article presents the goals and aims of SWAP through its main components: the proxy, the server and the annotation tool. The annotation tool is built using the proxy of SWAP. We explain how such design allows the annotation tool to be maintainable, independent of the browser and very flexible compared to other design.

Keywords: Web accessibility, Annotation tool, Proxy, Smart Web Accessibility Platform, Web page transformation

1 Introduction

Web accessibility means that people can access contents of web pages whichever disabilities they suffer (aging, impairment...). Active and passive web accessibility are two ways to achieve such universal access. Active accessibility relies on norms, recommendations[1] and laws [1] to enforce a proper structuring and tagging of documents during their creation by webmasters. Tools[2] and methodologies [2] complete the set to easy achievement. Besides lobbies strongly encourage on active web accessibility, it is not sufficient. A great part of the web remains not accessible for many reasons: "it costs too much", "constraints are too high", lots of web sites are not maintained anymore, lots of web sites are not professional works, web sites evolve, enforcing laws are country specific and have various requirements... A fully accessible web is not for tomorrow. Passive accessibility is a complementary approach which can handle those flaws. It mainly consists in using assistive technologies (zoom, speech synthesis...) or in transforming web pages contents *a posteriori* when pages are already available

[1] http://www.w3.org/TR/WCAG10/, http://www.w3.org/TR/WCAG20/

[2] http://achecker.ca/checker, http://www.binaryblue.com.au/access_wizard...

on the Internet . Web pages transformation is what we are concerned here. Many tools[3] and authors tried to handle the problem [3,4,5,6,7,8,9,10,11]. Those tools can be categorized by the location where the transformation occurs: on the web site, on a dedicated web site or on the user computer. Each location has pros and cons. On the web site or on a dedicated server, specific data can be used (database, templates...) but the server must be stronger to handle more tasks, webmasters must provide all what is necessary for the transformation and accessibility features can not be personalized for a user's disabilities. On the user computer, the tool can do specific transformations for the user and webmaster help is not required (if he cooperates, it is better). The main concern is that the tool does neither have access to the data used, nor to the templates to built the page. On user computer, the tool can be a specific process or an extension of a browser (plugins). Plugins have access to all the features of the browser but are highly dependent of the browser, of its evolution, of its support and of how web technologies advances through time. Dedicated processes are usually proxy software. The main advantage is that a proxy does neither enforce the user to a specific browser, nor to change his current assistive technologies. Moreover, the proxy evolution is independent of the browser and its evolution. The only requirement is that access to the web is made through the proxy. The main disadvantage is that some processing are done by both the proxy and the browser (e.g. document parsing and analysis).

The Smart Web Accessibility Platform (SWAP)[4] developed by the computer science laboratory of Université François Rabelais Tours aims to provide tools for passive accessibility using a client side proxy transformation approach. SWAP is an open source modular and extensible set of tools. Previous works on web accessibility demonstrated how SWAP can be used for textual color improvement [10,11,12,13]. To develop smarter transformations of the contents, we are faced with the problem of annotating contents for pattern recognition and automatic assessments of the results. A flexible annotation tool is required. In the following, we discuss the SWAP platform and demonstrate why and how the annotation tool can be defined as a specific component of the platform.

2 Smart Web Accessibility Platform for Passive Accessibility

SWAP is both a research tool, an experimentation tool and a final user tool. SWAP is built to be portable, easily extensible and modular allowing the assembling of any custom applications. It is written in Java using components, a design patterns approach and the Spring framework[5] for the Inversion of Con-

[3] `http://www.bbc.co.uk/education/betsie`, `http://apache.webthing.com/mod_accessibility/`, `http://muffin.doit.org/`, `http://rabbit-proxy.sourceforge.net/`...

[4] `https://projectsforge.org/projects/swap`

[5] `http://www.springsource.org/`

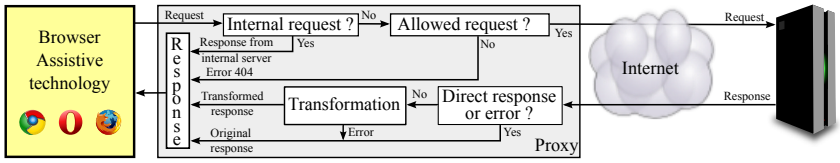

Fig. 1. Request processing by the proxy.

trol (IoC) part. Maven[6] is used to manage dependencies and to create specific custom assembly of components. The SWAP platform is currently composed of three high level applications: the proxy, the server and the annotation tool. Each application relies on a set of common components (the core) and specific components. A Java Virtual Machine (JVM) can be bundled with the applications removing the need of administrator privileges on a computer to use the applications. Consequently, the tools can be used even on very software standardized or constrained computers (bank...). The one and only constraint is that accessing the web is made through the proxy by both the browser and the assistive technologies.

2.1 The Core Components

The core components provide a simplified self expressive management of the components, of the dependencies (IoC) and of the database. This is mostly achieved using Java annotations with Spring, Hibernate[7] and Java Persistence API (JPA)[8] using Spring JPA. Core components provide HTTP protocol support, request and response caching, HTML Document Object Model (DOM) facilities and Cascading Style Sheets (CSS) facilities.

2.2 The Proxy Tool

The proxy tool is composed of the core components and of specific ones. The main specific component is the one handling the communication with the browser. It is realized using the Jetty web server[9]. Depending on which domain is accessed, the component does either the proxying of the request or forwards the request to the internal HTTP server (see Fig. 1). The internal HTTP server allows either the communication between a page in the browser and the proxy using Javascript (see section 3), or the configuration of the tool. This internal server is managed by the Web MVC (Model View Controller) component of Spring. To proxy a request, the component forwards the request to the web site. The obtained response can then be transformed if needed. The original or transformed response is finally sent to the browser. Since the proxy tool is executed on the

[6] `http://maven.apache.org/`
[7] `http://hibernate.org/`
[8] `https://www.jcp.org/en/jsr/detail?id=317`
[9] `http://www.eclipse.org/jetty/`

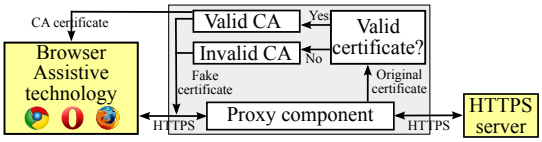

Fig. 2. *Man-in-the-middle* approach used for secured exchanges.

user computer, HTTPS requests (secured contents) can also be transformed. To not compromise security, a *man-in-the-middle* approach (see Fig 2) is used. Two internal Certification Authorities (CA) are used (self signed CA). The valid CA is used to create fake certificates for servers having valid certificates. The invalid CA is used in the other cases. The CA certificate of the valid CA is exported by SWAP and imported by the browser. Consequently, any certificate emitted by the valid CA is recognized as a valid certificate by the browser. Security is maintained while secured contents can be transformed if needed.

Multiple transformations can be applied on a web page. The proxy handles the proper parallel execution of all the transforming components using read/write locks and components synchronization. Transformations can be done at low level (bytes) or at high level (DOM). [10,11,12,13] shows the use of specific transformation components with the proxy tool. The annotation tool is another example of the use of the proxy tool with specific components (see section 3).

2.3 The Server Tool

The server tool includes the core components. One specific component allow a simple communication protocol between the proxy tool and the server tool (Remote Procedure Call). Another component handle database storage. For now, the server tool is only used for global knowledge storage. It is planned to use the server to do some learning calculus when pattern recognition will be used.

3 The Annotation Tool

The annotation tool is the proxy tool for which specific components are added (see below). The tool aims to collect data on web pages and to store them on the server as global knowledge.

3.1 The Problem

To create smart transformations, it becomes necessary to have a database of knowledge on web pages samples to tackle web variability. Such knowledge can be gathered using experts (paid service, volunteers...) and a dedicated annotation tool. The knowledge database allows the use of machine learning, the use of pattern recognition methods or automated validations of transformations.

3.2 Which Form for the Annotation Tool?

The annotation tool can take many forms such as a dedicated software, a plugins for a browser or a set of components for SWAP. By integrating a web browser into a dedicated software, it is possible to have access to all features of the browser almost without limits (internal representation, interaction...). However, browsers and technologies evolve rapidly increasing the risk on durability and maintainability of the tool. Browser plugins have similar pros and cons as the dedicated software. They are highly dependent on the API (Application Programming Interface) of the browser. The plugins API are browser specific and evolve rarely with an ascending compatibility. In both cases, the integration of the annotation tool with the page requires the interactive parts (selection, marking...) to be linked to the DOM of the page. A simpler way to integrate the interactive part can be done using injection of scripts in the page like a transformation using SWAP. In this case, the annotation tool is independent of the browser, of the browser API and of the evolution of the browser. Moreover, as being integrated in the page, the scripts have access to all features of the pages without limits. The main drawback is that the injection can lead eventually to a breakage of the page structure, of the aspect or of the behavior. A proper handling of the isolation of the DOM, of the styles and of the scripts of the annotation tool and of the page is necessary.

3.3 Isolation

The Document Object Model (DOM) is a hierarchical representation of the elements of a web page. Any element must be in the DOM to be visible. Javascript allow to manipulate and interact with the DOM to do things. There is not permissions for accessing the DOM: any script can modify the DOM. Consequently, the scripts of the page and the injected scripts both have access to the same content. Moreover, when an unexpected element is inserted in the DOM, it can lead to lots of consequences such as script errors or styles breakage. To reduce the consequence of the annotation tool on the page, scripts isolation, DOM isolation and styles isolation must be done. We identified three ways to isolate the DOM (see Fig. 3).

One way to isolate the DOM consists in creating a specific page containing all what is necessary for the annotations (scripts, CSS) and in embedding an `iframe` containing the web page to annotate. The contained DOM is isolated from the containing DOM by the `iframe` but the scripts of the containing page can access the content of the annotated web page. This approach seems simple but to interact with the annotated page, a complex Javascript API must be used and some features of the browsers can conflict with the annotation process. For example, popup or windows opening create new tab/window which does not contain the annotation tool. As a solution, we can consider that an injected script checks for the presence of the annotation tool and, if not, reload the annotation tool with the page as a parameter. This is not usable since the reloading of a page does not guarantee to obtain an identical page. It is even less the case

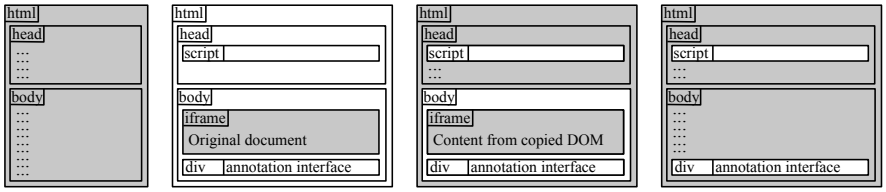

Fig. 3. The three ways to isolate the DOM. Parts of the original document are represented in grey. (a) Original DOM, (b) Specific `iframe` isolation, (c) The DOM is moved as a child of an `iframe`, (d) A root node is added at the end.

when POST request are used. Then this approach is not appropriated for the annotation tool.

The second way to isolate is similar to the previous but without a dedicated web page for the annotation. A script is injected in every page passing by the proxy. On page loading, the script copies the DOM of the document, replaces the current DOM by a simpler one containing an `iframe` and puts the copied DOM in the `iframe`. In this case, the DOM of the annotated page is isolated from the annotation tool and browser features does not interfere with the tool. This approach seems miraculous but it suffers major drawbacks. An important latency is introduced for the annotation tool since the complete DOM needs to be loaded before being copied and replaced. Since the complete DOM of the page is loaded, the embedded scripts of the page can not be unloaded or blocked leading in frequent cases to infinite loops or errors. Moreover, ours tests with recent browsers showed important stability and performance issues. This approach is appropriated only for simple static contents.

The third and also the simplest way to isolate consists in adding an element to the DOM. This element is used as a root node for any element added to the page for the annotation tool. It is simple and browser performances are preserved. The main drawbacks is that the DOM is modified. By regrouping the changes under an unique root node at the end of the document, we experimentally devised that it provokes no breakage on lots of page. Theoretically, it can lead to breakage of scripts or of styles but it is very infrequent. This approach is the most suited for the injection of the annotation tool in the page. To limit the influence of the page on the tool, styles isolation and scripts isolation must be performed.

Styles isolation is performed by marking the root element of the annotation tool with an unique class identifier and by resetting the style properties for any child element. The properties are marked as important to take precedence over any defined style in the annotated web page. Element properties evolve with standards and can sometimes be specific to the used browser. To increase maintainability and durability of the tool, we defined a script which compare the styles of the element with a neutral element outside the DOM (then not affected by styles of the page). If properties differ then they are reset to the neutral values. Performance degradation introduced by this script is insignificant.

Scripts isolation is achieved through methodology. The scripts are injected at the highest location in the DOM (first child of `head`). It implies that ours scripts are the first to be executed in the page, whichever the page's scripts, when the Javascript execution environment is clean. To eliminate any conflict between the annotation tool scripts and the page's scripts, ours scripts never create or modify global variables. Scripts are made anonymous and not callable by others scripts using anonymous function. Those functions and theirs local variables are not visible to the other scripts. We use: `(function() {/* isolated code */})();`.

3.4 Annotation Tool Components

SWAP is designed to be modular. The scripts of the annotation tool follows the same principles. The annotation tool is mainly composed of Javascript scripts designed like components having dependencies. The main component is the `Loader`. It provides an API for the components allowing to describe their names, their dependencies and theirs implementations. The loader is the first executed script on the page. It loads the required components using the isolation principle described above and the described dependencies. Many components are actually implemented. The *Communication* component defines an API simplifying the exchange between the page and the internal web server of the proxy. It relies on jQuery[10] to be browser neutral. The *EventEmitter* component implements the Observer pattern (subscription and emission of signals) to simplify the communication between the isolated components and to handle events. In the current implementation, those components are completed with the *Zone* component whose aim is to allow the marking, the identification and the annotation of zones in the page. It handles the transmissions of information on the zones to the proxy (type, anonymized DOM, Xpath...). The proxy forwards the information to the server for storage in the knowledge database. Figure 4 presents a screenshot of the annotation tool with the `Zone` component.

4 Conclusion

In this work, we extend the modular platform SWAP with components dedicated to the creation of an annotation tool. We discussed how it can be implemented using the proxy tool of the project and how the isolation problem can be resolved efficiently. The modular feature of SWAP is applied even for the Javascript code providing a flexible annotation tool. An example of specific components for zones annotation is presented but the tool can easily be extended with others kind of annotations by creating new components. The isolation principle and infrastructure are used for the annotation tool. They can also be reused for the injection of scripts in web page, for example, for nagivation assistance or for any interactive assistive task.

[10] `http://jquery.com/`

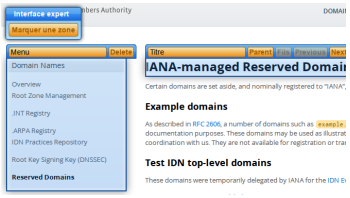

Fig. 4. A screenshot of the annotation tool on a web page with the `Zone` component.

References

1. République Française: Loi n°2005-102 du 11 février 2005 pour l'égalité des droits et des chances, la participation et la citoyenneté des personnes handicapées. JO n° 36 du 12 février 2005, p2353 (2005)
2. Colas, S., Monmarché, N., Burger, D., Mohamed, S.: A web site migration support tool to reach european accessibility standards. In: 9th European conference for the Advancement of Assistive Technology in Europe (AAATE'07). Volume 20., San Sebastian (Spain) (october 2007) 907–911
3. Brown, S.S., Robinson, P.: A world wide web mediator for users with low vision. In: Proceedings of Conference on Human Factors in Computing Systems (CHI 2001 Workshop No. 14), Seattle, WA (2001)
4. Colajanni, M., Grieco, R., Malandrino, D., Mazzoni, F., Scarano, V.: A scalable framework for the support of advanced edge services. In: High Performance Computing and Communications. Volume 3726. Springer (2005) 1033–1042
5. Parmanto, B., Ferrydiansyah, R., Zeng, X., Saptono, A., Sugiantara, I.: Accessibility transformation gateway. In: System Sciences, 2005. HICSS '05. Proceedings of the 38th Annual Hawaii International Conference on, IEEE (2005) 183a–183a
6. Han, R., Bhagwat, P., LaMaire, R., Mummert, T., Perret, V., Rubas, J.: Dynamic adaptation in an image transcoding proxy for mobile web browsing. IEEE Personal Communications **5**(6) (December 1998) 8–17
7. Gupta, S., Kaiser, G.: Extracting content from accessible web pages. In: International Cross-Disciplinary Workshop on WebAccessibility, ACM Press (2005) 26
8. Gupta, S., Kaiser, G.E., Grimm, P., Chiang, M.F., Starren, J.: Automating content extraction of HTML documents. World Wide Web **8**(2) (June 2005) 179–224
9. Hermsdorf, D., Gappa, H., Pieper, M.: WebAdapter: A prototype of a WWW-browser with new special needs adaptations. In: Proceedings of the International Conference on Computers Helping People with Special Needs. (1998) 151–160
10. Mereuţă, A., Aupetit, S., Slimane, M.: Improving web accessibility for dichromat users through contrast preservation. In: Computers Helping People with Special Needs. Volume 7382., Springer (2012) 363–370
11. Aupetit, S., Mereuţă, A., Slimane, M.: Automatic color improvement of web pages with time limited operators. In: Computers Helping People with Special Needs. Volume 7382., Springer (2012) 355–362
12. Mereuta, A., Aupetit, S., Monmarché, N., Slimane, M.: Web page textual color contrast compensation for CVD users using optimization methods. Journal of Mathematical Modelling and Algorithms in Operations Research (October 2013)
13. Mereuta, A., Aupetit, S., Monmarché, N., Slimane, M.: An evolutionary approach to contrast compensation for dichromat users. In: Proceedings of Evolution Artificielle 2013, Lecture Notes in Computer Science, Springer (2013) to appear.

POLYTECH TOURS
Département Informatique

Outil d'annotations pour le projet Smart Web Accessibility Platform

Vincent Rouillé et Sébastien Aupetit
Université François Rabelais Tours, Laboratoire Informatique (EA6300)
Equipe Handicap et Nouvelles Technologies
64, avenue Jean Portalis, 37200 Tours, France
aupetit@univ-tours.fr

***Résumé*—L'accessibilité du web peut être atteinte de deux façons. De façon active principalement grâce à l'utilisation de normes et de recommandations. De façon passive en transformant *a posteriori* les contenus pour les rendre plus accessible. La plateforme SWAP (Smart Web Accessibility Platform) s'intègre directement dans la démarche d'accessibilité passive. Dans ces travaux, nous présentons globalement le fonctionnement et l'intérêt de SWAP pour l'accessibilité passive grâce à trois composants de hauts niveaux : le proxy, le serveur et l'interface d'annotations. Cette interface est réalisée en utilisant directement les capacités de proxy de la plateforme SWAP. Ce mode de fonctionnement opposé à l'approche classique (outil spécifique, plugins navigateur...) permet de s'affranchir presque intégralement de l'évolution frénétique des navigateurs et des technologies associées. Cette indépendance induit inévitablement des contraintes fortes sur la réalisation techniques. Nous montrons qu'il est possible de s'en affranchir totalement en faisant des choix judicieux. De plus, grâce à l'approche modulaire suivie par le projet, l'outil d'annotations peut être facilement étendu en fonctionnalités tout comme le reste de la plateforme SWAP.**

Mots clés*—*Accessibilité web, Outil d'annotations, Proxy, Smart Web Accessbility Platform, Transformation de pages web

I. Introduction

De nombreux sites web sont peu ou pas accessibles ou utilisables pour des personnes souffrants d'un handicap (visuel par exemple) ou tout simplement vieillissantes. Avec l'explosion d'Internet, de ses usages et de l'évolution des technologies, accéder à l'information pour certaines personnes devient de plus en plus complexe voire impossible. Dans ce cas, au lieu de contribuer à l'inclusion dans la société moderne, Internet devient un facteur d'exclusion. Pour limiter voire éliminer cette exclusion, deux démarches complémentaires peuvent être suivie : l'accessibilité active et l'accessibilité passive.

L'accessibilité active consiste à appliquer de façon proactive des principes et des normes, telles que les WCAG [1], [2], dès la conception des sites web mais aussi pendant toute la vie des sites web. Ces principes et normes ont été établis par plusieurs associations et lobbies. Des lois [3] ont ensuite été promulguées afin de forcer leurs applications et le bon usage des technologies associées dans certains cas particulier. L'ensemble a été complété rapidement par la création de validateurs [4], [5] et par des accompagnements de webmasters tels que [6]. Malgré tous ces dispositifs et les bonnes volontés, force est de constater qu'une proportion très importante du web reste inaccessible. De nombreuses raisons sont souvent évoquées pour justifier cet état de fait : « trop coûteux en temps et en argent », « trop complexe à appliquer », « trop contraignant », beaucoup de sites web ne sont plus maintenus mais existent toujours, d'autres sont faits « à la main » et beaucoup évoluent dans le temps... De plus, les normes et les lois « accessibilité » diffèrent souvent d'un pays à un autre (actions, champs d'application...) lorsqu'elles existent mais le web est international. Cette approche active de l'accessibilité s'appuyant fortement sur les webmasters est essentielle mais insuffisante : un web 100% accessible restera une utopie pendant encore de nombreuses années.

Une autre approche complémentaire est l'accessibilité passive. Dans ce cas, la mise en accessibilité des sites web s'effectue *a posteriori* après la mise en ligne. Elle prend la forme d'outils transformant les pages web de manière à les améliorer ou par l'utilisation d'aides techniques spécialisées (zoom, synthèse vocale...). Dans ces travaux, nous nous focalisons sur les outils de transformation [7]–[17]. Ces outils peuvent être catégorisés par le lieu où s'effectue la transformation : sur le serveur hébergeant les pages web, sur un serveur dédié ou sur la machine de l'utilisateur. Chaque lieu possède des inconvénients et des avantages. En étant situé sur un serveur, l'outil peut avoir accès à des informations spécifiques (base de données, *template*...) mais il nécessite des capacités de calcul supplémentaires, une aide et une vigilance permanente des webmasters, et il est peut flexible par rapport au besoin de l'utilisateur. En étant situé sur la machine de l'utilisateur, l'outil est capable de faire du spécifique pour l'utilisateur et la coopération des webmasters n'est pas requise (c'est mieux s'il coopère). Le principal inconvénient provient du fait qu'il n'est pas possible dans ce cas d'avoir accès aux données brutes, annotées ou à des *templates* de mise en page. Sur la machine de l'utilisateur, l'outil peut être un processus séparé du navigateur ou intégré à celui-ci. Lorsqu'il est intégré au sein du navigateur (*plugins*), l'outil a accès à toutes les capacités du navigateur cependant il contraint l'utilisateur à un navigateur particulier. De plus, le rythme des évolutions techniques et technologiques des navigateurs peut être un obstacle à la pérennité de l'outil. Les outils en processus dédié prennent couramment la forme de proxy s'exécutant sur la machine de l'utilisateur. Le principal avantage d'un proxy est qu'il ne contraint nullement l'utilisateur à tel ou tel navigateur ou aide technique : la seule contrainte est que l'accès au web s'effectue à travers le proxy. De plus, l'évolution du proxy est indépendante du rythme d'évolution du navigateur ou des aides techniques. Le principal inconvénient de cette approche est qu'un certain nombre de traitements sont effectués en double (navigateur+proxy) tels que l'analyse du document...

Le projet Smart Web Accessibility Platform (SWAP) [1], développé par le Laboratoire d'Informatique de l'Université François Rabelais de Tours, est une parfaite matérialisation du principe de l'accessibilité passive par transformation des contenus via un proxy. Le projet SWAP a pour objectif principal de proposer un ensemble d'outils *open source* permettant de réaliser de façon modulaire et extensible la transformation de pages web à la volée en utilisant entre autre un proxy *coté client*. Nos travaux précédents [18]–[23] ont montré l'utilisation du proxy pour la correction des couleurs des textes. Afin de proposer de nouvelles transformations « intelligentes » capables de gérer la grande diversité des contenus web via des techniques d'apprentissage automatique ou semi-automatique, il nous est devenu nécessaire de pouvoir annoter les pages web. Dans la suite, nous présentons l'architecture technique du projet SWAP puis nous décrirons les problématiques liées à la création d'un outil d'annotations pour le projet SWAP puis sa réalisation.

II. SWAP : une plateforme logicielle générique au service du handicap

En tant qu'outil de recherche, d'expérimentation et d'utilisation finale, la plateforme SWAP a été conçue dès le début de manière à être portable, facilement extensible et assemblée en fonction des besoins. Pour cela, elle a été écrite en langage Java sous la forme de composants indépendants et interchangeables grâce à l'utilisation de design patterns et du framework Spring [24] pour l'inversion de contrôle (Inversion of Control, IoC). L'assemblage des composants et de leurs dépendances en une application « sur mesure » est réalisé grâce à l'outil Maven [25]. Ce mode de conception et d'assemblage permet de maximiser la réutilisation des composants logiciels. Trois applications finales composent la plateforme SWAP : le proxy, le serveur et l'interface expert. Chacune s'appuie sur un jeu de composants standards nommé le « coeur ».

Un assemblage de composants pour le proxy peut être copié avec une machine virtuelle (Java Virtual Machine, JVM) [2] afin d'obtenir une application ne nécessitant aucune installation préalable, ni droits administrateur. Ainsi, aucune contrainte d'environnement (poste sécurisé, standardisé…) ne devrait empêcher son utilisation. La seule et unique contrainte est que le navigateur ou les aides techniques accèdent au web via le proxy.

A. Le coeur

Les composants du coeur permettent une gestion simplifiée et auto expressive des composants et de la base de données. Ceci est obtenue grâce à une phase d'autodétection des composants et grâce à l'utilisation extensive d'interfaces et d'annotations Java permettant de piloter les bibliothèques Spring [24], Hibernate [26] et Java Persistence API (JPA) [27] via Spring JPA. Une série de composants standards sont intégrés au coeur. Ils permettent entre autres la gestion des requêtes HTTP et la mise en cache des réponses, l'analyse de documents HTML en un DOM (Document Object Model), l'analyse des feuilles de styles CSS et l'application des CSS à un DOM.

1. https://projectsforge.org/projects/swap
2. Testé sur GNU/Linux et Windows avec des machines virtuelles Oracle Java 7.

B. Le proxy

Le proxy inclut l'intégralité des composants du coeur et d'autres composants spécifiques. Le principal composant est le composant gérant la communication avec le navigateur. Il est réalisé grâce à l'utilisation du serveur web Jetty [28]. En fonction des demandes, Jetty aiguille les requêtes (Fig. 1) soit vers le composant proxy, soit vers un serveur HTTP interne au proxy. Les requêtes vers le serveur HTTP interne sont gérées via le modèle WebMVC de Spring. Ce serveur HTTP interne est utilisé pour : (1) la communication entre le navigateur et le logiciel proxy et (2) comme interface web de configuration et d'information du logiciel. L'outil d'annotations décrit par la section III est une parfaite illustration du mécanisme de communication entre le navigateur et le logiciel proxy.

Du fait de sa localisation sur la machine de l'utilisateur, le logiciel proxy est capable de transformer les communications sécurisées en HTTPS sans remettre en cause la sécurité. Pour cela, nous utilisons une approche du type *man-in-the-middle* (Fig. 2). Lorsque le proxy reçoit une requête HTTPS, il contacte le site web afin d'établir une communication HTTPS et obtenir le certificat du serveur. Ce certificat peut être utilisé pour déchiffrer les échanges mais il ne peut pas l'être pour rechiffrer les échanges entre le proxy et le navigateur. Pour résoudre ce problème, le proxy émet un nouveau certificat fictif et l'utilise pour échanger avec le navigateur. Ce certificat fictif reprend les caractéristiques du certificat original. Il peut être émis par l'une des deux autorités de certification (Certification Authority, CA) incluses dans le logiciel proxy. La première est utilisée pour créer des certificats pour les domaines valides tandis que l'autre est utilisée pour les certificats non valides (auto signé, chaîne de validation non reconnue…). Le certificat racine du CA créant les certificats valides doit être importé dans le navigateur afin que celui-ci reconnaisse comme valide tout certificat émis par la CA valide. Ainsi, il est possible de maintenir à la fois la sécurité des échanges et la capacité de transformer les pages web sécurisées.

La transformation d'une page web consiste à appliquer un certain nombre de composants « transformant » sur la page. Ces composants sont exécutés en parallèle de manière à exploiter les capacités des processeurs récents. Des mécanismes de synchronisation entre composants « transformant » et de gestion d'accès (verrous lecteur/rédacteur) sont utilisés pour orchestrer l'ensemble. La transformation d'un contenu peut actuellement s'effectuer à bas niveau (octet) ou à haut niveau (DOM HTML) après décodage du contenu, détection de l'encodage…

Sans assemblage particulier, le logiciel proxy ne sert que de passerelle vers l'extérieur. En ajoutant, une ou plusieurs transformations telles que celles de [18]–[23], il permet d'améliorer l'accessibilité des contenus web. En lui ajoutant les composants de l'interface expert, il devient un outil flexible d'annotation (cf. section III).

C. Le serveur

Le serveur inclut l'intégralité des composants du coeur et d'autres composants spécifiques. L'objectif principal du serveur est de stocker, constituer et manipuler les connaissances transversales utiles à l'ensemble des logiciels proxy. Les principaux composants spécifiques mettent en place des

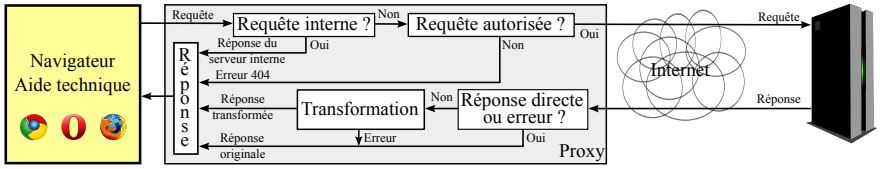

FIGURE 1. Traitement des requêtes au sein du proxy.

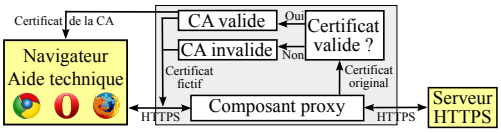

FIGURE 2. Approche *man-in-the-middle* pour la transformation des communications sécurisées.

facilités de communications du type RPC (Remote Procedure Call) sur HTTP et de gestion de la base de données « serveur ». Dans son état actuel, le serveur n'est utilisée que comme base de données communes aux logiciels proxy et à l'interface experte. Les données qui y sont stockées sont anonymisées autant que possible.

D. L'interface expert

L'interface expert correspond à un assemblage spécifique incluant intégralement la base du logiciel proxy. Les composants spécifiques et les problématiques qui y sont liées sont décrits à la section III. L'interface expert a pour vocation la collecte d'informations stockées sur le serveur afin que les proxy puissent les utiliser avec ou sans prétraitement.

III. LE MODULE D'ANNOTATIONS DES PAGES WEB

A. Problématique

Pour permettre la conception de transformations évoluées améliorant l'accessibilité des pages web, il devient rapidement nécessaire de pouvoir disposer d'une base de connaissances sur des pages web exemples. Pour obtenir ces connaissances, il est nécessaire de recourir à des utilisateurs « expert » (volontaires, services payants…) annotant des pages web avec les informations recherchées.

Lorsqu'une telle base existe, plusieurs utilisations deviennent possibles. Entre autres, il devient envisageable d'utiliser des techniques de reconnaissances de formes, d'automatiser le test de nouvelles transformations ou tout simplement de « corriger » des pages de façon fiable. La conception de transformations intelligentes capables de gérer les variabilités du web autrement qu'avec des règles absolues ouvre le champs à de nombreuses améliorations. La validation automatique de l'efficacité de telles ou telles transformations permet une accélération non négligeable des capacités d'innovation et d'expérimentation de nouvelles méthodes de transformation. La correction de pages web pallie aux faiblesses des transformations face à l'hétérogénéité du web.

B. Réalisation d'une interface d'annotations

Il existe plusieurs moyens permettant de réaliser l'interface d'annotations.

1) Logiciel dédié: La réalisation de cette interface peut se faire par la création d'un logiciel dédié intégrant un navigateur web. Le principal intérêt de cette approche est qu'il est possible d'accéder aux mécanismes internes de gestion, de représentation et d'interaction des pages web du navigateur sans aucune limite. Néanmoins, l'évolution constante des standards du web, l'implémentation des fonctions de navigation et la gestion de la sécurité nécessitent un suivi constant de l'évolution du navigateur. L'intégration d'un navigateur peut donc s'avérer être une tâche complexe, peu efficace et peu pérenne entraînant rapidement des problèmes de maintenabilité importants.

2) Extensions de navigateur: Une autre possibilité est la création d'extensions (plugins) pour un navigateur. Cette approche possède globalement les mêmes avantages et inconvénients que le logiciel dédié. Elle est très fortement dépendante du navigateur choisi et la compatibilité sur le long terme (API…) est directement liée à l'organisation en charge du développement du navigateur. De plus, les API permettant aux extensions de fonctionner sont non standards et interdisent le support de multiple navigateur.

3) Utilisation de la plateforme SWAP: La gestion de l'intégration de l'affichage de l'outil d'annotations (zones…) posent les mêmes problèmes quelque soit la solution retenue (logiciel dédié, plugins). Pour obtenir une intégration suffisante avec la page web à annoter, il est incontournable d'incorporer la partie interactive (sélection, zones…) directement au sein de la page (DOM…). Comme la page web doit être modifiée pour l'interaction, il devient intéressant d'utiliser l'infrastructure du proxy de SWAP afin d'insérer l'interface d'annotations directement dans les pages web. Le principal avantage est l'indépendance complète du projet par rapport aux navigateurs et à leurs évolutions. L'expert peut donc conserver ses habitudes et utiliser le navigateur auquel il est habitué (sous réserve qu'il s'agisse d'un navigateur récent). De plus, en étant intégré directement dans la page web, toutes les capacités du navigateur sont accessibles et l'intégration à la page web est maximale. La principale difficulté de cette approche est que l'interface peut interférer avec le reste de la page web. L'interface est injectée directement via le code source de la page visitée mais celle-ci possède probablement déjà des styles et du JavaScript. Le code injecté peut entraîner des incompatibilités au niveau du JavaScript et des styles. Pour résoudre ce problème, il est nécessaire d'isoler le code injecté par rapport à la page web. L'interface d'annotations est réalisable comme un ensemble de composants SWAP ordinaires en utilisant le proxy de la

plateforme SWAP.

C. Isolation

Pour comprendre les différents moyens d'isolation, il est nécessaire d'avoir une compréhension du DOM et de sa structure. Le DOM est un arbre d'objets créé pour chaque page web qui représente la structure de la page. Le DOM part d'un noeud racine qui représente la fenêtre du navigateur et qui s'étend jusqu'aux éléments HTML de la page web. Tout élément visible d'une page web est présent dans le DOM. Les différents scripts des pages web utilisent le DOM pour toutes les opérations qui visent à modifier l'affichage. Le DOM permet de structurer une page web et d'interagir avec les éléments de celle-ci. Comme il n'existe pas de gestion de droits d'accès sur les éléments du DOM, la page est modifiable à la fois par le script injecté et par les scripts de la page. L'ajout d'éléments dans le DOM peut avoir des conséquences importantes sur l'utilisabilité et l'aspect de la page. Pour injecter l'outil d'annotations, il est nécessaire d'isoler au maximum les interactions de la page web avec les éléments de l'interface d'annotations au niveau du DOM mais aussi au niveau des styles d'affichage.

Nous avons identifié trois façons principales d'isoler l'interface du DOM de la page web avec le mécanisme d'injection de contenus (Fig. 3). Du résultat de cette analyse découle l'isolation des styles et du JavaScript.

1) Isolation du DOM par une `iframe` avec URL externe: La balise `iframe` permet l'incorporer un document HTML (contenu) au sein d'un autre document HTML (contenant). Les interactions entre le contenant et le contenu sont fortement contraintes. Le contenant a son propre DOM indépendant et invisible du point de vue du DOM du contenu. La première façon d'assurer l'isolation du DOM consiste en la création d'un document HTML contant l'interface et une `iframe` contenant le document à annoter. Cette méthode garantie une isolation totale du document annoté par rapport à l'interface (DOM, styles et JavaScript). Cependant, cette organisation a de lourdes conséquences. L'implémentation des interactions entre l'interface et la page web visitée impose l'utilisation d'une API de communication JavaScript complexifiant le codage et la maintenance de l'outil. De plus, la gestion de l'`iframe` au sein de l'interface d'annotations entre plus ou moins en conflit avec des fonctionnalités du navigateur telles que l'ouverture dans une autre fenêtre/onglet ou la gestion des popups. Les interactions entre l'`iframe` et le navigateur peuvent provoquer un changement de la page web courante et ainsi faire disparaître l'interface d'annotations (popup sans interface d'annotations...). Une solution possible à cette disparition serait de détecter via la page web la présence où non de l'interface d'annotation. En cas d'absence, on recharge la page avec l'interface d'annotations via une URL particulière spécifiant l'URL de la page à annoter à inclure dans l'`iframe`. Le problème est que rien ne garantit qu'un second affichage de la même page web donne le même résultat notamment dans le cas de requête POST. De façon résiduelle, l'URL affichée par le navigateur correspond au contenant (l'interface d'annotations) plutôt qu'à celle de la page web visitée.

2) Isolation du DOM par une `iframe` avec DOM interne: Une solution alternative à l'isolation avec une `iframe` insensible au rechargement consiste à ajouter à toutes pages web un code JavaScript se déclenchant après la création complète du DOM de la page. Ce script copie le DOM de la page web, supprime tous les éléments du DOM de la page puis insère dans le DOM une `iframe` contenant le DOM copié (donc le DOM original). Cette approche permet de conserver l'isolation totale du DOM du contenu, de ne pas altérer les habitudes de navigation et enfin de garantir l'affichage en toute circonstance de l'interface d'annotation. Cette solution a, en théorie, tous les avantages, sans les inconvénients précédents. Elle a néanmoins des conséquences importantes. Une latence importante est introduite car il est nécessaire d'attendre que le DOM soit complètement disponible. Le DOM étant déjà créé, il n'est pas possible de décharger les scripts existants de la page web. Ceux-ci provoquent souvent des échecs ou des boucles pouvant aller jusqu'au plantage du navigateur. Le fait de créer une `iframe` et d'y insérer le contenu de la page induit une nouvelle analyse complète du contenu ce qui augmente à nouveau la latence ainsi que des problèmes de performances et de stabilité du navigateur. Cette solution n'est adaptée qu'à de simples pages statiques, ce qui est de nos jours est de plus en plus rare.

3) Isolation du DOM par ajout d'un noeud: La dernière solution que nous avons considéré est également la plus simple. Elle consiste à ajouter un élément dans le DOM de la page web. Cette élément sert de noeud racine pour tous les autres éléments de l'interface d'annotations. Les principaux avantages sont une grande simplicité de mise en place, une faible complexité de gestion des éléments et un impact quasi nul sur les performances de navigation. L'inconvénient majeur est que le DOM de la page web est modifié. Dans nos expérimentation, aucun site web ne s'est montré incompatible avec cette méthode. Néanmoins, il est théoriquement possible qu'une page web cesse de fonctionner ou de s'afficher correctement du fait de cette modification. Pour limiter les risques, nous insérons l'élément à la fin du DOM.

Au regard des différents avantages et inconvénients des différentes approches, nous avons considéré que cette méthode était la plus appropriée. Pour que celle-ci soit viable, il est nécessaire de réaliser une isolation des styles et des scripts de l'interface afin que ceux-ci ne soient pas influencés par ceux du document original.

4) Isolation des styles: La gestion des styles CSS est fortement lié à la forme du DOM. Afin que les éléments de l'interface insérée dans la page aient la même apparence sur toutes les pages web, les règles CSS à appliquer doivent être créées selon une méthode particulière.

L'approche la plus évidente consiste à marquer l'élément parent contenant l'interface par une classe CSS « unique ». A partir de cette classe, des règles ne s'appliquant qu'aux éléments de l'interface d'annotations peuvent être créées. Afin d'éviter tout problème de priorité entre les règles, toutes les propriétés sont marquées comme importantes. La définition de toutes ces règles devient très rapidement complexe à moins de restreindre les éléments HTML utilisables. De plus, les propriétés CSS à définir sont nombreuses, varient en fonction des navigateurs et des évolutions des standards. Maintenir ces styles devient rapidement un problème.

Une solution consiste à appliquer les styles CSS aux éléments de l'interface via du code JavaScript. Pour cela, l'en-

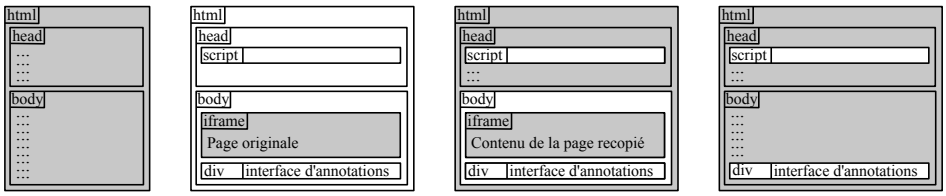

FIGURE 3. Les trois manières d'isoler le DOM. (a) DOM original (en gris), (b) Isolation par une `iframe` avec URL externe, (c) Isolation par une `iframe` avec DOM interne, (d) Isolation par ajout d'un noeud.

semble des propriétés CSS qui sont appliquées à un élément sont comparées avec un élément du même type non modifié. Toute propriété qui diffère est alors changée par la valeur définie par l'élément de référence. Pour obtenir cet élément de référence qui n'est autre qu'un simple élément d'une page web sans CSS, une `iframe` non visible est créée et un élément du même type est crée. C'est cet élément qui sert de référence. Cette approche induit un léger impact sur les performances de création de l'interface, mais le gain en terme de simplicité, de sécurité et d'évolutivité prévaut largement.

5) Isolation du Javascript: Il reste à isoler le code JavaScript injecté de celui de la page. Le proxy de l'interface d'annotations est conçu pour insérer du JavaScript le plus haut possible dans la structure de la page. Le script ainsi inséré est donc systématiquement le premier chargé et exécuté par la page web visité. Cette approche permet d'éviter tout problème que pourrait poser un quelconque script sur la page web qui pourrait modifier le comportement de certaines API dans le contexte de la page web.

Le code chargé se doit de ne pas entré en conflit avec celui de la page web. La solution est un système de module et une bonne méthodologie. Dans les différents modules, on s'interdit de créer ou de modifier des variables globales. Il est même possible de rendre le script injecté totalement invisible à la page. Cette technique est très simple. Elle consiste en la création d'une fonction anonyme que l'on appelle directement au niveau de sa définition.

```
(function() {
  // Code isolé
})();
```

Cette méthode permet d'éviter la création d'une quelconque référence dans l'espace de nom de la page et permet donc l'utilisation de variables locales. Il n'existe pas de moyen pour la page web d'accéder aux variables et fonctions qui pourraient être définies dans cette espace.

Le système de chargement de modules utilise cette méthode pour s'initialiser. Il met à disposition une interface permettant aux modules de se définir. Chaque module utilise cette interface pour définir son nom, ses dépendances et son implémentation.

D. Modularité

Le projet SWAP a été conçu dès le départ pour être modulaire. La réalisation du code JavaScript ne déroge pas à ce principe. Pour atteindre cet objectif, le code JavaScript a été décomposé en modules élémentaires. Le module *Loader* est complètement autonome des autres modules. Il est le seul qui est injecté dans la page web à annoter (un élément `script` en premier enfant de l'élément `head` de la page) au moment de son transit au sein du proxy.

Son rôle consiste à charger les autres modules tout en résolvant automatiquement le chargement des dépendances entre modules et en garantissant que chaque module n'est chargé et exécuté qu'une seule fois lorsque ses dépendances sont résolues.

L'interface d'annotations s'appuie sur deux modules principaux qui sont ensuite complétés par des modules spécifiques d'annotations. Le module *Communication* offre une API simple de communication avec le proxy via le serveur web interne du proxy. Il repose sur l'implémentation jQuery [29] des communications afin d'assurer une indépendance par rapport au navigateur. Le module *EventEmitter* offre une API implémentant le pattern Observateur (abonnement à l'émission de signaux). Il permet de simplifier les communications entre les modules, de gérer plus simplement les événements et de permettre aux différents modules d'exposer des interfaces publiques très simples à utiliser.

L'interface d'annotations est en cours d'extension. Pour l'instant, le module *Zone* permet le marquage de zones dans le document. Une zone correspond à une annotation (type, commentaire...). Elle peut être dans différents états : en cours de création, valide, invalide ou supprimée. Ce module gère l'enregistrement, le chargement, l'affichage et la sélection des annotations. Les zones sont transmises au proxy via le serveur interne puis transmises au serveur pour stockage avec différentes informations telles que le Xpath, le DOM anonymisé de la page ou le type de zone (entête, section, menu, contenu principal...). La figure 4 présente un exemple d'utilisation de ce module d'annotations.

D'autres modules d'annotations sont prévus à l'avenir afin de permettre d'annoter plus de choses dans une page web (annotations de liens, d'enchaînements, de tableaux...).

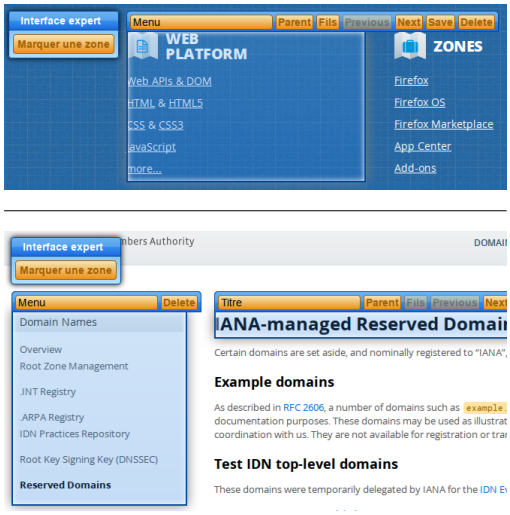

FIGURE 4. Deux exemples d'utilisation du module d'annotations de zones

IV. CONCLUSION

De part sa conception modulaire, nous avons vu que la plateforme SWAP se prêtait bien à la création d'une interface d'annotations de page web. Elle est réalisée grâce à l'inclusion de code JavaScript au sein de la page à annoter lors de son passage dans le proxy. Nous avons montré que l'isolation du code JavaScript et donc de l'interface d'annotations pouvait être réalisée de façon relativement simple. L'infrastructure de l'interface d'annotations a été conçue de manière modulaire et un premier module d'annotations de zones a été réalisé. De nombreux modules d'annotations spécifiques restent encore à concevoir mais grâce aux paradigmes choisis, leurs réalisations sera aisées. Finalement, nous pouvons noter que l'infrastructure de l'interface d'annotations peut être généralisée et détournée de son objectif premier. Plutôt que d'annoter, il est possible par exemple de l'utiliser afin de fournir de nouveaux moyens de navigations ou d'accès à l'information au sein de la page sans interférer avec la page initiale.

RÉFÉRENCES

[1] Web content accessibility guideline 1.0. W3C Web Accessibility Initiative. [Online]. Available at http://www.w3.org/TR/WCAG10/

[2] Web content accessibility guideline 2.0. W3C Web Accessibility Initiative. [Online]. Available at http://www.w3.org/TR/WCAG20/

[3] "Loi n°2005-102 du 11 février 2005 pour l'égalité des droits et des chances, la participation et la citoyenneté des personnes handicapées," JO n° 36 du 12 février 2005, p2353, République Française, 2005. [Online]. Available at http://www.legifrance.gouv.fr/WAspad/UnTexteDeJorf?numjo=SANX0300217L

[4] Idi web accessibility checker : Web accessibility checker. [Online]. Available at http://achecker.ca/checker/index.php

[5] Accessibility wizard. Binary Blue. [Online]. Available at http://www.binaryblue.com.au/access_wizard/

[6] S. Colas, N. Monmarché, D. Burger, and S. Mohamed, "A web site migration support tool to reach european accessibility standards," in *9th European conference for the Advancement of Assistive Technology in Europe (AAATE'07)*, vol. 20, San Sebastian (Spain), october 2007, pp. 907–911.

[7] Betsie. BBC. Http ://www.bbc.co.uk/education/betsie.

[8] S. S. Brown and P. Robinson, "A world wide web mediator for users with low vision," in *Proceedings of Conference on Human Factors in Computing Systems (CHI 2001 Workshop No. 14)*, Seattle, WA, 2001.

[9] M. Colajanni, R. Grieco, D. Malandrino, F. Mazzoni, and V. Scarano, "A scalable framework for the support of advanced edge services," in *High Performance Computing and Communications*. Berlin, Heidelberg : Springer Berlin Heidelberg, 2005, vol. 3726, pp. 1033–1042.

[10] mod_accessibility. WebThing. [Online]. Available at http://apache.webthing.com/mod_accessibility/

[11] B. Parmanto, R. Ferrydiansyah, X. Zeng, A. Saptono, and I. Sugiantara, "Accessibility transformation gateway," in *System Sciences, 2005. HICSS '05. Proceedings of the 38th Annual Hawaii International Conference*. IEEE, 2005, pp. 183a–183a.

[12] R. Han, P. Bhagwat, R. LaMaire, T. Mummert, V. Perret, and J. Rubas, "Dynamic adaptation in an image transcoding proxy for mobile web browsing," *IEEE Personal Communications*, vol. 5, no. 6, pp. 8–17, Dec. 1998.

[13] S. Gupta and G. Kaiser, "Extracting content from accessible web pages," in *International Cross-Disciplinary Workshop on WebAccessibility*. ACM Press, 2005, p. 26.

[14] S. Gupta, G. E. Kaiser, P. Grimm, M. F. Chiang, and J. Starren, "Automating content extraction of HTML documents," *World Wide Web*, vol. 8, no. 2, pp. 179–224, Jun. 2005.

[15] Muffin. [Online]. Available at http://muffin.doit.org/

[16] Rabbit. [Online]. Available at http://rabbit-proxy.sourceforge.net/

[17] D. Hermsdorf, H. Gappa, and M. Pieper, "WebAdapter : A prototype of a WWW-browser with new special needs adaptations," in *Proceedings of the International Conference on Computers Helping People with Special Needs (ICCHP 98)*, 1998, p. 151–160.

[18] A. Mereuta, S. Aupetit, and M. Slimane, "Compensation automatique de la perte de contraste de pages web pour le daltonisme," in *Handicap 2012*, Paris, Jun. 2012, p. 39–44.

[19] S. Aupetit, A. Mereuta, and M. Slimane, "Accessibilité par les couleurs : comparaison d'opérateurs interruptibles pour la recolorisation automatique de pages web," in *Handicap 2012*, Paris, Jun. 2012, pp. 27–32.

[20] A. Mereuţă, S. Aupetit, and M. Slimane, "Improving web accessibility for dichromat users through contrast preservation," in *Computers Helping People with Special Needs*, vol. 7382. Springer Berlin Heidelberg, 2012, pp. 363–370.

[21] S. Aupetit, A. Mereuţă, and M. Slimane, "Automatic color improvement of web pages with time limited operators," in *Computers Helping People with Special Needs*, vol. 7382. Springer Berlin Heidelberg, 2012, pp. 355–362.

[22] A. Mereuta, S. Aupetit, N. Monmarché, and M. Slimane, "Web page textual color contrast compensation for CVD users using optimization methods," *Journal of Mathematical Modelling and Algorithms in Operations Research*, Oct. 2013.

[23] A. Mereuta, S. Aupetit, N. Monmarché, and M. Slimane, "An evolutionary approach to contrast compensation for dichromat users," in *Proceedings of Evolution Artificielle 2013, Lecture Notes in Computer Science*. Springer, 2013, to appear.

[24] Spring framework. [Online]. Available at http://www.springsource.org/

[25] Apache maven. [Online]. Available at http://maven.apache.org/

[26] Hibernate. [Online]. Available at http://hibernate.org/

[27] Java Persistence API. [Online]. Available at https://www.jcp.org/en/jsr/detail?id=317

[28] "Jetty servlet engine and HTTP server." [Online]. Available at http://www.eclipse.org/jetty/

[29] jQuery. [Online]. Available at http://jquery.com/

Interface d'annotations de page web pour le projet SWAP

Département Informatique
5e année
2013 - 2014

Rapport de projet de fin d'études

Résumé : Projet de fin d'étude consitant à la mise en place d'une interface d'annotations de page web. Ce PFE utilise le projet SWAP comme base. L'interface est intégrée à une page web via un proxy. Le but étant de ne pas dépendre d'un navigateur particulier pour réaliser l'interface.

Mots clefs : pfe, java, spring, proxy, persistance, interface, annotations, web, javascript

Abstract: Final study project about making an annotation interface for webpages. The SWAP project is used as base and provide the proxy functionnalities. The goal of this interface is to not rely on a particular browser to work. Everything is portable and browser independant.

Keywords: pfe, java, spring, proxy, persistance, interface, annotations, web, javascript

---

Encadrant
Sébastien AUPETIT
sebastien.aupetit@univ-tours.fr

Étudiants
Vincent ROUILLÉ
vincent.rouille@etu.univ-tours.fr

Université François-Rabelais, Tours

DI5 2013 - 2014