

Implémentation dans Active Tags d'un module Web pour XUnit

The logo for XUnit, featuring the word 'Xunit' in a stylized, outlined font with a slight glow effect.

Table des matières

1. Introduction	2
2. Application Web interactive en Reflex	2
3. Module Web pour XUnit	3
3.1. Principes	3
3.2. Implémentation des tags	3
3.2.1. <web:start> implémenté par la classe Start.java	4
3.2.2. <web:post> et <web:get> implémentés par la classe PostGet.java	4
4. Scénario de tests	4
5. Utilisation et Tutoriaux	5
6. Synthèse	5
6.1. Problèmes rencontrés	5

1. Introduction

L'essor du langage XML est largement dû à sa syntaxe simplifiée. Active Tags est une spécification en XML d'un langage de programmation. Reflex, développé par Philippe Poulard, en est une implémentation en Java. Reflex propose à l'aide de différents modules de développer des applications batch ou web. Xunit est un autre module permettant à la manière de JUnit, de réaliser en Active Tags des tests. Notre travail consistait à développer une nouvelle version du module Web permettant d'émuler le fonctionnement d'une servlet afin de tester une application Web Active Tags en batch. Ce projet nous a confrontés pour la première fois à l'étude d'un langage et de son implémentation. Nous avons commencé par faire les tutoriaux pour pas à pas en arrivant à étudier les sources de l'implémentation. Entre temps nous avons développé une application Web sur laquelle nos tests porteront.

2. Application Web interactive en Reflex

L'application Web développée permet, par l'intermédiaire d'un formulaire, d'écrire côté serveur, des fichiers contenant les informations saisies. Des tests sont effectués sur les champs afin de vérifier la cohérence des types.

Un schéma décrivant le fonctionnement principal de l'application Web est fourni en annexe.

L'application a été réalisée successivement de 3 manières différentes pour finalement arriver à la forme retenue. En voici donc les 3 versions successives :

1ère version:

Nous avons une page html en dur (le formulaire) qui après un traitement xcl crée un fichier xml en dur contenant les réponses puis affiche la liste de ces fichiers xml (cette page est créée dynamiquement à partir du xcl). Finalement on réaffiche la première page avec le formulaire rempli grâce à un traitement xcl (page pas en dur). Cette version avait pour principal inconvénient une forte redondance du code. Mais cette application nous a permis de comprendre l'esprit Reflex, et peu à peu d'entrer dans l'implémentation.

2ème version:

Nous avons factorisé le code de la version 1, tout le traitement se fait à partir d'un unique fichier xcl.

Il n'y a donc aucun fichier en dur autre que ce fichier. Cette version avait le problème de ne pas être modulaire et demandait plus de travail lors d'un changement du formulaire. En effet, l'utilisateur aurait dû modifier le code xcl afin de pouvoir effectuer ses changements.

3ème version:

Il s'agit d'un mélange des 2 versions précédentes nous utilisons des fichier xml avec une feuille de style xslt pour les pages qui seront jamais modifiées. Cela permet de ne modifier avec une feuille xcl que certaines données. Ainsi nous avons un fichier xml qui définit simplement un formulaire ce qui permet une aisance future lors d'une modification du formulaire. En effet, xml a cet avantage d'être facilement compréhensible. Lors d'un enregistrement, la feuille xcl devra récupérer ce fichier représentant le formulaire et le modifier avec les valeurs données par l'utilisateur.

L'application effectue une vérification limitée sur les champs du formulaire :

- Un champ ne peut être vide
- Un type compatible avec ce qui est attendu

Chaque utilisateur pourra bien évidemment définir ses propres vérifications sur les champs (qu'il pourra aussi modifier).

3. Module Web pour XUnit

L'implémentation du nouveau module n'a pu être entreprise qu'après la compréhension des grandes lignes du fonctionnement de Reflex.

3.1.Principes

Le module Web doit permettre la réalisation de tests sur une application Web en batch. Il doit pour cela émuler les envois de requêtes, POST et GET, à l'application et effectuer les mapping nécessaires à leur exécution. Il est aussi nécessaire de gérer l'initialisation de l'application et d'émuler le contexte d'exécution de l'application qui est habituellement fourni par la servlet. Nous avons créé une implémentation de trois nouvelles balises active tags spécifiées dans notre module Web 1.1.

Ces balises sont :

- <web:start>
- <web:get>
- <web:post>

3.2.Implémentation des tags

3.2.1. <web:start> implémenté par la classe Start.java

Le rôle de cette classe est d'exécuter le web:init de l'application web. Il a en général pour but de définir un ensemble de variables visible globalement. Web:start créé un ServletContext qui nous est propre définissant seulement le path de l'application Web (représenté par Web://). Ce contexte est fournit à l'application.

Il défini dans le contexte interne à l'aide d'un XComponent, qu'il a créé pour l'exécution de l'application, le stream qui est normalement utilisé pour la réponse au client (\$web:response/@web:output) .Ce stream est dirigé vers un fichier temporaire qui sera ensuite récupéré pour définir le corps de la réponse.

3.2.2. <web:post> et <web:get> implémentés par la classe PostGet.java

Le rôle de cette classe est d'exécuter le bon mapping correspondant à l'url spécifiée. Dans le cas d'un post, on récupère les paramètres issus des balises <xcl:param>. Dans le cas d'un get on découpe l'url pour récupérer les paramètres. Ces paramètres sont ensuite placés dans un XComponent. Cet objet, sur lequel XPath peut opérer, sera à son tour inséré dans le contexte créé pour l'exécution du mapping adéquat.

Le fichier temporaire créé par le start est ensuite parsé afin de créer la réponse complexe. L'attribut body contient le nœud DOM du fichier parsé.

4. Scénario de tests

Voici l'organisation des différents tests :

- Test suite du POST :

- Cas où les arguments donnés au formulaire sont bons : vérification que le fichier créé est le même qu'un fichier préalablement écrit en dur.
- Cas où les arguments donnés au formulaire sont bons : vérification qu'un fichier a été écrit sur le disque.
- Cas où il y a une erreur dans les arguments donnés au formulaire : vérification que la page suivante est bien la page d'erreur.
- Cas où il y a une erreur dans les arguments donnés au formulaire : vérification qu'aucun fichier n'a été écrit sur le disque.

(NB : ces 2 derniers tests sont réalisés pour chacun des 4 champs du formulaire. De plus, le champ de l'âge provoque une erreur dans 2 cas, donc pour ce champ il y a un test en plus)

- Test suite du GET :

- Cas où les arguments donnés au formulaire sont bons : vérification que le fichier créé est le même qu'un fichier préalablement écrit en dur.
- Cas où les arguments donnés au formulaire sont bons : vérification qu'un fichier a été écrit sur le disque.

- Cas où il y a une erreur dans les arguments donnés au formulaire : vérification que le page suivante est bien la page d'erreur.
 - Cas où il y a une erreur dans les arguments donnés au formulaire : vérification qu'aucun fichier n'a été écrit sur le disque.
- **Test suite contenant divers tests :**
- Vérification de la page qui liste les formulaires existants

Tests abandonnés pour diverses raisons :

- Test que le nombre d'éléments dans la liste des formulaires s'incrémente bien quand on ajoute un nouvel élément.

5. Utilisation et Tutoriaux

Tout ce qui suit suppose que Reflex est installé et fonctionne correctement sur votre machine.

Le module se compose d'un jar et d'un catalogue. Pour l'installation, il suffit de mettre le jar dans le même répertoire que celui de l'installation de Reflex.

Le module est accompagné d'une batterie de tests (exemple.xcl). Ce sont des exemples d'utilisation. Des tutoriaux sont disponibles à l'adresse <http://pw.swd.fr/tutorial.htm> .

6. Synthèse

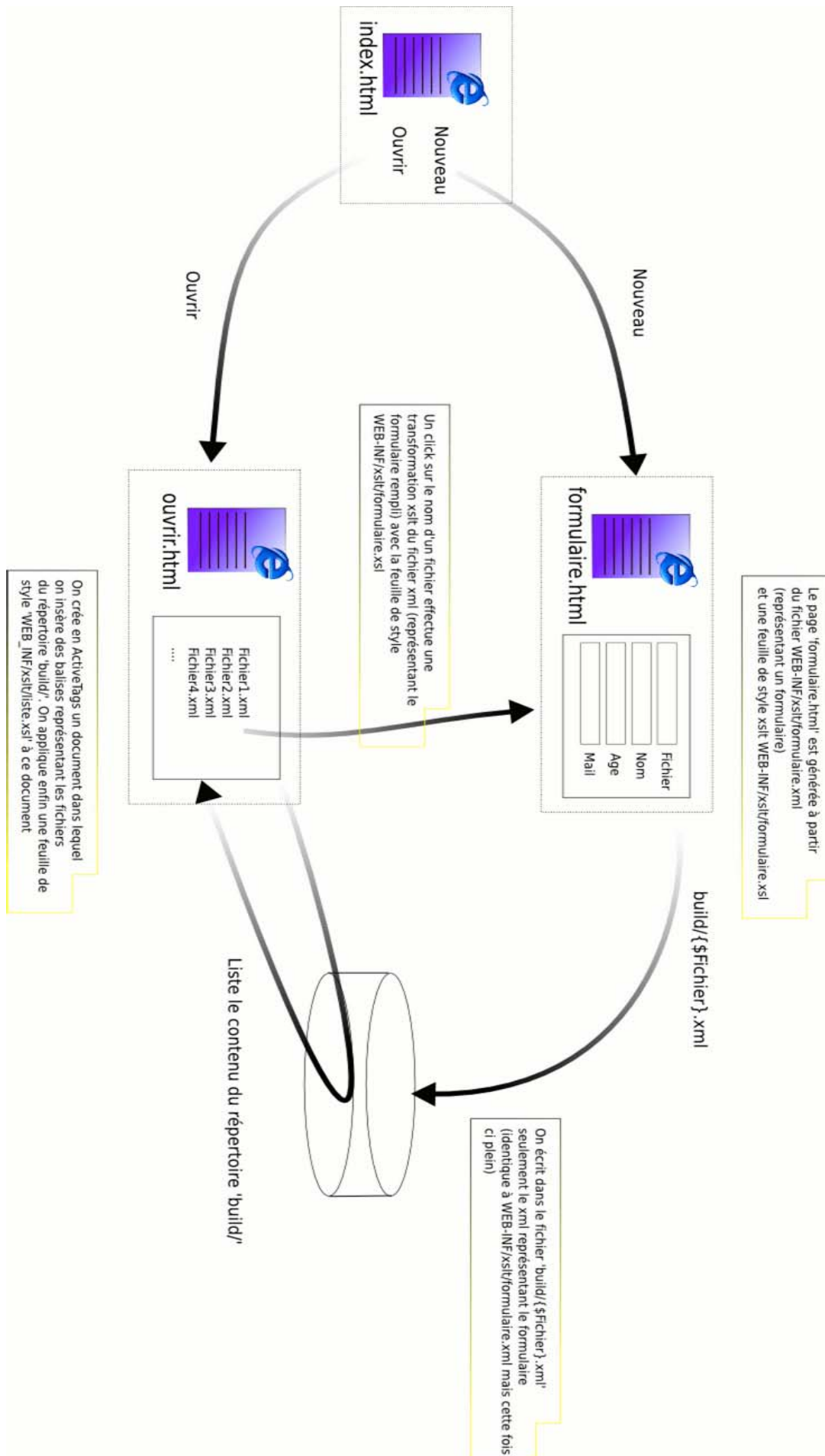
6.1.Problèmes rencontrés

Concernant l'implémentation notre attribut `$web:response/@header` n'est pas rempli. Il aurait par exemple pu contenir des informations sur le mime-type. A part pouvoir effectuer sur le mime-type l'implémentation de l'attribut header ne nous est pas apparu pertinente. Il aurait fallu créer une `ResponseProperty` alors que nous avons directement créé un document DOM à partir du fichier temporaire créé.

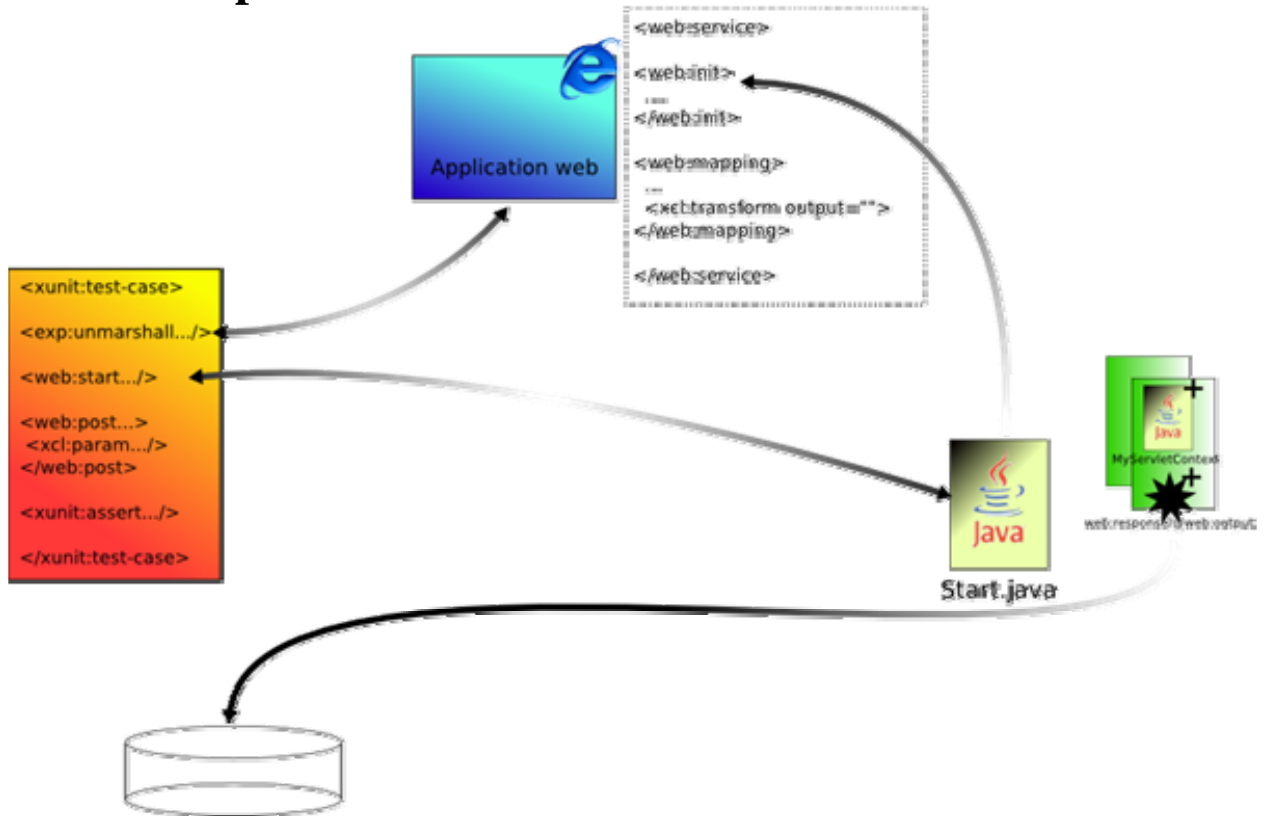
Le test qui vérifie si la page de listing est correcte nous a obligé à faire un système de set up/tear down à la manière de JUnit. En effet, pour faire une comparaison avec un fichier écrit

en dur, on doit travailler sur un listing qui est toujours identique. Avant de lancer ce test, on sauvegarde les fichiers contenus dans le répertoire build dans un répertoire tmp-form qui se trouve dans le répertoire d'où sont lancés les tests, puis on supprime les fichiers xml du répertoire build. On ajoute ensuite 2 formulaires avec des `<web:post>`. Après le test, le répertoire build est remis à l'état d'origine.

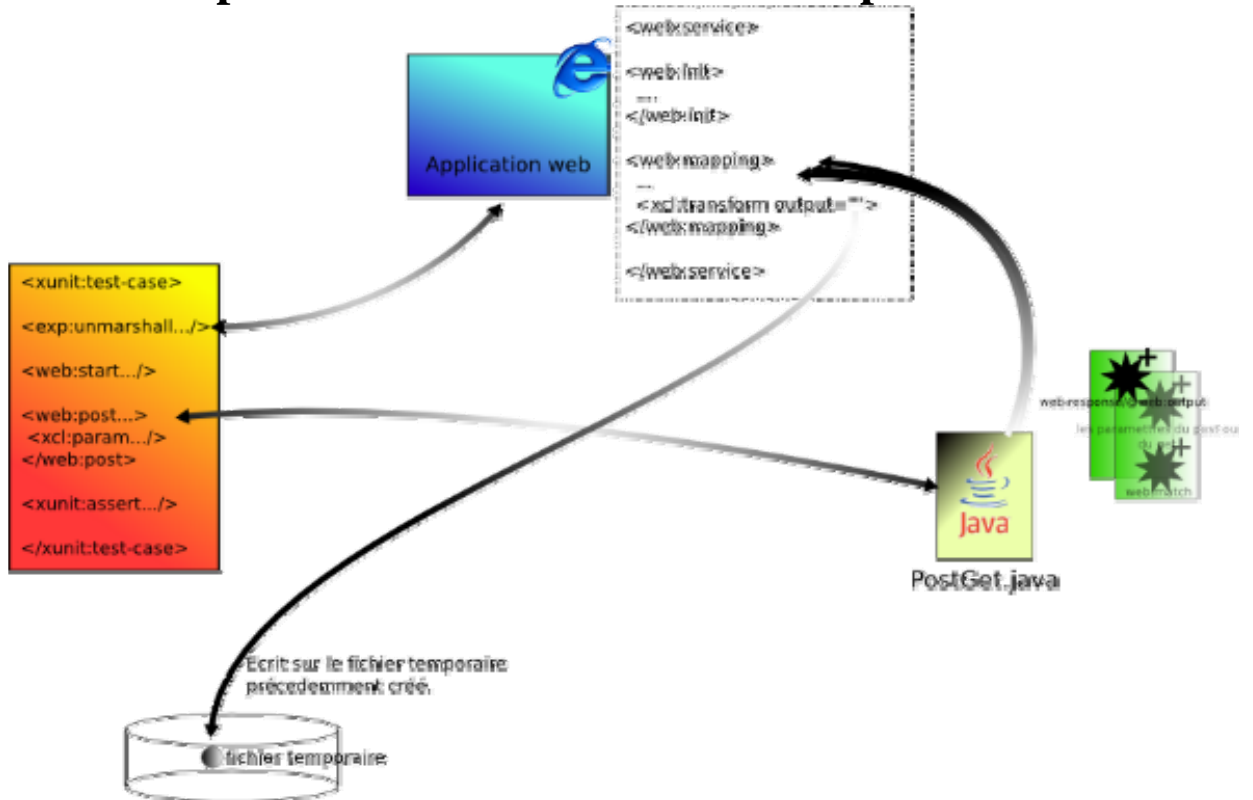
Nous nous sommes demandés s'il était pertinent de vérifier qu'un fichier n'est pas écrit sur le disque dans le cas où on donne au formulaire un champ « fichier » vide. En effet, avec un tel champ vide, comment s'appellerait le fichier créé sur le disque? Nous avons décidé de garder ce test dans le cas où le fichier créé s'appellerait « .xml ». Nous aurions pu éventuellement compter le nombre de fichier dans build.



Implémentation module 1.1 <web:start>



Implementation module 1.1 <web:post>



Implementation module 1.1 les tests

