Pillar et Étude du format Epub

X. Van de Woestyne et P. Laforgue

Pillar

Pillar

> ***Pillar*** *is a markup syntax and a project with associated tools to write and generate documentation and books. It is written in* ***Pharo*** *and covered with* ***many unit tests****.*

Torsten Bergmann

Premier contact avec l'auteur

Plusieurs axes à explorer

- Modifier le flot de compilation vers les formats de sorties
- Proposer des formats d'exports alternatifs (lisible par une liseuse)
- Agrémenter la grammaire de **Pillar**
- Réfactoring général de plusieurs composantes

Pré-contribution

*La lecture des documents relatifs à **Pillar** amènent généralement à cet excellent document "**Documenting your Project with Pillar**. Certains liens cassés entraînent notre première PullRequest en rapport avec le projet **Pharo**."*

Liens cassés (ou modifiables)

- Voyage documentation
- Pillar CheatSheet
- Emacs: pillar-mode
- Pillar documentation (itself)

Intégration d'un nouveau format

*La première étape, après l'apprentissage sommaire de **Pharo**, que nous avons décidé d'explorer a été la compréhension de l'anatomie de **Pillar** en entamant la sontruction d'un exportateur **epub**.*

- Compréhension de la structure du projet
- Initiation au **TDD**
- Nécessité de se documenter

Constituants grammaticaux de Pillar

Mise en forme du texte

- `""text""` : Texte en gras
- `''text''` : Texte en italique
- `==text==` : Texte mis dans du code inline
- `--text--` : Texte barré
- `@@text@@` : Texte en indice
- `^^text^^` : Texte en exposant
- `__text__` : Texte souligné

Structure logique d'un document

- `!` `en début de ligne` : Correspond à la description d'un chapitre. Le niveau d'imbrication des chapitres est défini par le nombre de `!` utilisé (comme le # en **Markdown**).
- Les paragraphes démarrent avec une ligne vide et peuvent être annotés au moyen d'un texte préfixé par `@@`.

Enumérations

- `-` : Démarre une liste non ordonnée
- `#` : Démarre une liste ordonnée
- `;key :value` : Démarre une liste labelisée

Les listes pouvant être imbriquées.

Liens et images

- `*label>url*` : lien sur `label` pointant vers `url`
- `@anchor` (sur une ligne séparée) : défini une ancre
- `*anchor*` : lien pointant vers une ancre
- `+caption>url|parameters1|param2|etc+` : figure

Code et script

```
[[[
    code
]]]
```

Paramètrable

```
[[[param1|param2|param3|etc
    code
]]]
```

Un snippet peut être défini au moyen du paramètre `label` et afficher au moyen d'un lien de référence : `*label*`.

Tables

```
|!Language |!Coolness
|Agda | Hypra cool
|OCaml| Hypra cool²
|Java | baaad
```

Ou en utilisant des aligneur de texte :

```
||centered||centered||centered
|{ left |} right || center
```

Et encore

- Un système de configuration expressif
- La possibilité d'intégrer du code **LaTeX** brute
- Et sans aucun doute beaucoup d'autre chose nous ayant échappé!

Composition sommaire de Pillar

Pillar utilise le motif de conception **Visitor** pour désolidariser la logique de transformation de la grammaire cible.

Model

- `Pillar-Model\Core` : Object et BasicObject Monkeypatch
- `Pillar-Model\Document` : Description de la morphologie d'un document
- `Pillar-Model\ScriptLanguage` : Outil de parsing des langages colorisables par Pillar
- `Pillar-Model\Visitor` : Visiteur Abstrait

Epub

Epub : une standardisation du HTML

Epub et HTML : Brothers From Different Mothers

> *"La magie d'Epub c'est qu'il est un standard construit à partir de standards déjà existants. Il en résulte un format très facile à manipuler à destination des ebooks".*

Bill Kasdorf (Vice President, Apex Content Solutions)

Epub 3 : un format extrêmement riche

Le format Epub permet d'ajouter quelques informations supplémentaires au HTML. Notamment dans la sémantique des expressions.

Les Metadata : l'arme fatale de l'Epub 3 ?

- Fichiers XML (XHTML, SVG)
- Images (GIF, JPEG, PNG, SVG)
- Feuille de style (CSS3)
- Scripts javascript
- Audio and vidéo
- Fonts (openType or WOFF)
- **Metadata** !

Think outside the book ;)

Rétrocompatibilité entre Epub 3 et Epub 2

Il tient simplement en trois metadonnées élémentaires : un **identifiant**, un **titre** et une **langue**.

- **identifier** : identifier element contains a single identifier associated with the EPUB Publication, such as a UUID, DOI, ISBN or ISSN. There is a unique main identifier but it is possible to include extra-identifiers.
- **title** : title element represents an instance of a name given to the EPUB Publication. Main title must be unique, but we can have numerous titles as subtitle, collection etc..
- **language** : language element specifies the language of the Publication content (what a surprise).

Ces trois metadonnées sont regroupées sous le sigle `DCMES` (Dublin Core Metadata Element Set).

Premiere étape : où disposer des metadatas dans Pillar ?

Heureusement, Pillar nous offre la possibilité d'utiliser un fichier de configuration externe : *pillar.conf* .

Il est écrit en `STON`, un format similaire au fameux `JSON`.

Il est aussi possible d'incure des metadatas directmeent dans les fichiers en les renseignant entre double-accolades.

Metadonnées dans Pillar

Par exemple : `{{ title : 'Pharo par l'exemple !'}}`

- `{{keyword : value}}` : Crée une relation entre un document et un template

  L'usage de `{{key}}` dans un template, substitue par `value` à la compilation du document.

Nous préferons toutefois utiliser le fichier de configuration externe lorsque cela est possible.

Exemple

```
<package ... unique-identifier="pub-id">
 ...
 <metadata xmlns:dc="http://purl.../">
  <dc:identifier id="pub-id">urn:98...</dc:identifier>
  <dc:title>Pharo par l'exemple !</dc:title>
  <dc:language>fr</dc:language>
  <meta property="dcterms:modified">2014-06..</meta>
 </metadata>
  ...
</package>
```

Traduit en **STON** :

```
{
    "title" : "Pharo par l'exemple !",
    "language" : "fr"
    "EPUB" : {
        "outputType" : #epub,
        "identifier" : {
            "urn:uuid" : "98987..."
        },

    }
}
```

Enrichir son document avec les metadonnées

- Fournir plusieurs versions du titre est possible
- Raffinage avec les title-type : sous-titre, titre de la série. . . ou bien le numéro du tome etc..
- Utilisation des items : possibilité de préciser l'image de couverture, du contenu mathématique etc..
- Préciser les auteurs et leurs rôles dans le développement du livre..

Enrichir son document avec les metadonnées

- **epub:type** : utilisation de la sémantique xml
- **epub:switch** : permet d'intégrer du code xml externe
- **epub:trigger** : inclure des vidéos HTML5 et autre contenu multimédia

Comment (bien) écrire un document epub ?

- Penser à la pluralité des médias, ceci n'est pas un simple *bouquin* !
- Modulariser votre code : séparer vos fichiers (chapitres) etc..
- Pour le moment, la compilation de plusieurs fichiers dans Pillar demande au lecteur de renseigner l'ordre dans lequel il désire les voir s'afficher. *Nous pensons améliorer les routines de compilation si cela nous est possible*.

Difficulté de l'intégration à Pillar

Actuellement, **Pillar** produit des documents *monomorphiques* (même s'il est possible de fragmenter la sortie). Pour la construction d'un document **Epub**, il est nécéssaire de produire une collection de fichier (dont certains sont constants et d'autres variants).

Processus d'exportation confortable

Actuellement, il est confortable d'utiliser un squelette proposant une collection de scripts **bash** pour automatiser certaines tâches.

Avantages

- Contrôle de la création des fichiers constant
- Contrôle des passes de **Pillar**

```
function pillar_all() {
    ./pillar export --to='HTML by chapter'
}

function compile_chapters() {
    chapters=$(./pillar show inputFiles 2>/dev/null)
    for chapter in $chapters; do
    pillar_file=$(basename $chapter)
    dir=$(dirname $chapter)
    done
}

echo 'Generating files for all chapters'
pillar_all
compile_chapters
echo 'Done'
```

Et voici notre fichier de configuration *pillar.conf*

```
{
    "newLine" : #unix,
    "headingLevelOffset" : 1,
    "defaultScriptLanguage" : "smalltalk",
    ...
    "inputFiles" : [
        "Book/Chap1.pillar",
        "Book/Chap2.pillar"
    ]
}
```

Fichiers générés

Fichiers sources :

```
-- Book/
---- pillar.conf
---- chap1.pillar
---- chap2.pillar
```

Fichiers générés

Fichiers générés :

```
-- Book.epub/ (zipped)
---- mimetype
---- META-INF/container.xml
---- META-INF/com.apple.ibooks.display-options.xml
---- stylesheets.css
---- title_page.xhtml
---- nav.xhtml
---- toc.ncx
---- chap1.xhtml
---- chap2.xhtml
```

Compilation

Compilation en deux passes :

- **step one** : Génération de l'arborescence des fichiers et du parcours de lecture (nav.xhtml, toc.ncx. . .)
- **step two** : Génération des chapitres (un fichier pour un chapitre)

Travaux récents

- Modification du flux de compilation pour le morcelement de fichiers
- Ajout d'un type d'exportateur (nommé Epub) pour décorer les **headers** d'identifiant (ancrables)

  La décoration d'identifiant permet, au delà de générer des documents lisible dans une liseuse, générer dans des fichiers HTML classiques une table des matières.

Le code est évidemment testé.

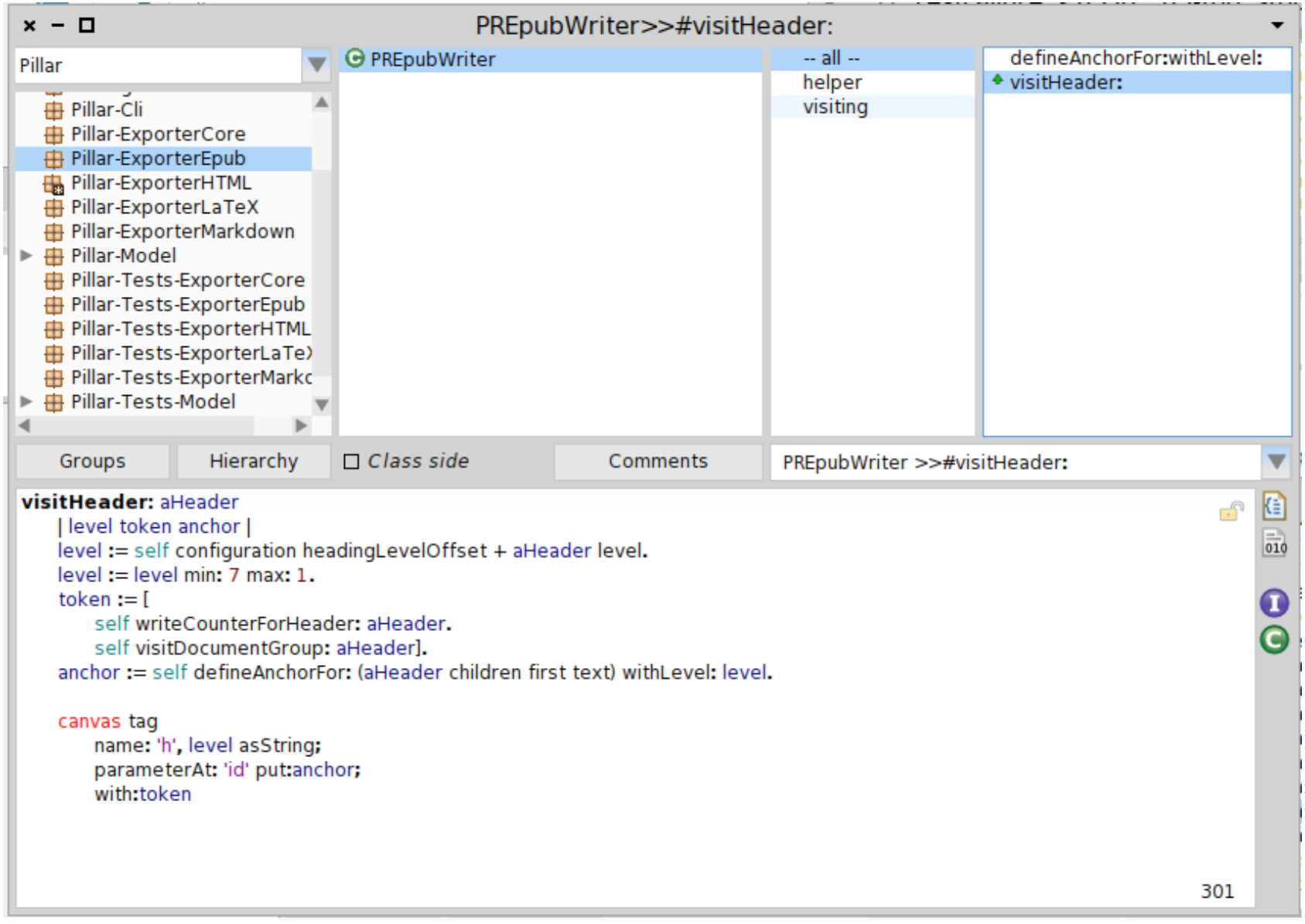

Figure 1: Identification des titres

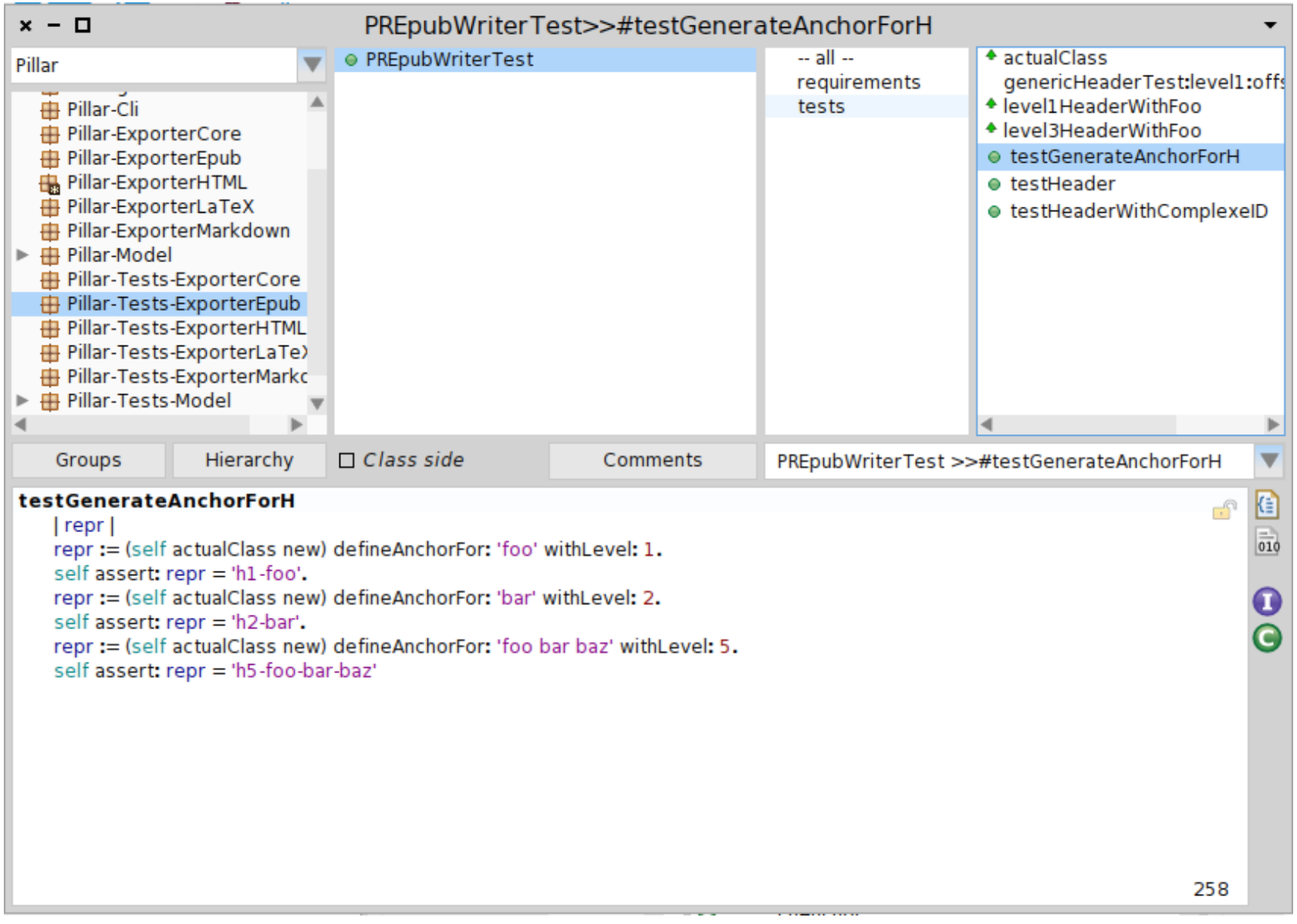

Figure 2: Test – Identification des titres

Travaux futurs

Pour Epub

- Production d'un exportateur vers les tables des matières
- Production d'un exportateur pour les chapitres (presque fini)
- Modification du squelette de construction de projet

En complément

- Fragmenter certaines composantes (`PRExternalLink` par exemple) (entâmé)
- Penser à un export en diapositive (**Beamer**, **reveal.js**)
- Internalisation des routines de compilations

FIN !

Questions, réactions ?