Université de Lille
Cité Scientifique
59650 Villeneuve d'Ascq

Université de Lille

RAPPORT DU PROJET IDM

Transformation de fichiers markdown vers bootstrap et UIkit par une approche IDM

Auteurs:

- DIALLO Mamadou
- FUNGWA MOKE Junior

Année d'étude: Master 1 E-Services

Période : Septembre 2020 - Janvier 2021

Introduction **2**

1. Présentation du Projet **3**
Récupération du Projet sur GitLab 3
Architecture du projet 3

2. Manuel d'installation **3**
Outils nécessaires 3
Installation du projet sur Eclipse 4
Tests des modèles par défaut 6

3. Manuel d'utilisation **8**
Les propriétés générales 9
Les propriétés particulières à un concept 9
Les types supplémentaires uniques à bootstrap 11

4. Manuel de Maintenance **13**
Description des métamodèles 13
Le métamodèle usd 13
le métamodèle Bootstrap 13
le métamodèle UIkit 14
Explication des transformations 14
Les transformations simples 14
Les transformations complexe 15
La génération des page web vers une techno cible 16

5. Historique de développement **16**

7. Conclusion **17**

Annexes **18**
Annexe 1 : Chaîne de transformation 18
Annexe 2 : Étapes de transformation d'une section 19
Annexe 3 : Métamodèle USD 20
usd 20
Table usd 20
Annexe 4 : Métamodèle Bootstrap 20
Bootstrap 21
BootstrapProperty 21
BootstrapTable 22
Annexe 5 : Métamodèle UIKit 22
UIKit 22
UIKitProperty 23

Introduction

Le but de ce projet est de permettre une modélisation des pages web à partir d'une approche basée sur la syntaxe markdown et une approche IDM (Ingénierie Dirigée par les Modèles).

Cette approche permet en passant par la conception des métamodèles avec Ecore, la transformation de modèle vers un autre avec QVTO et la génération avec Acceleo de produire des pages web utilisant les concepts de Bootstrap ou UIkit.

Ainsi, à partir d'un fichier écrit en *markdown* il est possible de passer par quatre étapes pour obtenir une page web qui respecte les approches des deux frameworks *Bootstrap* et *UIkit.* Une première étape consiste à parser le fichier *markdown* en un modèle *markdown,* puis depuis ce modèle par une transformation produire un modèle pivot *usd* qui unifie les concepts visés à partir d'un haut niveau d'abstraction sans pour autant entrer dans les détails spécifiques, ensuite pour le framework visé, effectuer une nouvelle transformation vers un modèle représentatif de ce dernier et enfin par un générateur, générer le code correspondant. (c.f : Annexe 1 : chaîne de transformation).

Dans ce qui suivra dans ce rapport, nous commençons par présenter le projet et les installations requises pour utiliser le logiciel, ensuite nous mettons en évidence les différentes fonctionnalités développés ainsi qu'une illustration de son utilisation avec un petit tutoriel pour enfin mettre en exergue certains éléments clés de sa conception et des perspectives d'améliorations.

1. Présentation du Projet

Dans cette partie, nous présentons comment récupérer le projet qui permet l'utilisation du logiciel et nous fournissons quelques explications sur les principaux dossiers qui le composent.

Dans une prochaine partie de notre rapport, nous parlerons plus en détail des chaque modules qui constituent le projet.

a. Récupération du Projet sur GitLab

Vous pouvez cloner le projet en cliquant sur cette adresse GitHub.

b. Architecture du projet

Le projet cloné est composé des dossiers suivants:

- ***fil.idm.markdown-master:*** Il contient le métamodèle *markdown*, et le parseur de fichier *markdown* en modèle *markdown.*
- ***pivot:*** Dans lequel nous trouvons le projet du métamodèle ***usd*** "*idm.simpleusd.mm",* les deux projets "*idm.simpleusd.mm.edit & idm.simpleusd.mm.editor"* générés à partir du métamodèle et le projet de transformation de markdown vers usd "*idm.md2usd.transfo*".
- ***bootstrap:*** Qui lui contient le projet du métamodèle ***bstrap "****idm.bstrap.mm",* les deux projets "*idm.bstrap.mm.edit & idm.bstrap.mm.editor"* générés à partir du métamodèle, le projet de transformation de markdown vers usd "*idm20.usd2bstrap.transfo*" et le projet de génération de code *Bootstrap "idm.bootstraptransfo.bstrap.gen.java"*.
- ***Ulkit:*** Qui contient le projet du métamodèle ***Ulkit "****idm.Ulkit.mm",* les deux projets "*idm.Ulkit.mm.edit & idm.Ulkit.mm.editor"* générés à partir du métamodèle, le projet de transformation de markdown vers usd "*idm20.usd2Ulkit.transfo*" et le projet de génération de code *Bootstrap "idm.Ulkittransfo.Ulkit.gen.java".*
- ***test:*** Contenant quelques fichiers *markdown* qui peuvent être utilisés pour un éventuel test.
- ***WebsiteTest:*** Un dossier contenant des pages web générées.

2. Manuel d'installation

a. Outils nécessaires

Pour utiliser le projet développé, les outils listés ci-dessous doivent être installés:

- ***EMF****:*

Selon Eclipse, *EMF* est un cadre de modélisation et un outil de génération de code pour créer des outils et d'autres applications basés sur un modèle de donnée structurée. Il se base sur une spécification de modèle pour fournir des outils et une prise en charge de l'exécution pour produire un ensemble de classes java pour le modèle ainsi qu'un éditeur de base.

Pour ce projet, *EMF* permet d'avoir un aperçu du métamodèle et éventuellement de le faire évoluer. Il permet également de générer le code de manipulation des modèles (dont l'éditeur arborescent).

- ***Qvto***:
  Le composant opérationnel Eclipse *QVT* est une implémentation du langage de mappage opérationnel défini par *MOF* (Meta Object Facility). Il assure la transformation d'un modèle A vers un nouveau modèle B.

- **Acceleo** :
  Basé sur des modèles comprenant des outils de création pour créer des générateurs de code personnalisés, *Acceleo* permet de produire automatiquement tout type de code source à partir de n'importe quelle source de données disponible au format *EMF*. Ainsi, il nous permet de générer du code pour android en java à partir des modèles conforme à notre métamodèle de départ.

- ***Éclipse***:
  Nous utilisons cet *IDE* pour permettre de programmer et compiler *EMF, QTV et Acceleo.*

b. Installation du projet sur Eclipse

Pour installer le projet afin d'utiliser le logiciel, ci-dessous vous trouverez les étapes essentielles:

- Lancer *Eclipse* et importer les dossiers des métamodèles *markdown, usd, bstrap et Ulkit* ainsi que leurs dossiers Edit et Editor correspondants. ***Si les dossiers contiennent une erreur, il suffit de créer un dossier `src` vide à la racine du dossier qui contient un métamodèle.*** Importer aussi le parser de fichier *markdown* vers un modèle *markdown.*

  Click sur file > Import > Existing Projects into workspace / Projects from Folder or Archive > Browse > selectProject > Finish

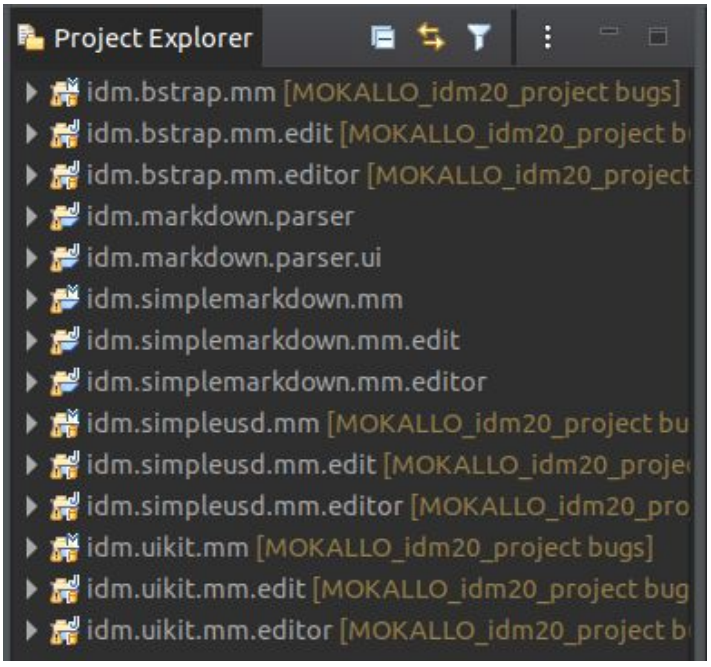


FIGURE 2.b-1 - Les projet à ouvrir sur l'Eclipse de dev

- Faites un clic droit sur l'un des dossiers d'un métamodèle et sélectionner
  Run AS > Eclipse Application

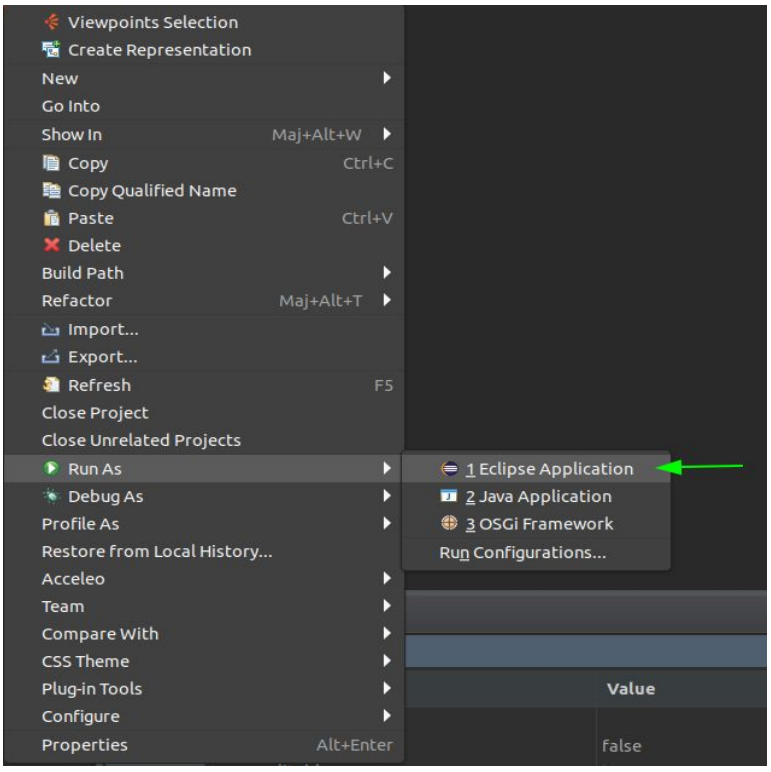


FIGURE 2.b-2 - Lancer le projet sur un Eclipse de test

- Un second *Eclipse* s'ouvre et vous devez y importer les dossiers de transformations (*idm.md2usd.transfo, idm20.usd2bstrap.transfo, idm20.usd2UIkit.transfo)* ainsi que les dossier de test (*tests*).
- Dans ce même *Eclipse,* importer les dossiers de génération et projet dest qui contiendra les pages générées (*idm.bootstraptransfo.bstrap.gen.java, idm.UIkittransfo.UIkit.gen.java et WebSiteTest).*

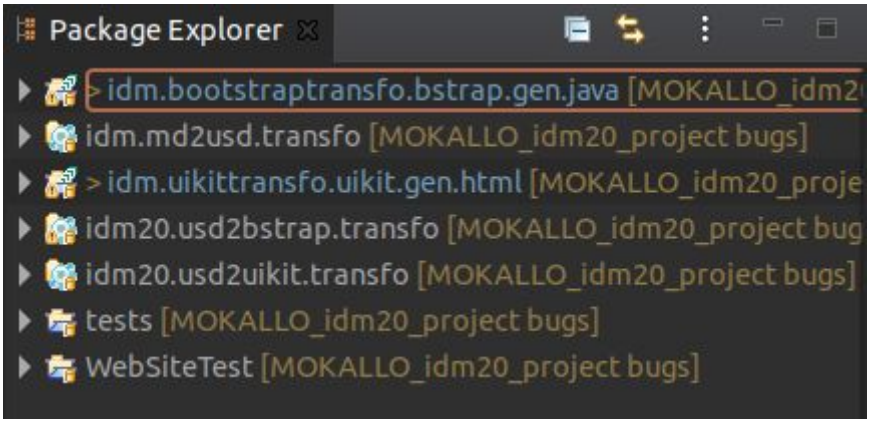


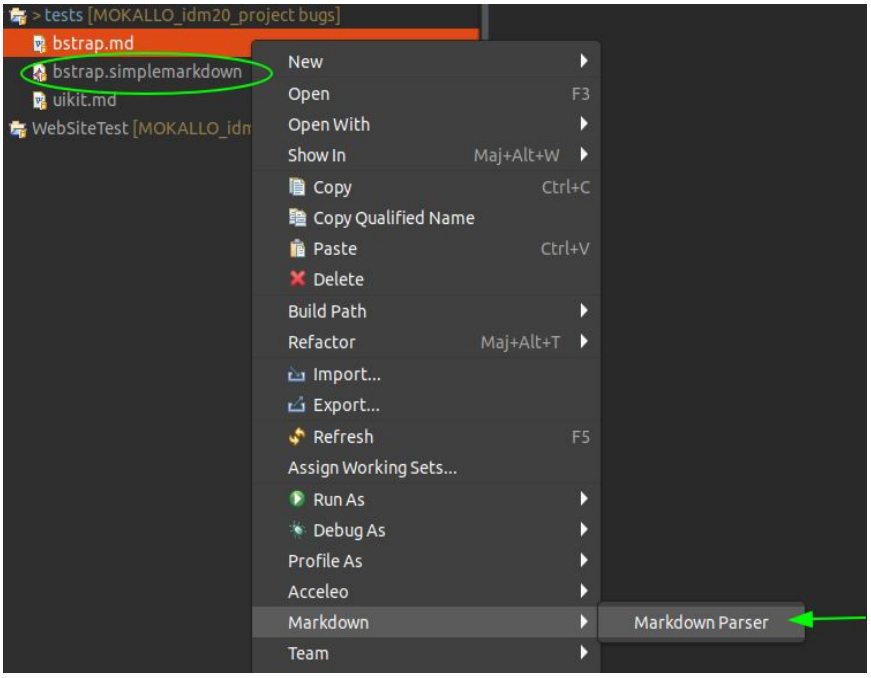
FIGURE 2.b-1 - Les projet à ouvrir sur l'Eclipse de test

c. Tests des modèles par défaut

Dans le dossier *tests* vous trouverez deux fichiers (*bstrap.md* et *uikit.md*) que vous pourrez utiliser pour la chaîne de transformation jusqu'à la génération des pages web. Après la section ***installation***, ces différentes étapes vous aideront à tester ces fichiers:

- Commencer par parser le fichier *bstrap.md* vers un modèle *markdown*

  bstarp.md -> bstrap.simplemarkdown



- Transformer le fichier *simplemarkdown* obtenu vers un modèle *usd*

  Placez vous dans le dossier *md2usd.transforms* du projet *idm.md2usd.transfo* puis

  clic droit, Run As > Run Configuration > double clic QVTO Operational Transformation > config comme sur la capture ci-dessous > Apply > Run

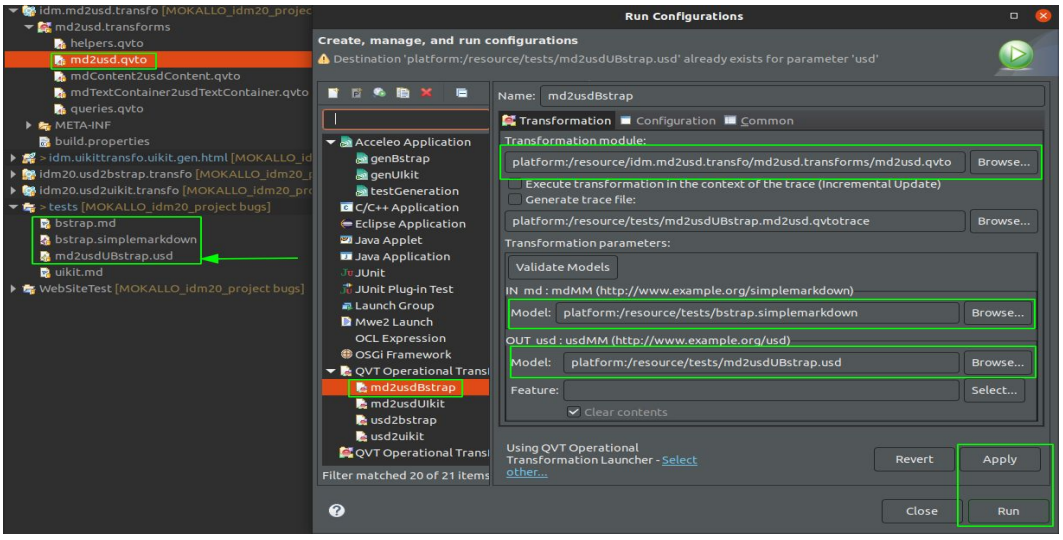


FIGURE 2.c-1 - Configurer la transformation de *markdown* à *usd*

- Transformer le fichier *usd* obtenu vers un modèle *bstrap*
  Placez vous dans le dossier *usd2bstrap.transforms* du projet *idm20.usd2bstrap.transfo* puis
  clic droit, Run As > Run Configuration > double clic QVTO Operational Transformation > config comme sur la capture ci-dessous > Apply > Run

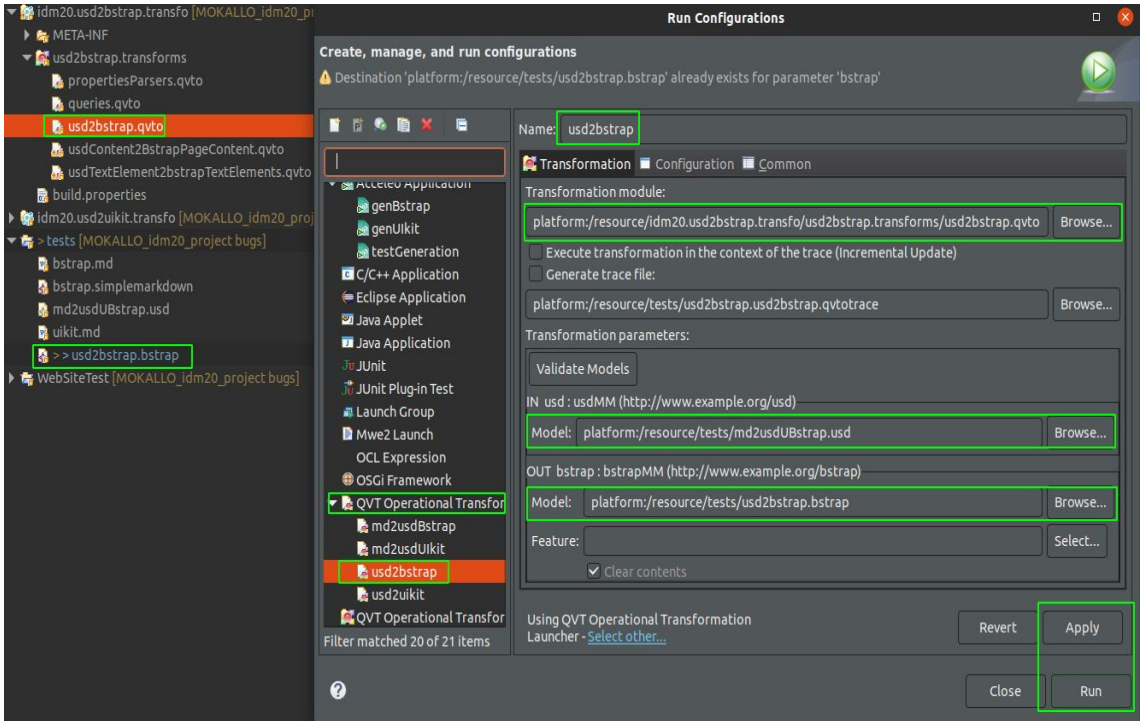


FIGURE 2.c-2 - Configurer la transformation de *usd* à *Boostrap*

- Générer le code correspondant au modèle bstrap obtenu.
  - Placez vous dans le module ***main*** du dossier ***src*** du UIkit générateur puis
  - *Après un double clic sur Acceleo dans le menu à gauche, remplissez la fenêtre qui s'ouvre en choisissant comme modèle l'un de modèle proposé dans `tests/bstrap ou tests/bstra `et en spécifiant comme target `webSites`*
  - Vous pouvez cliquer sur ***Apply>Run***

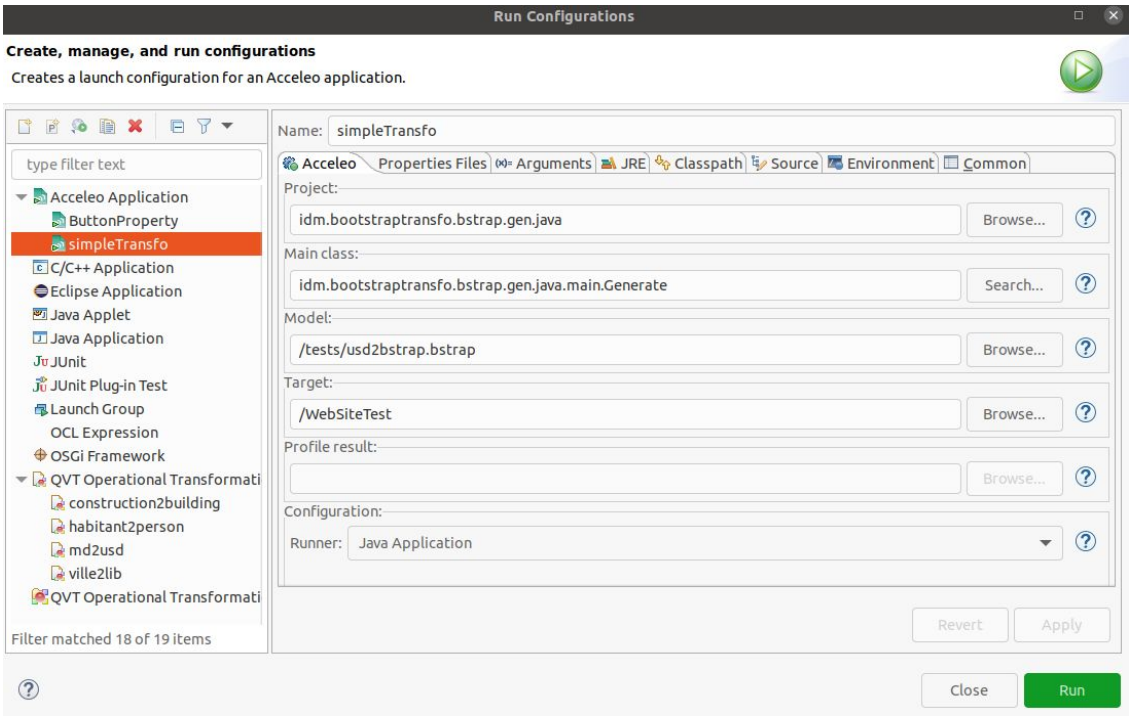


FIGURE 2.c-3 - Configurer la génération de code du modèle *Bootstrap*

- Dans le dossier sélectionné lors de la dernière configuration vous trouverez un fichier *html* ouvrez le et visualisez son contenu.
- Maintenant vous pouvez reproduire les mêmes manipulations pour transformer le fichier *uikit.md* vers une page web avec les concepts *UIkit.*
  - Ce qui change c'est la transformation du fichier usd vers maintenant un modèle *UIkit*, il faut donc sélectionner le projet de transformation de *usd* vers *uikit* (*idm20.usd2uikit.transfo*).
  - Aussi la génération du code vers une page web avec les concepts *UIkit* sélectionner bien le projet de génération d'une page web avec les concepts *UIkit (idm.uikittransfo.uikit.gen.html)* .

3. Manuel d'utilisation

Dans cette partie, nous décrivons les différents concepts principaux et la syntaxe *markdown* prise en charge par notre logiciel ainsi que les concepts supplémentaires.
La conception mise en œuvre pour les deux framework permet de décorer chaque concept par des propriétés spécifiques à un framework. Ci-dessous, nous détaillons comment cette décoration s'effectue.
Les propriétés sont toujours présentées comme suit:

```
properties:(props:val, props:{propVal1:val1, propVal2:val2, ...}, ...)
```

Par exemple:

```
properties:(color:warning,background:primary,margin:{level:5,alignment:l
eft},padding:{level:5,alignment:center},
border:{border,borderColor:secondary,...},...)
```

Nous avons avons ainsi jugé nécessaire de séparer les propriétés en deux type qui sont:

a) Les propriétés générales

Ces propriétés sont celles que peut avoir tous concepts pris en charge par notre logiciel.

Propriété	Valeurs Bootstrap	Valeurs Ulkit	Exemple d'utilisation
Color	dark, primary, secondary, success, info, white, warning, danger et light	muted, emphasis, primary, secondary, success, warning et danger	color:primary
Background	Toutes les couleurs précités	Toutes les couleurs précités	color:secondary
Margin et Padding	level{1,2,3,4} et alignement(center, top, bottom, left, right)	size(default, small,large} et align{center, top, bottom, left, right}	Bootstrap => margin:{level:5,alignment:left} Ulkit => padding:{size:large,alignment:left}
Border	border et coloredBorder	Non	border:{border,borderColor:secondary}
Alert	alertStyle {color} animated	Non	alertProperty:{alertStyle:success,animated}

b) Les propriétés particulières à un concept

Concept	Propriétés Bootstrap	Valeurs Bootstrap	Propriétés Ulkit	Valeurs Ulkit	Exemple d'utilisation
Titre	display	{display1 display2, display3 diplay,4}	titleSize	default, small, large	Bootstrap => display:display1 Ulkit => titleSize:large
	emphasis	emphasis {*italic*,**bold**}	Non		emphasis:*italic*
Paragraph	emphasis	emphasis {*italic*,**bold**	Non	Non	emphasis:**bold**

Link	Non	Non	linkProperty	{mutedLink, TextModifier}	linkProperty:mutedLink
			alertLink (réservé que pour les liens dans un élément ayant une propriété alert)	HeadingModifier (réservé que pour les liens dans un titre)	linkProperty:headingModifier linkProperty:alertLink
Button	style	{btn-Color, btn-link, outline-Color , spinner}	style	btn-Color	[[btn-dark]Dark](#) [[outline-primary]Outline](#) [[spinner]Spinner](#)
	size	{default, small,large}	size	{default, small,large}	[[btn-lg]Large](#)
	state	{active, disabled}	Non	Non	[[btn active]ActiveButton](#)
Image	shape	{rounded, circle, thumbnail}	Non	Non	![properties:imageShape:circle)image]url)
	aligning	{center, responsive, float:Alignement(left,top,...)}	Non	Non	![properties:imageAligning:center)image]url)
	Rien	Rien	imageProperty	Non	![properties:(imageProperty)image]url)
List	listProperty	{basicList , HorizontalList}	Non	Non	listProperty:basicList
Table	tableProperty	{basictable, borderedTabl, darkTable, stripedRows , hoverRows}	Non	Non	tableProperty:darkTable
Navigation		{basic, brand, centred, vertical}	Non	Non	[[nav vertical] Vertical navigation](#)

NB: En plus des propriétés particulières, ces concepts peuvent aussi avoir toutes les propriétés générales citées ci-dessus.

- **Les Sections**
  En *markdown*, nous utilisons la syntaxe des *blockQuote* pour représenter nos sections. comme vous pouvez voir ci-dessous:

```
> start section //pour signaler le début d'une section
> des propriétés pour la section
> # Titre de la section
> Tout autre type d'élément que peut contenir une section
(link, paragraph, section, list, blockQoute, image,
codeBlock)
> > start section // une sous section
> > des propriétés de la sous section
> > ## Titre de la sous section
> > son contenu
```

Pour ajouter des propriétés à une section, une ligne doit être réservée pour ces propriétés dans la section car cette ligne ne sera jamais affichée.
En *Bootstrap*, nous transformons les sections en container selon le type de container passé dans les propriétés soit un *fix container* ou un *fluid container*.

`containerType:fluidContainer / fixContainer`

Sur *UIkit*, comme en bootstrap, les sections sont représentées avec la même syntaxe la seule différence est que les sections sont transformées en un seul type qui est *container* et pas besoin de spécifier ce dernier dans les propriétés.

Les éléments ci-dessous sont des concepts supplémentaires

- **Les emphasis** (*italique* et **bold**)

  Comme pour les titres, les accentuations peuvent avoir des propriétés générales et sont ajoutées devant le texte à afficher et précédé par *properties:* pour l'UIkit.

- ***Les types supplémentaires uniques à bootstrap***

  1) **BlockQuote**

     La syntaxe des blockquotes reste la même que celle des sections sauf qu'il n'est pas nécessaire de préciser le début d'un bloc car cela est reconnu par le parseur du fichier *markdown* vers un modèle. Et peuvent avoir toutes les propriétés générales sur une ligne réservée pour celles-ci.

  2) **Listes**

     La syntaxe des listes est la syntaxe normale de *markdown* et l'ajout des propriétés se fait comme suit: pour chaque niveaux de la liste on peut réserver un item qui contiendra les propriétés et qui ne sera jamais, chaque item peut aussi avoir ses propres propriétés générales et peut être également déclaré comme active en faisant précéder son contenu par (*itemProperty: active)*. (c.f: *tests/bstrap.md*)

3) CodeBlock

Ayant composés que des codeLine, un codeLine peut être réservé pour les propriétés générales qu'ils peuvent avoir . (c.f: *tests/bstrap.md*)

4) Navigation

Une navigation est représentée par plusieurs liens dans un même paragraphe dont le premier lien contient les propriétés qui distinguent une navigation des autres représentations utilisées par les liens (button et badge) et peut également contenir des propriétés générales et particulières comme (*brand, basic, vertical, centred*).

```
[(les propriétés générales)[navbar-brand/ vertical/
basic/centred]Navbar brand](#)
[Link 1](#)
[Link 2](#)
[Link 3](#)
```

5) Table

Les tables gardent également la même syntaxe basique de *markdown.* L'ajout des propriétés se fait comme suit:

- Sur la première cellules de la première ligne, nous pouvons précéder son contenu par les propriétés de toutes la table générales et spécifiques, de l'entête et de la cellule
- Sur la première cellule de chaque ligne, nous pouvons faire précéder son contenu par les propriétés de toute la section, de la ligne et de de la cellule.
- Sur n'importe quelle cellules nous pouvons précéder son contenu par les propriétés de la cellules.

Les propriétés d'une table doivent êtres représentées comme fait ci-dessous (un exemple c.f: *tests/bstrap.md*).

```
tabProperties:(tableProps:(), sectionProps:(),
rowProps:(), cellProps:(),tablePropery:basicTable/
borderedTable/ darkTable/stripedRows/hoverRows)
```

6) **Horizontal line**

Une ligne horizontale garde la même structure depuis la syntaxe *markdown* et ne peut avoir aucune propriété.

4. Manuel de Maintenance

Afin de permettre de faire évoluer ce projet, nous décrivons en détail les points majeurs qui sont développés.

a) Description des métamodèles

Pour mener à bien ce travail, nous avons réalisé trois métamodèles qui sont:

1) *Le métamodèle usd*

Le métamodèle usd (Unified Site Descriptor) sert de pivot, il est le point central de toute la chaîne de transformation. Les modèles *markdown* sont transformés vers lui, par la suite transformés vers une technologie cible comme *Bootstrap* ou *UIkit.*
Ce métamodèle est d'un très haut niveau d'abstract car ne fait pas apparaître de concepts techniques et permet ainsi la modélisation des pages et des sites de façon indépendante des technos cibles. Il a été représenté sur deux diagrammes qui sont:

- ***usd:*** Il regroupe les concepts représentant les éléments suivant *titre, paragraph, section, link, image, emphasis, list, blockQoute, codeBlock, navigation et horizontalLine.* (Annexe 3.a : usd)
- ***table:*** Celui-ci représente simplement le concept table avec ses différentes sections, lignes et cellules. (Annexe 3.b : Table usd)

2) *le métamodèle Bootstrap*

Ce métamodèle est de plus bas niveau de conceptions car il propose des concepts similaires à ceux de *Bootstrap*. Nous avons donc représenté tous les concepts du métamodèle usd. Et pour étendre le langage *markdown* de façon à pouvoir représenter les concepts spécifiques à *Bootstrap*, nous avons choisis d'ajouter sur un chaque concept des métadonnées qui regroupent des propriétés à parser pour concrétiser les concepts de *Bootstrap.* (Détails c.f: Manuel d'utilisation)
Le métamodèle a été représenté sur trois diagrammes qui sont:

- ***bstrap:*** Il regroupe les concepts représentant les éléments suivants: titre, *paragraph, section, link, image, button, emphasis, list, blockQoute, codeBlock, navigation, span(que pour représenter les badges) et horizontalLine.* (*Annexe 4.a : Bootstrap*)
- ***table:*** Celui-ci représente simplement le concept table avec ses différentes sections, lignes et cellules. (*Annexe 4.c : Bootstrap Table*)
- ***bstrapProperty:*** Ce dernier regroupe toutes les propriétés de *Bootstrap* que notre logiciel prend en charge dont générales (pour tous les concepts) et particulières (pour un concept unique). (*Annexe 4.b : BootstrapProperty*)

3) ***le métamodèle Ulkit***

Ce métamodèle est également de plus bas niveau comme sur celui de *Bootstrap* il propose des concepts similaires à ceux de *Ulkit.* Nous avons représenté ce métamodèle sur deux diagrammes qui sont:

- ***uikit:*** Il regroupe les concepts représentant les éléments suivants: *titre, paragraph, section, link, image, button et emphasis.* (*Annexe 5.a : UIKit*)
- ***uikitProperty:*** Ce dernier regroupe toutes les propriétés de *Ulkit* que notre logiciel prend en charge dont générales (pour tous les concepts) et particulières (pour un concept unique). (*Annexe 5.b : UIKitProperty*)

b) Explication des transformations

Soucieux d'avoir des transformations structurées et facilement maintenables, nous avons enrichi nos métamodèles par des niveaux d'abstractions générales regroupant tous les éléments ayant une idée générale commune cela facilite donc les transformations ainsi que la génération de code vers une techno cible.
Nos différentes transformations peuvent êtres classées en deux types qui sont:

- ***Les transformations simples***

  Ces sont les concepts qui se transforme directement d'un modèle à un autre

Concept	Markdown	USD	Bootstrap	Ulkit
Titre	Titre	Titre	Titre (avec ses propriétés)	Titre (avec ses propriétés)
Paragraph	Paragraph	Paragraph	Paragraph (avec ses propriétés)	Paragraph (avec ses propriétés)
Image	Image	Image	Image (avec ses propriétés)	Image (avec ses propriétés)
Italic	Emphasis	ItalicEmphasis	Italic (avec ses propriétés)	Italic (avec ses propriétés)
Strong	StrongEmphasis	StrongEmphasis	Strong (avec ses propriétés)	Strong (avec ses propriétés)
Text	Text	Text	Text	Text
Code	Code	Code	Code	Code
Line	HoizontalLine	HoizontalLine	HoizontalLine	Non

CodeBlock	CodeBlock	CodeBlock	CodeBlock (avec ses propriétés)	Non
CodeLine	CodeLine	CodeLine	CodeLine	Non
List	List	ListElement	List (avec ses propriétés)	Non
Table	Table	Table	Table (avec ses propriétés)	Non

- ***Les transformations complexe***

  Ces sont les concepts qui nécessitent certaines modifications avant de passer à un nouveau modèle et qui peuvent dès fois représenter d'autres concepts non reconnus par *markdown*.
  Parmis lesquels nous pouvons trouver :

  1) **Link**

     Un lien écrit avec la syntaxe de *markdown* peut avoir cinq interprétations:
     - Peut rester un lien tout simplement jusqu'à la fin de la chaîne de transformation et devient badge link si il contient une propriété badge.
     - Peut être un lien de référence et dans ce cas la référence est trouvée dès la première transformation (*markdown* vers *USD)* et directement liée au lien pour enfin être transformer en lien normal.
     - Mis dans un paragraphe avec plusieurs autres liens dont l'un contenant des propriétés lui désignant comme une navigation, de *markdown* vers *USD* le paragraphe est transformé en *Navigation* et peut contenir tous les autres liens sauf celui contenant les propriétés car les propriétés seront directement intégrées dans la *Navigation*. Ce concept n'est pris en charge que sur *Bootstrap.*
     - Peut être interprété comme un bouton dès la première transformation de *markdown* vers *USD* quand le lien est représenté avec des propriétés d'un button. De *USD* vers une techno cible, si ces propriétés sont significatives, le button est donc directement décoré par les propriétés mises en œuvre dans les métadonnées.
     - Interprété comme un badge en *Bootstrap* si il contient une propriété *badge* ou *badge-COLOR.*

  2) **BlockQuote**

     Un *BlockQuote* écrit avec la syntaxe de *markdown* peut avoir deux interprétations:

- Peut être interprété comme un *BlockQuote* tout au long de la transformation et n'est pris en charge que sur *Bootstrap.*
- Peut avoir le rôle d'une section mais à condition de signaler le début de la section par "*start section".* De *markdown* vers *USD*, le *BlockQuote* devient une *Section* puis un container dans les deux frameworks (*Bootstrap & Ulkit)*.
  (c.f: *Annexe 2*)

c) La génération des page web vers une techno cible

Sur la partie de génération avec *Acceleo*, nous avons tâché de respecter la notion d'héritage entre les templates de génération tout en respectant le principe ouvert fermé. Ainsi ces fichiers sont réparties dans des modules comme suit:

- **Main**
  Ce module contient le fichier principal ***generate*** qui est appelé lors de la génération du code. Ce fichier fait simplement appel à la génération de la page web.
- **Files**
  Ce module contient tous les templates qui génèrent des fichiers html pour notre un seul car nous générons une page.
- **Common**
  Ce module contient des fichiers utiles à la génération du template contenu dans le module ***Files.*** Leurs noms sont assez explicites pour comprendre leurs utilités À titre d'illustration, le fichier ***handleGeneralProperties*** contient toutes les fonctions auxiliaires qui facilitent la génération des propriétés générales.
- **Request**
  Ce module contient le fichier qui définit les notions générales de l'application tel que le dossier dans lequel sera généré la page web, le chemin de ce dernier et tant d'autres. C'est ici que nous faisons des requêtes afin d'obtenir la plupart des éléments du langage.

5. Historique de développement

Nous avons utilisé GitHub pour développer ce projet. Nous avons exploité le système de branche qu'offre Git pour mieux répartir le travail. Ainsi, chaque fonctionnalité était développée sur une branche que nous fusionnions par la suite avec la branche Master. Vous pouvez trouver le projet sur ce lien et vous aurez accès à l'historique de développement en cliquant sur ***commits*** dans le menu à gauche.

6. Bilan et Perspectives d'amélioration

En somme, le logiciel développé permet la modélisation de tous les attendus du cahier de charge et également certains éléments supplémentaires. Qui plus est, il favorise une meilleure façon d'écriture des fichiers *markdown* tout en respectant sa logique. Ainsi, avec une manière unanime, il est facile de rajouter des propriétés comme des

métadonnées sous forme de texte et qui seront parser uniquement lors de transformations de bas niveau.
Toutefois, nous pouvons envisager de faire évoluer notre logiciel de la manière:

- Faire évoluer la partie *UIkit* pour prendre en charge au moins tous les concepts supplémentaires pris en charge sur le *usd* et le *bstrap.*
- Prendre en charge le plus de concepts possible existants en *markdown* comme (TasksList, Emoji,...).
- Exploiter la syntaxe *markdown* pour d'autres concepts qui ne sont pas évidents sur *markdown* par exemple (les formulaire, checkBox, …).
- Changer la façon dont nous représentons les propriétés des tables car la syntaxe actuelle rend moins compréhensible la logique de *markdown.*

La contrainte du temps ne nous a pas permis de développer toutes ces fonctionnalités, elles feront donc l'objet d'un travail futur.

7. Conclusion

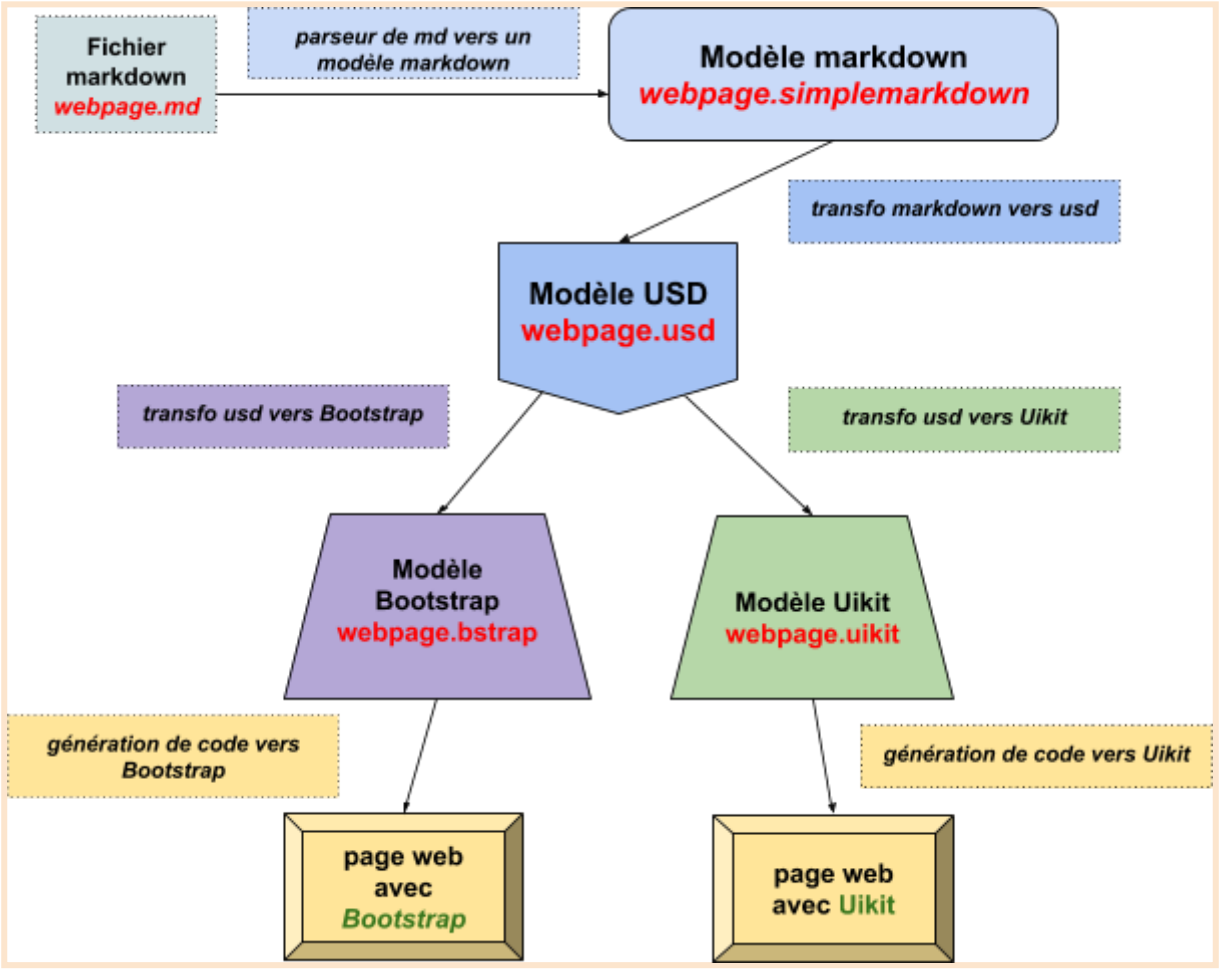
Au cours de ce projet, la prise en main de l'espace de travail a été un défi pour nous au début. L'installation des outils nécessaires ou encore la gestion des versions de JRE entre les différents Eclipse des membres de l'équipe était un peu difficile. Mais avec l'aide du responsable de l'UE nous avons rapidement pu y remédier.

Ce projet nous a permis d'acquérir et de développer des nouvelles connaissances et compétences. Ce projet nous a entre autre permis de :

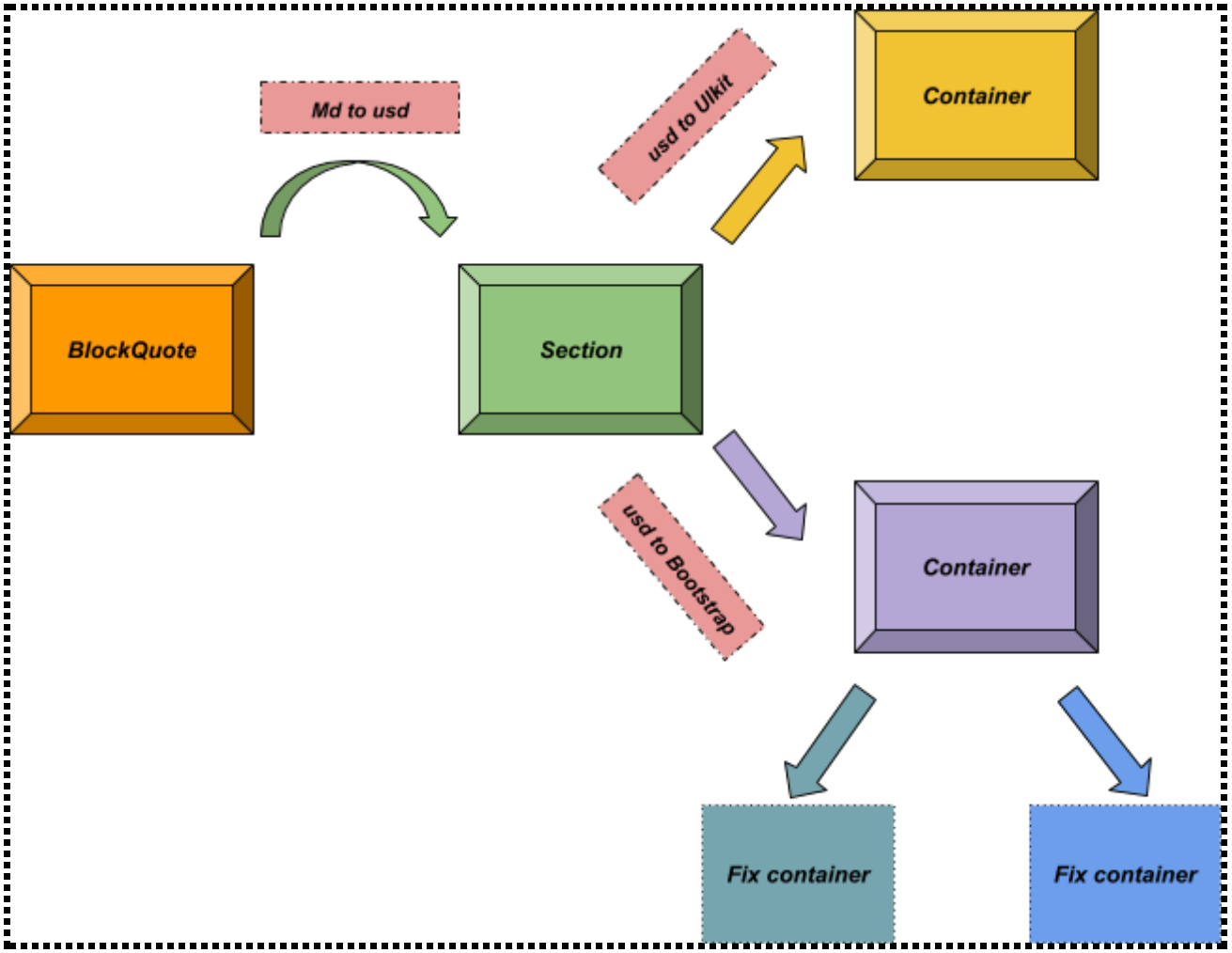
- Mieux comprendre l'approche IDM utilisé pour développer ce logiciel, ce qui nous a été utile dans le cadre de notre cursus académique avec l'UE PJE-A;
- Mieux exploiter ces technos (*markdown, Bootstrap, et UIkit)* pour lesquelles nous n'avons aucune connaissance poussée. Nous sortons donc très construit et avertit sur leur utilisation.
- Mieux exploiter l'outil de versionning *Git* par le partage des tâches entre les membres, l'utilisation des commits régulier dans des branches et les pull requests réguliers une fois qu'une tâche est accomplie.
- Le travail d'équipe et l'exploitation des idées de chacun.

Annexes

Annexe 1 : Chaîne de transformation

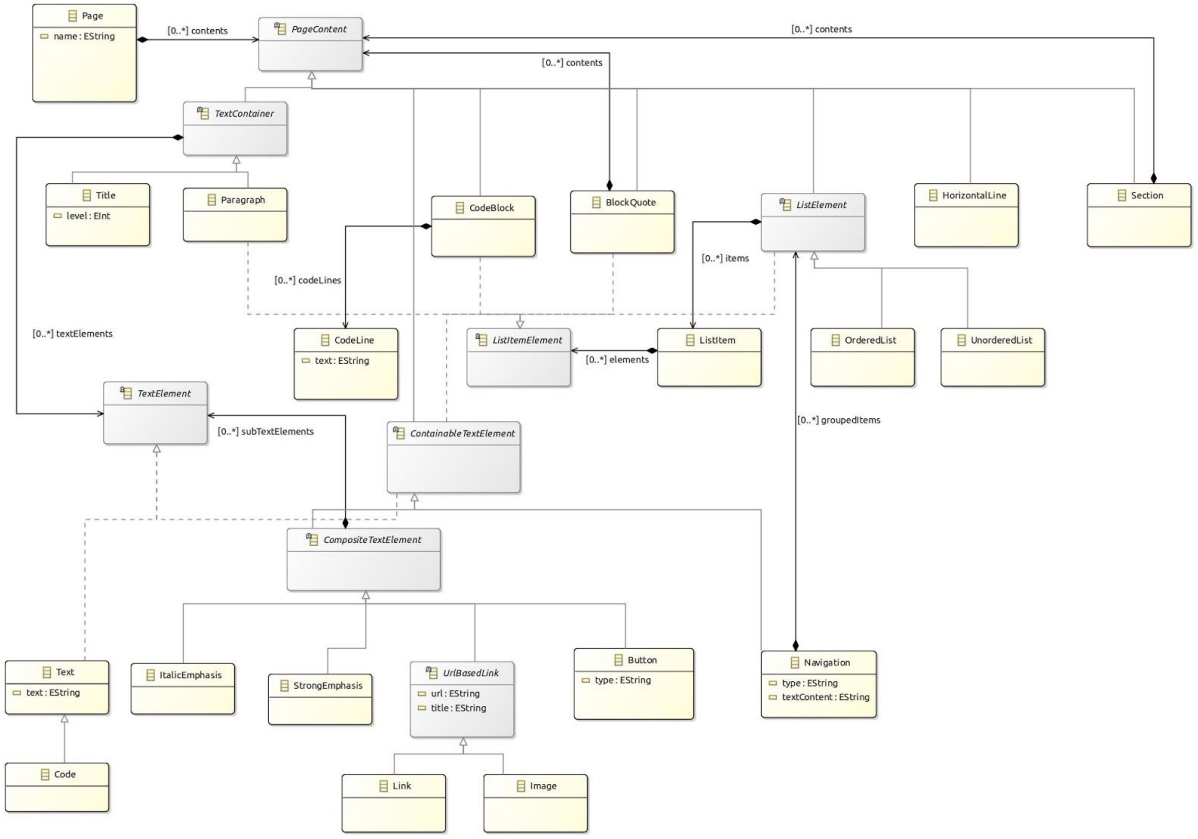


Annexe 2 : Étapes de transformation d'une section

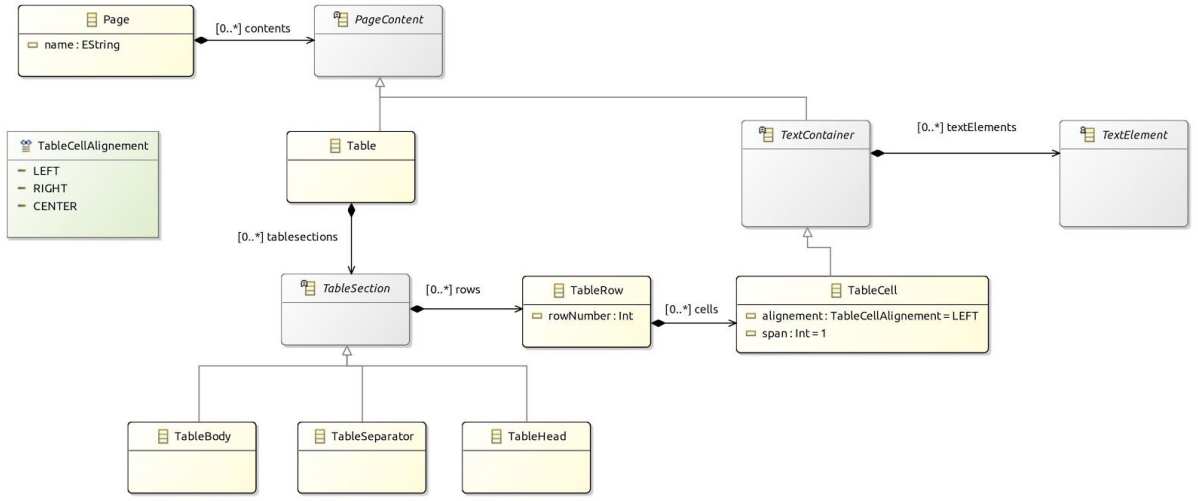


Annexe 3 : Métamodèle *USD*

a. *usd*

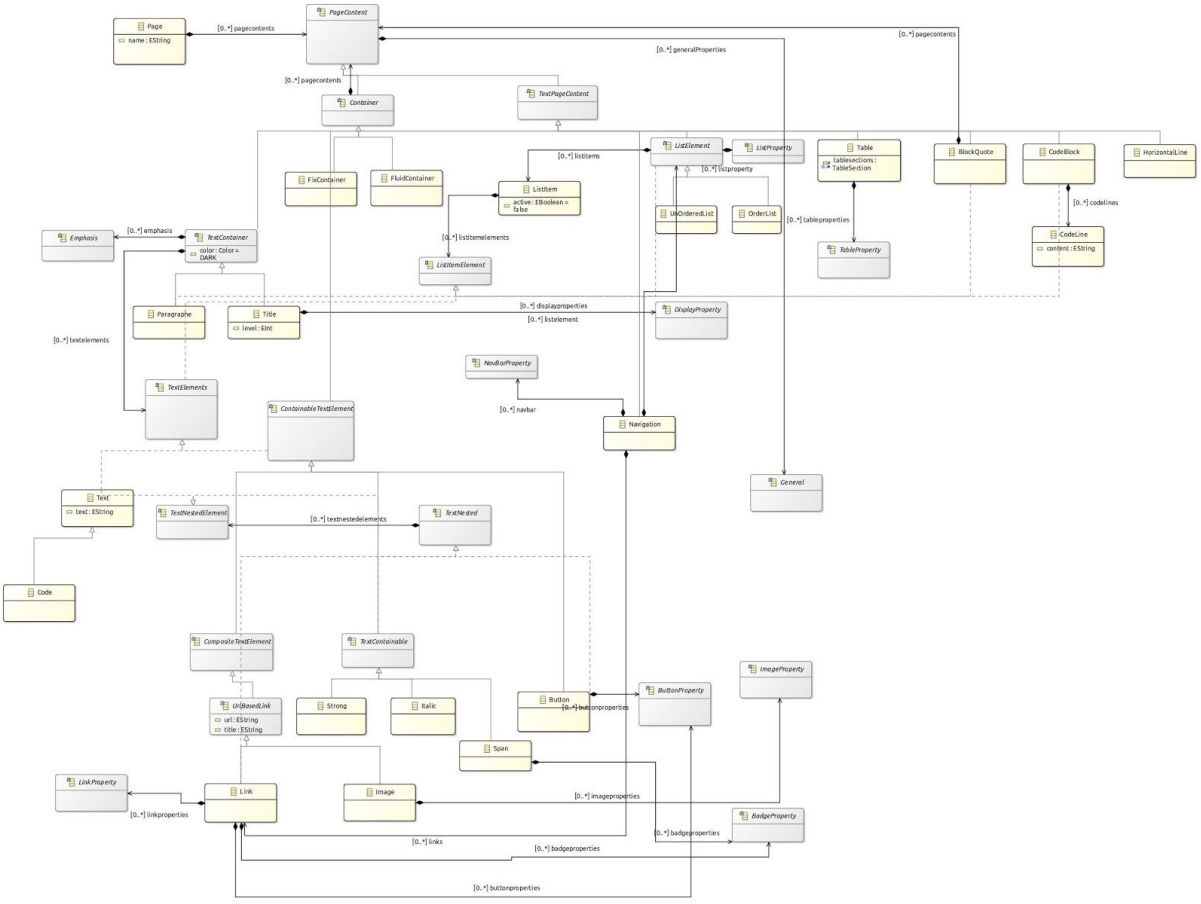


b. *Table usd*

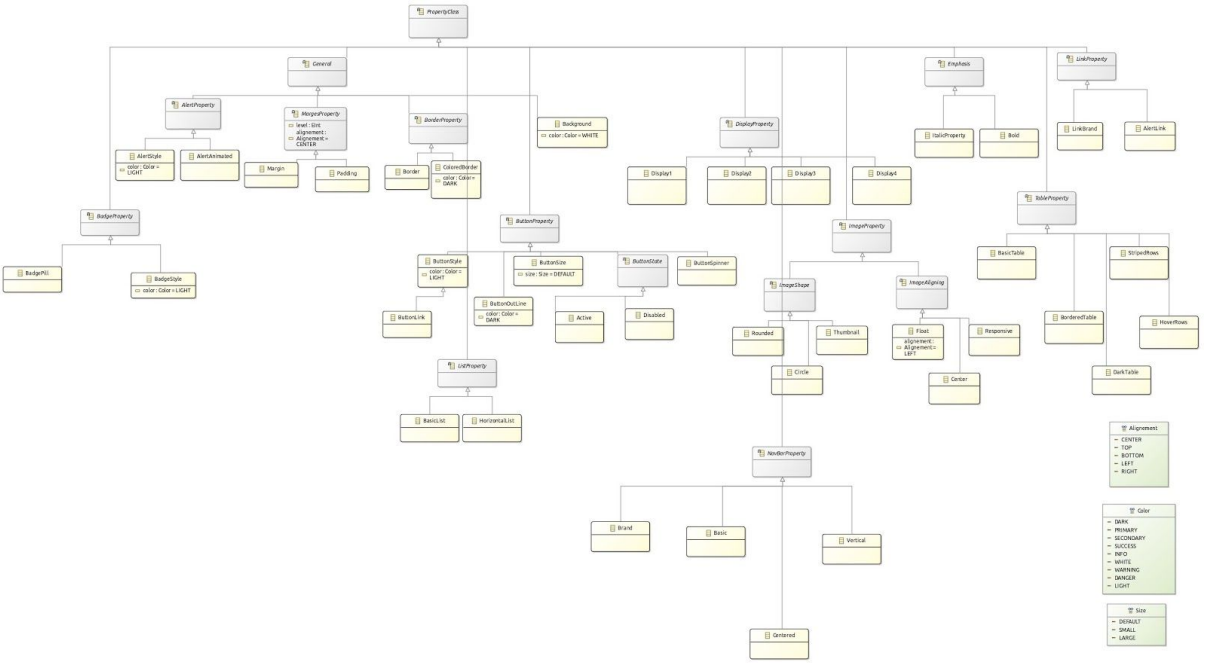


Annexe 4 : Métamodèle *Bootstrap*

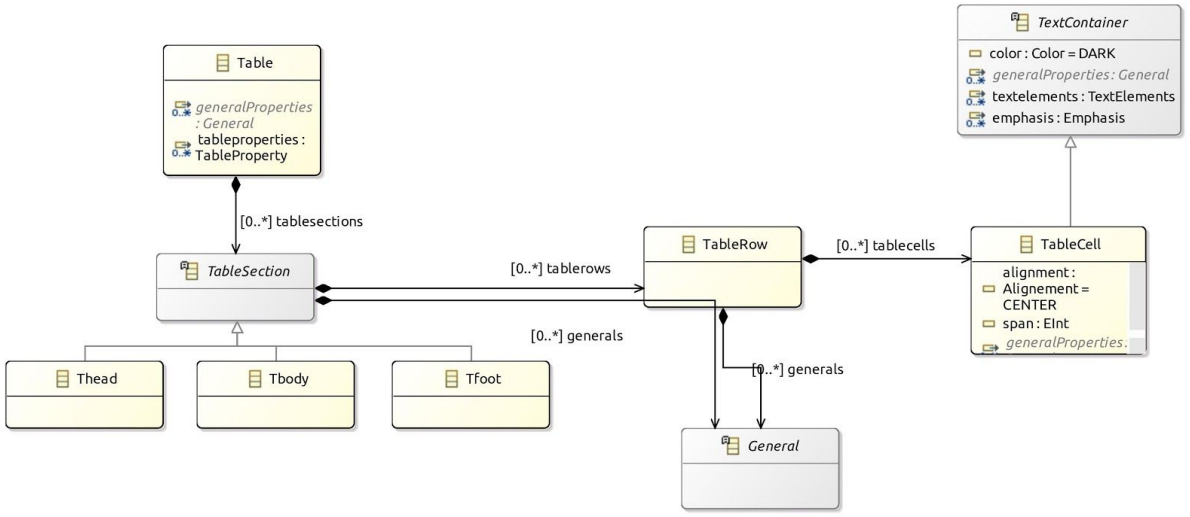
a. ***Bootstrap***



b. ***BootstrapProperty***

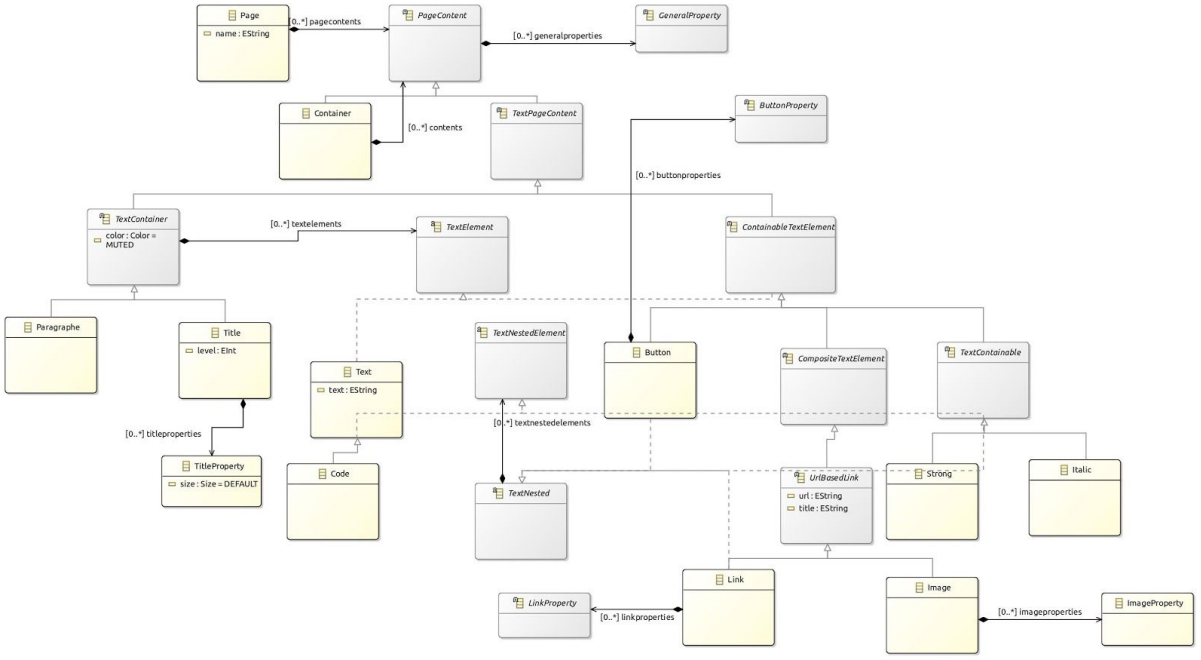


c. ***BootstrapTable***



Annexe 5 : Métamodèle *UIKit*

a. ***UIKit***



b. UIKitProperty