

UML Quicksheet

EPITECH 2013



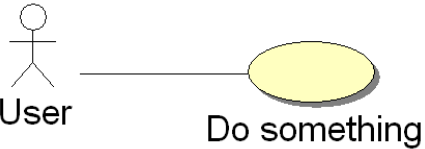
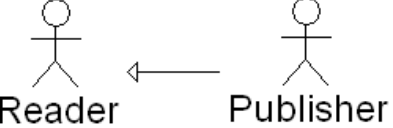
sebatien@migniot.com

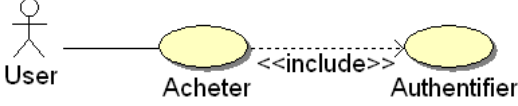

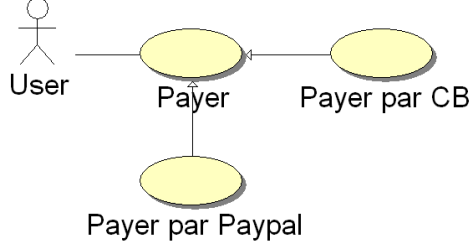
1. LA MÉTHODE

UML sert à représenter des logiciels. UML apporte une méthode


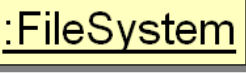
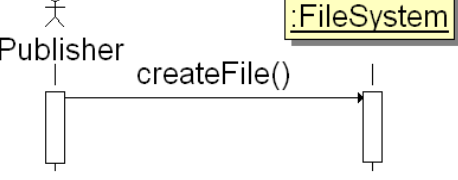
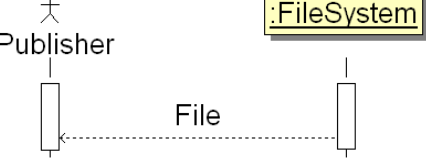
Etape	Description
Use Case	Scenari décrits par les acteurs du logiciel
Séquence	Enchainement de messages entre acteurs et classes
Classes	Diagrammes de classes, comme en C++

2. NOTATIONS USE CASE

Notation	Description
 User	Acteur : un participant au logiciel
 Do something	Use Case : un comportement accessible a un acteur
 User Do something	Association : lien entre un acteur et son usecase accessible
 Reader Publisher	Generalisation : un acteur hérite d'une autre définition d'acteur . Ex : un contributeur est aussi un lecteur sur wikipedia

	<p>Include : Le déroulement d'un usecase inclue <i>obligatoirement</i> le déroulement d'un autre</p>
	<p>Extend : Le déroulement d'un usecase peut être enrichi <i>optionnellement</i> par un autre ou étendu</p>
	<p>Généralisation : Un usecase fils est un usecase parent, par exemple un sous-cas</p>

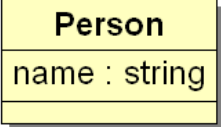
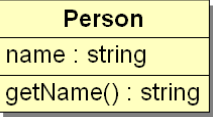
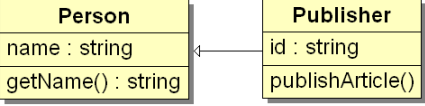
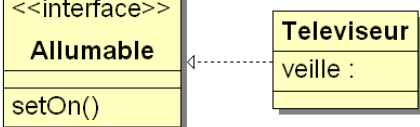
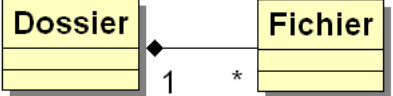

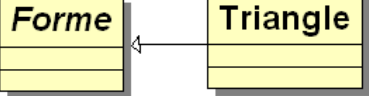
3. NOTATIONS SEQUENCES

Notation	Description
	<p>Acteur : un participant au logiciel <i>issu d'un usecase</i></p>
	<p>Classe : un acteur interne ou logiciel participant à la séquence d'appels</p>
	<p>Message : un participant appelle une fonction sur un autre.</p> <p><i>Séquence système : ceux qui partent des acteurs</i></p>
	<p>Retour : un message de retour, par exemple createFile() renvoie un File.</p>

<pre> sequenceDiagram participant FS as :FileSystem participant Dir as :Directory participant File as :File FS->>Dir: traverse() activate Dir Dir->>File: traverse() activate File File-->>Dir: deactivate File Dir-->>FS: deactivate Dir </pre>	<p>Message reflexif : une fonction récursive ou une fonction déjà décrite ailleurs</p>
<pre> sequenceDiagram participant FS as :FileSystem participant Dir as :Directory participant File as :File FS->>Dir: delete() activate Dir loop Delete item[For each file] Dir->>File: delete() activate File File-->>Dir: deactivate File end Dir-->>FS: deactivate Dir </pre>	<p>Bloc : bloc typé :</p> <ul style="list-style-type: none"> • De type loop pour l'itération • De type alt pour if/then/else/switch • De type opt pour if simple • De type assert pour une condition obligatoire : une exception est lancée si non remplie
<pre> sequenceDiagram actor User as :User participant MS1 as :MailServer participant MS2 as :MailServer User->>MS1: OK, stored activate MS1 MS1->>MS2: sendMail() activate MS2 deactivate MS2 MS1-->>User: deactivate MS1 </pre>	<p>Message asynchrone : Flèche simple avec <i>extrémité non remplie</i> : Le retour peut arriver longtemps après</p>

4. NOTATIONS CLASSES

Notation	Description
	<p>Classe : Une classe, issue d'un usecase ou d'un diagramme de sequences</p>

 <pre> classDiagram class Person { name : string } </pre>	Attribut : Un membre de la classe
 <pre> classDiagram class Person { name : string getName() : string } </pre>	Méthode : Une fonction de la classe
 <pre> classDiagram class Person { name : string getName() : string } class Publisher { id : string publishArticle() } Person < -- Publisher </pre>	Généralisation : Héritage de classe
 <pre> classDiagram class Allumable { <<interface>> setOn() } class Televiseur { veille : } Allumable < .. Televiseur </pre>	Généralisation : Implémentation d'interface
 <pre> classDiagram class Dossier class Fichier Dossier "1" *-- "*" Fichier </pre>	Composition : Contenance physique
 <pre> classDiagram class Lien class Fichier Lien "1" o-- "1" Fichier </pre>	Aggrégation : Contenance par <i>pointeur</i>
 <pre> classDiagram class Forme class Triangle Forme < -- Triangle </pre>	Classe abstraite : En italique