



Master 2 Méthodes Informatiques Appliquées à la Gestion d'Entreprise
parcours Agilité des Systèmes d'Information et E-business

Mémoire de recherche

Des exigences vers des tests d'acceptation automatisés

Auteur :
Khaoula ZITOUN

Encadrants :
Pr. Pascal POIZAT

Version 1.0 du
12 juin 2018

Remerciements

Je tiens tout d'abord à remercier le professeur Pascal Poizat, Professeur des Universités à l'université Paris Nanterre pour ses précieux conseils pour la rédaction de ce mémoire. Je le remercie également pour le temps qu'il m'a consacré pour répondre à mes interrogations.

D'autre part, je souhaite remercier Khadija Machhout, Hajer Zitoun, Sana Zitoun et Adem Maadi pour leur soutien tout au long de cette dernière année de Master.

Je remercie aussi Adil Houssni et Jean-Christophe Claye de m'avoir accordé le temps et l'espace de travail nécessaire à la rédaction de cet écrit.

Enfin mes remerciements vont à l'équipe Europcar de Capgemini de m'avoir motivée pendant la rédaction de ce mémoire. Mention spéciale pour Sanae, Sebastien, Martin, Vincent, Nabil, Jean-Marc, Mohammed, Siham, Abdoulaye, Biljana, Saâd et Edmond.

Table des matières

Introduction	1
1 Etat de l'art	5
1.1 Workflow général et Critères de comparaison	5
1.2 Présentation de l'existant	7
1.2.1 Processus	7
1.2.2 Modélisation des exigences	10
1.2.3 De la modélisation vers les tests d'acceptation - Les framework BDD	27
1.2.4 JDave	29
1.2.5 Conclusion	30
2 Des exigences vers les tests et vice versa	31
2.1 Un DSL pour modéliser le domaine	31
2.2 Générer les tests	31
2.2.1 Génération des méthodes	31
2.2.2 Rédaction des commentaires	31
2.3 Analyser les commentaires	31
2.4 Critique de la solution	31

Table des figures

1.1	Phases ciblées dans ce mémoire	6
1.2	Workflow BPMN de HP ALM	8
1.3	Exemple d'un rapport de run de tests d'un projet sur lequel nous avons travaillé	8
1.4	Exemple d'une table FIT et de la fixture générée	9
1.5	Diagramme entité/association de notre cas	12
1.6	Exemple de diagramme de classe	14
1.7	Exemple de diagramme de flux de données	15
1.8	Exemple de DFD de niveau 0	16
1.9	Exemple de DFD de niveau 1	17
1.10	Diagramme de cas d'utilisation de notre cas	19
1.11	Exemple de fiche descriptive d'une scénario	20
1.12	Exemple de diagramme de séquence	22
1.13	Exemple de diagramme d'activité	23
1.14	Exemple de diagramme d'exigences [dia]	25
1.15	Exemple de syntaxe Gherkin	28
1.16	Exemple de scénario Groovy	29
1.17	Exemple de rapport EasyB	29

Liste des sigles et acronymes

ASK	<i>Amplitude Shift Keying</i>
AWGN	<i>Additive White Gaussian Noise</i>
BABG	Bruit Additif Blanc Gaussien
BCJR	<i>Bahl, Cocke, Jelinek, Raviv</i>
BER	<i>Binary Error Rate</i>
BFDM	<i>Biorthogonal Frequency Division Multiplexing</i>

Introduction

Les projets informatiques sont généralement le résultat de l'implantation d'un ou des besoins exprimés.

Depuis 1994, les résultats du rapport CHAOS du Standish Group sont mis à jour chaque année et montrent les principales raisons d'échec et de succès d'un projet au sein d'un panel représentatif d'entreprises américaines. Les différentes études montrent que les projets qui se terminent dans le temps et le budget avec un périmètre fonctionnel livré conforme au périmètre initialement défini représentent 20%. Les projets qui se terminent en ayant respecté le temps ou le budget avec un périmètre fonctionnel livré légèrement différent du périmètre initial représentent 50%. Enfin les projets qui ont été abandonnés en cours de projet pour diverse raisons représentent 30%. Il est donc à souligner que le taux d'échec des projets reste important. Par ailleurs, ce rapport montre que parmi les facteurs d'échec, 44,1% d'entre eux sont relatifs aux exigences (exigences incomplètes, manque d'implication des utilisateurs, changement sur les exigences et les spécifications). D'autre part, les facteurs de succès d'un projet relatifs aux exigences représentent 37,1% (implication des utilisateurs, définition claire des exigences, attentes réalistes). [Stéphane Badreau 2014] Cette étude nous permet donc d'affirmer que l'implication des parties prenantes ainsi que la définition et la gestion des exigences jouent un rôle important dans l'avenir d'un projet.

L'implication des parties prenantes permet notamment de réduire l'écart qui existent entre celles-ci [Barret R.Bryant 2011]. En effet, un écart de communication entre les experts du domaine et les ingénieurs en logiciel a été observé [Troyer 2014], comme cela a été rapporté par des chercheurs. [C. Coulin 2005]. Définir des exigences pour ensuite les implanter peut donc devenir périlleux. Les experts domaines, n'ayant pas forcément des compétences techniques mais métier, et les ingénieurs en logiciel ayant des compétences techniques mais n'appréhendant pas entièrement tous les domaines, il est donc nécessaire de trouver un moyen de définir les exigences tout en s'assurant que les experts puissent les rédiger et les ingénieurs les comprendre.

D'après la définition du IEEE et du CMMi une exigence est : «

1. *Condition ou capacité nécessaire à un utilisateur pour résoudre un problème ou atteindre un objectif.*
2. *Condition ou capacité qui doit être assurée par un produit pour satisfaire à un*

contrat, une norme, une spécification ou à d'autres documents imposés formellement.

3. *Une représentation documentée de cette condition ou capacité telle que définie en 1. ou 2.*

»

Il existe plusieurs niveaux d'exigences : les besoins des utilisateurs, les exigences métier qui sont défini à partir des besoins et les exigences produit qui traduisent la solution fonctionnelle et technique. De plus, il existe plusieurs types d'exigences : les exigences fonctionnelles, les exigences non fonctionnelles, les exigences de contrainte. Dans ce mémoire nous nous axerons sur les exigences fonctionnelles. Après avoir recueilli le besoin des utilisateur il est important de définir les exigences. L'objectif de la définition des exigences est de les exprimer de manière compréhensible par tous. [Ges] Les exigences sont exprimées de différentes manières mais pas toujours compréhensibles par tous. Une solution à ceci problématique est l'ingénierie des domaines. Cette dernière est un élément clé pour que l'ingénierie des exigences soit efficace [Barret R.Bryant 2010].

La spécification des exigences liées au domaine nécessite un langage d'exigences propre au domaine : les Domain Specific Requirement Language (DSRL). Ces langages permettent de spécifier les exigences en terme d'abstractions de domaine d'application. [Barret R.Bryant 2010] Certaines approches sont génériques et sont des solutions à de nombreuses problématiques générales de certains domaines. Il arrive toutefois que ces approches ne répondent pas à des problèmes spécifiques à un domaine. Une approche spécifique fournit une bien meilleure solution pour un ensemble plus restreint de problèmes. [Arie van Deursen 2000] Les Domain Specific Model rendent la modélisation des exigences moins compliquée et réduit l'effort d'apprentissage pour les scientifiques et favorise le génie logiciel dans les projets. [Barret R.Bryant 2011]

«Le but de l'ingénierie de domaine est d'identifier, de modéliser, de construire, de cataloguer et de diffuser des artefacts qui représentent les points communs et les différences au sein d'un domaine. [Iris Reinhartz-Berger 2011, Arie van Deursen 2000] ». La modélisation de ces artefacts peuvent se faire de différentes façons : par des diagrammes UML, SysML, les DSL (Domain Specific Language), etc que nous présenterons dans la première partie de ce mémoire.

Bien qu'il existe des méthodes pour modéliser le domaine métier qui nous permettrait de définir les exigences fonctionnelles, il n'en reste pas moins que des tests d'acceptation sont nécessaires pour assurer la satisfaction du client. Ces tests sont souvent créés en fonction d'une spécification des exigences et servent à vérifier que les obligations contractuelles sont respectées. [Maurer 2004] Il semble nécessaire de déterminer comment valider des exigences orientées domaine en passant par les tests. Il existe beaucoup de recherches et d'implantations d'outils sur des tests basés sur des modèles, la dérivation de tests à partir d'un modèle. [Felderer 2010] Dans ce document nous nous intéresseront aux tests relatifs

à la recette tels que les tests d'acceptation, appelés aussi tests fonctionnels. Les tests d'acceptation peuvent être spécifiés de plusieurs façons : depuis les user stories basées sur la compréhension de textes suivis jusqu'aux langages formels. Parce que l'exécution des tests d'acceptation est longue et coûteuse, il est hautement souhaitable d'automatiser ce processus. *L'automatisation des tests d'acceptation donne une réponse objective lorsque les exigences fonctionnelles sont remplies. [Maurer 2004]*

La définition des exigences fonctionnelles et la vérification de ces dernières jouent donc un rôle important dans la réussite d'un projet. Nous souhaitons donc dans ce mémoire répondre à la problématique suivante : **comment à partir d'une spécification d'exigences générer des tests d'acceptation en mettant le domaine métier au coeur de la démarche ?**

Dans ce document nous présenterons dans une première section comment définir des exigences, quelques outils de suivi, les différentes méthodes de modélisation du domaine ainsi que les techniques de génération de tests.

Chapitre 1

Etat de l'art

Dans cette section de ce mémoire, nous allons tout d'abord présenter les éléments existants permettant à partir d'exigences fonctionnelles de procéder aux tests d'acceptation. Nous présenterons les outils qui existent sur le marché qui tentent d'automatiser ce processus. Puis dans une seconde partie nous présenterons les différentes façons de modéliser des exigences fonctionnelles de sorte à pouvoir générer des tests d'acceptation. Enfin dans la dernière partie de ce chapitre nous proposerons, compte tenu des éléments précédemment présentés, une ébauche de solution à notre problématique que nous présenterons plus en détail dans le second chapitre de ce document.

1.1 Workflow général et Critères de comparaison

Les modèles de gestion de projets les plus répandus sont le cycle en V et la méthode agile scrum. Dans ces modèles, le besoin est recueilli, les exigences sont définies et celles-ci sont vérifiées et testées. Dans le cycle en V schématisé dans la figure, nous nous intéressons aux phases "Analyse des besoins", "Spécifications", "Recette" et "Test de validation". Dans la méthode agile scrum, nous nous intéressons au contenu du product backlog, au contenu du sprint backlog et des user stories car ces étapes représentent la définition des exigences et les tests à réaliser à la fin de chaque sprint. En combinant les deux méthodes et en ne ciblant que les phases projet qui nous intéressent, à savoir les phases d'analyse et de test, nous proposons un workflow générique d'élucidation des exigences et de validation :

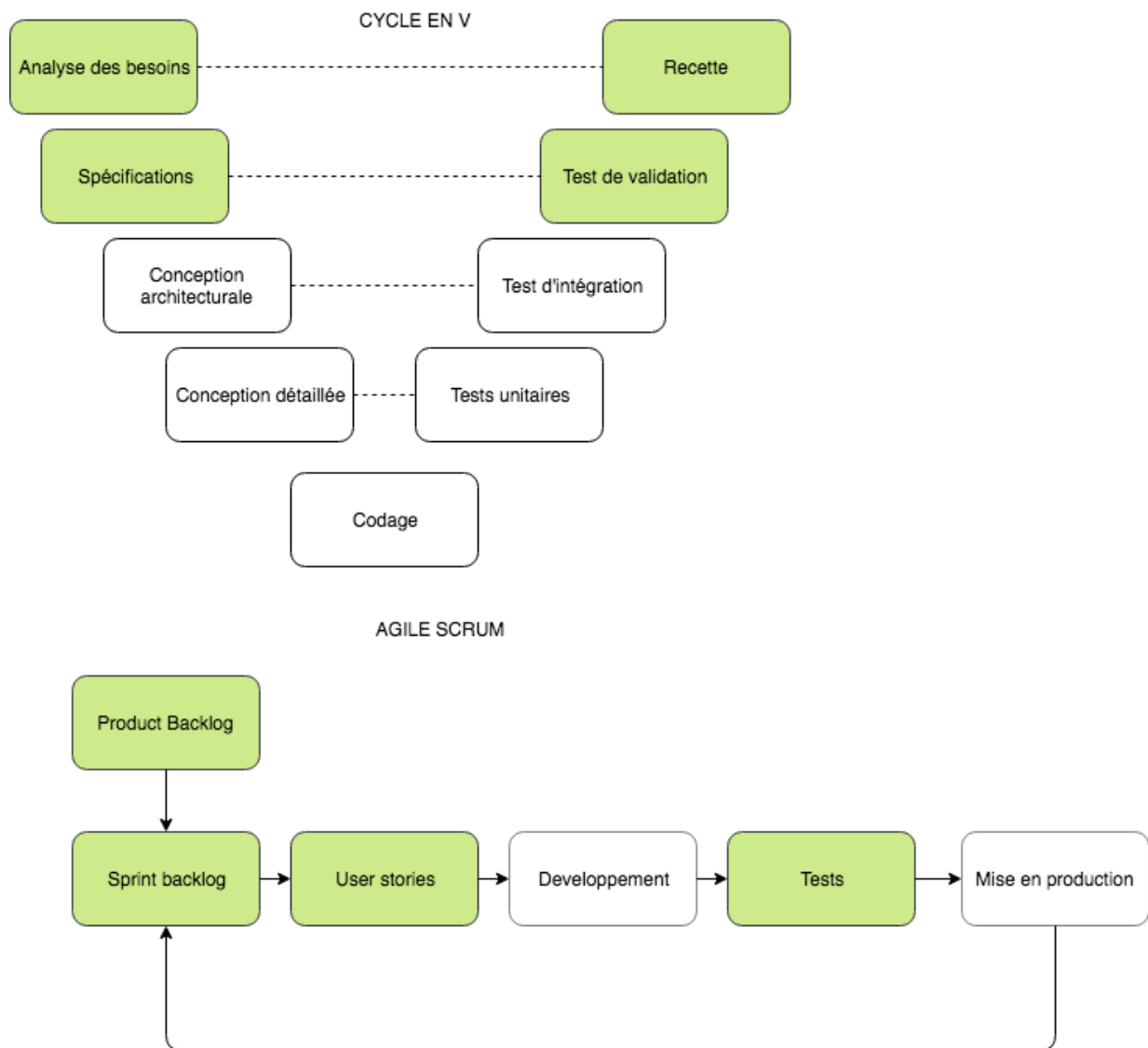


FIGURE 1.1 – Phases ciblées dans ce mémoire

Dans cet état de l'art, nous comparerons et analyserons les différents outils et méthodes qui répondent entièrement ou partiellement à notre problématique selon les critères suivants :

Description des critères de comparaison
Les exigences peuvent être écrites par n'importe quelle partie prenante
Les exigences peuvent être lues par n'importe quelle partie prenante
Les spécificités fonctionnelles du domaine peuvent être correctement exprimées
Les exigences sont exprimées de façon à pouvoir générer des tests clairs
Il existe un moyen d'automatiser les tests
Les tests automatisables sont des tests d'acceptation
Le resultat des tests est disponible au niveau des exigences

1.2 Présentation de l'existant

1.2.1 Processus

Des exigences aux tests

Il existe plusieurs outils sur le marché qui permettent aux clients de renseigner les tests d'acceptation à dérouler ainsi que les résultats attendus pour satisfaire les exigences. Parmi ces outils, HP ALM (Application Lifecycle Management) anciennement Quality Center. Il existe d'autres outils relativement similaire à HP ALM mais nous décidons de présenter celui-ci car nous avons récemment eu l'occasion de travailler avec au sein de Capgemini pour un client.

HP Application Lifecycle Management (ALM)

Il existe plusieurs outils sur le marché qui permettent aux clients de renseigner les tests d'acceptation à dérouler ainsi que les résultats attendus pour satisfaire les exigences. Parmi ces outils, HP ALM (Application Lifecycle Management) anciennement Quality Center. Nous décidons de présenter cet outil car nous avons eu l'occasion de l'utiliser en entreprise en collaboration avec un client. Cet outil permet la collaboration des différentes parties prenantes. En effet, les chefs de projets peuvent planifier le projet en déterminant le nombre le périmètre de cycles de releases, la spécification des exigences est également réalisée par les business analystes, les testeurs peuvent dérouler les tests, etc. Tout ceci permet d'avoir une vision globale du projet. La figure 1.2 représente une instance des parties que nous avons ciblées du cycle en V.

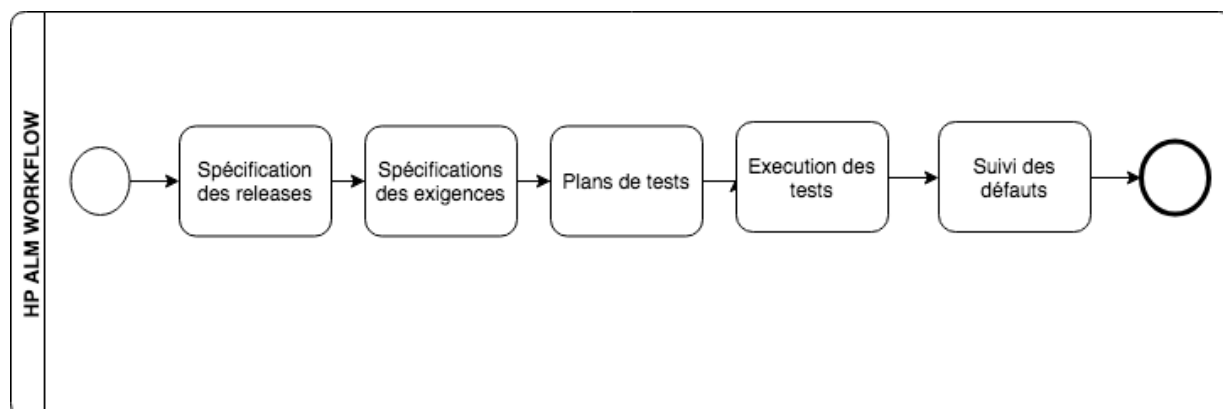


FIGURE 1.2 – Workflow BPMN de HP ALM

La rédaction des exigences et des plans de tests se fait manuellement, en langage naturel. Les personnes en charge de dérouler les plans de test lisent les différentes étapes, les exécutent et comparent les résultats obtenus aux résultats attendus.

Last Run Report					Steps Details
Step Name	Status	Exec Date	Exec Time	For	
Step 1	✓ Passed	03/05/2017	08:12:14		
Step 2	✓ Passed	03/05/2017	08:12:14		
Step 3	✓ Passed	03/05/2017	08:12:15		
Step 4	✓ Passed	03/05/2017	08:12:15		
Step 5	✓ Passed	03/05/2017	08:12:15		
Step 6	✓ Passed	03/05/2017	08:12:15		
Step 7	✓ Passed	03/05/2017	08:12:16		
					<p>Description: Create two or more fines for same RA in Table Fines (evtl. per Dataload)</p> <p>Expected: Fines are loaded in Table</p> <p>Actual:</p>

FIGURE 1.3 – Exemple d'un rapport de run de tests d'un projet sur lequel nous avons travaillé

Lorsqu'un comportement anormal est détecté pendant les tests, un "defect" est créé pour tracker le défaut de l'application.

Bien que ce type d'outil permette d'avoir une visibilité entre les exigences et le résultat des tests il n'en reste pas moins qu'il n'existe pas d'automatisation des tests à partir des exigences.

Framework For Integrated Test (FIT)

Des frameworks comme FitNess[fit] basés sur FIT (Framework For Integrated Test) tentent de répondre à la problématique de l'automatisation des tests à partir des exigences. FIT est un moteur qui traite chaque table en utilisant le code fixture à laquelle la table

correspond. «Une fixture est un morceau de code qui permet de fixer un environnement logiciel pour exécuter des tests logiciels. Cet environnement constant est toujours le même à chaque exécution des tests. Il permet de répéter les tests indéfiniment et d'avoir toujours les mêmes résultats [fix].» Fitness permet d'afficher les résultats des tests FIT. Les tests FIT sont donc des représentations tabulaires des exigences. A partir de ces tables, des tests sont générés [Maurer 2004]. Les attentes des clients sont comparés aux résultats réels. [fit]

fit.ActionFixture		
enter	select	1
check	title	Akila
check	artist	Toure Kunda
enter	select	2
check	title	American Tango
check	artist	Weather Report
check	album	Mysterious Traveller
check	year	1974
check	time	3.70
check	track	2 of 7

```
public class Browser
    extends ActionFixture {
    ...
    public void select(int i) {
        MusicLibrary.
            select(MusicLibrary.library[i-1]);
    }

    public String title() {
        return MusciLibrary.looking.title;
    }

    public String artist() {
        return MusciLibrary.looking.artist;
    }
    ...
}
```

FIGURE 1.4 – Exemple d'une table FIT et de la fixture générée

FitNess permet de d'écrire et de lire les exigences de façon simplifiée mais pas simple.. Les tests sont facilement mis en oeuvre par les développeurs d'après l'article [Maurer 2004]. Cependant «Pour rédiger une série complète de tests, il est souhaitable d'acquérir des connaissances et de l'expérience dans les domaines du test et de l'ingénierie logicielle (par exemple, un ingénieur en assurance qualité pourrait travailler en étroite collaboration avec le client)». FitNess permet également de générer des tests automatiquement. Toutefois, FIT ne semble pas gérer toutes les exigences : un article [Maurer 2004] montre que l'hypothèse selon laquelle 100% des exigences implémentées auraient des tests FIT correspondants est erroné. Les informations non pertinentes sont plus difficiles à inclure dans des tableaux bien structurés que dans des documents rédigés en prose.

Ravenflow - Bender RBY - Rational Rhapsody

<https://searchsoftwarequality.techtarget.com/answer/Tools-that-generate-test-cases-from-software-requirements>

Il existe des outils tel que Ravenflow qui permettent de générer des cas de tests à partir de formes d'exigences. Ce type d'outil se base sur des cas d'utilisation dans un format structuré et identifie chaque chemin à travers ces derniers.

Il existe également des outils plus rigoureux tel que Bender RBT qui développe des flux logiques de graphes cause/effet à partir d'une méthode structurée de documentation des spécifications fonctionnelles et identifie l'ensemble minimum de tests.

Les outils d'analyse d'état, tels que Rational Rhapsody, dessinent un diagramme d'état à partir de descriptions structurées des différents états d'une unité, qui peuvent être un périphérique ou un programme. L'état détermine les comportements que l'unité doit et ne doit pas présenter, y compris ce qui déclenche la transition vers un autre état spécifié. Les cas de test sont alors définis pour exercer chaque chemin d'états, démontrant les comportements qu'il devrait présenter et assurant que ceux qu'il ne devrait pas présenter ne se produisent pas.

Bien que chacune de ces méthodes présente certains avantages, il est important de noter qu'aucune n'est susceptible d'intercepter des problèmes liés à des exigences métier incorrectes ou négligées, des facteurs de qualité, d'autres moyens d'invoquer un processus décisionnel ou des décisions de conception de spécifications fonctionnelles.

Ces différents outils se basent sur une structure qui modélise les exigences. Nous proposons donc de d'abord modéliser les exigences pour en générer les tests.

1.2.2 Modélisation des exigences

Nous avons vu dans la section précédente que les processus permettant de passer des exigences aux tests étaient insuffisants du point de vue de notre problématique. Nous proposons de présenter dans un premier temps différentes manières de modéliser les exigences fonctionnelles. Un modèle est une représentation abstraite de la réalité. Un outil est un formalisme, une langue permettant d'exprimer un modèle. Une exigence peut être décrite par plusieurs modèles. Par conséquent, nous proposons de comparer les différents modèles selon les critères suivants :

1. La modélisation doit pouvoir exprimer l'exigence
2. La modélisation doit être lisible par n'importe quelle partie prenante
3. La modélisation doit pouvoir être écrite par les personnes qui expriment l'exigence
4. La modélisation doit permettre de générer des tests
5. Les tests générés doivent être des tests fonctionnels

6. N'importe quel domaine peut modéliser avec cet outil.

7. Le domaine est au coeur de la modélisation

Nous proposons un cas pratique pour pouvoir comparer les outils : une application pour réserver une table à un restaurant depuis son téléphone.

Modélisation structurelle

Le diagramme entité association

Ce diagramme représente graphiquement des entités et les interactions entre elles à travers des relations (ou associations). Une entité est un objet comme par exemple "Utilisateur", "Restaurant", etc. Il est souvent utilisé pour modéliser les relations entre les tables d'une base de données. Il repose sur différents concepts : les entités, les associations, les attributs d'entités ou d'association et les cardinalités. Ce modèle permet d'identifier et de caractériser les objets du domaine et d'établir leurs liens.

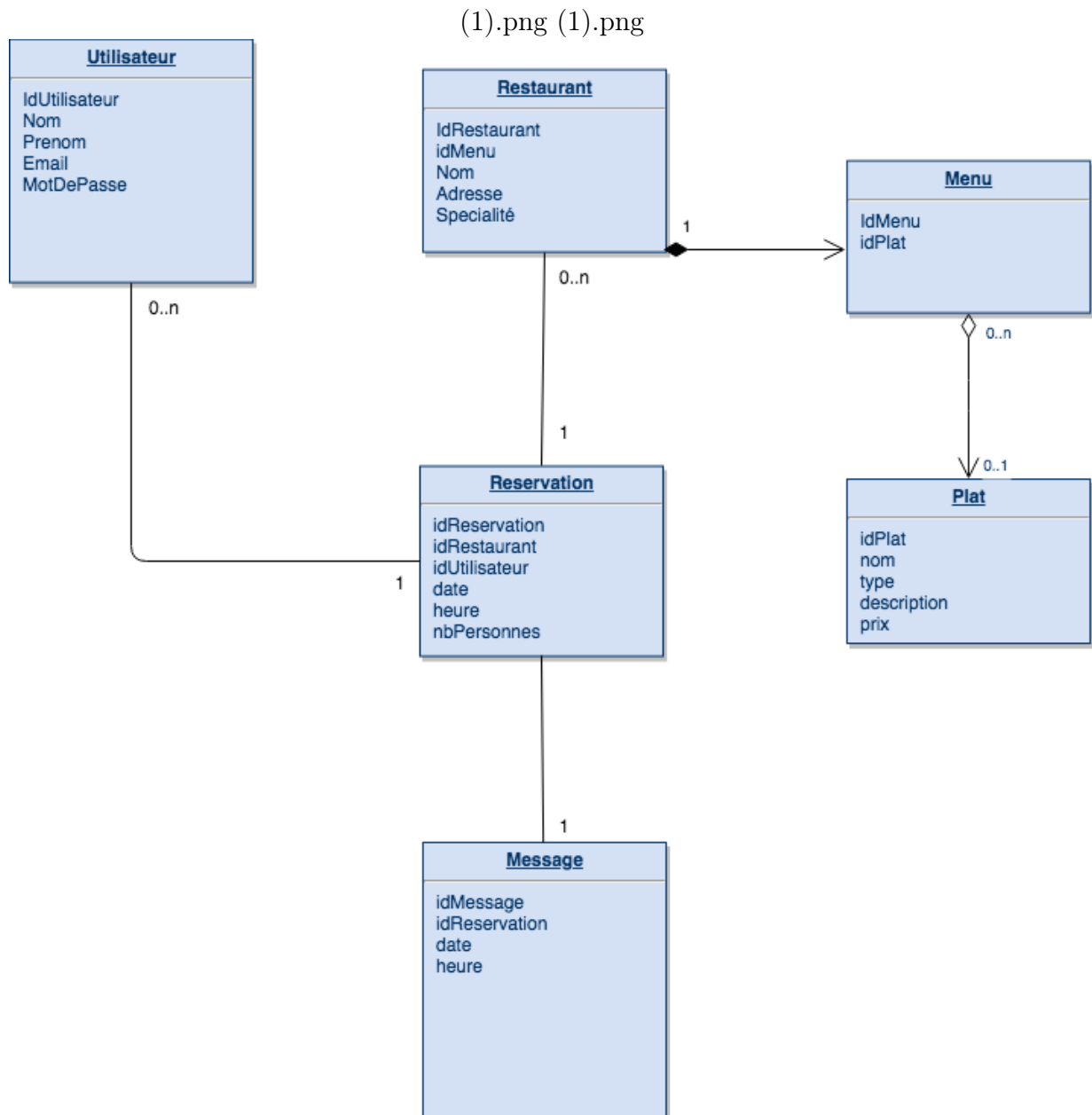


FIGURE 1.5 – Diagramme entité/association de notre cas

Si nous comparons ce diagramme à notre grille d'exigences nous notons qu'il permet d'exprimer des exigences fonctionnelles mais de façon incomplète. En effet, dans ce diagramme, il n'y a aucune précision quant à comment se connecter, sur les conditions d'annulation, sur la façon dont le restaurant doit recevoir sa réservation, etc. De plus, ce diagramme est facilement lisible par tout le monde, et le temps d'apprentissage pour produire ce type de diagramme semble acceptable au vu de sa simplicité. Toutefois, ce type de diagramme ne permet pas de générer des tests et les représentations graphiques proposées ne

sont pas orientées domaine.

Critère	Répond au critère
La modélisation doit pouvoir exprimer l'exigence	NON
La modélisation doit être lisible par n'importe quelle partie prenante	OUI
La modélisation doit pouvoir être écrite par les personnes qui expriment l'exigence	OUI
La modélisation doit permettre de générer des tests	NON
Les tests générés doivent être des tests fonctionnels	NON
N'importe quel domaine peut modéliser avec cet outil.	OUI
Le domaine est au coeur de la modélisations	NON

TABLE 1.1 – Évaluation du diagramme entité-association selon les critères défini

Le diagramme de classe

Ce diagramme représente les classes et les interfaces ainsi que les relations entre elles. Une classe est représentée par un rectangle, possède des attributs, une visibilité, une multiplicité. Les classes peuvent hériter l'une de l'autre, s'agréger ou l'une peut composer l'autre.

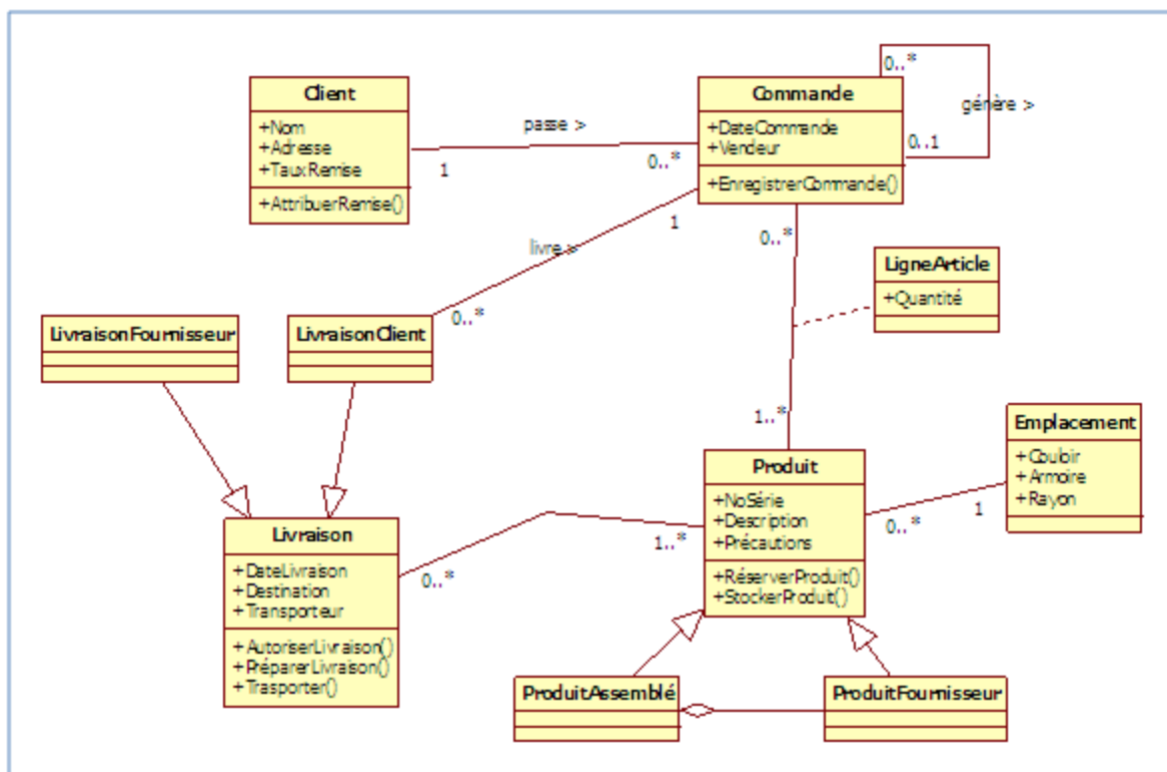


FIGURE 1.6 – Exemple de diagramme de classe

Bien que ce diagramme offre un niveau de détail plus important que le diagramme d'utilisation concernant le comportement de la solution, il reste toutefois que la notation n'a aucun lien avec le domaine. D'autre part, un diagramme de classe ne semble pas être simple pour toutes les parties prenantes.

Critère	Répond au critère
La modélisation doit pouvoir exprimer l'exigence	NON
La modélisation doit être lisible par n'importe quelle partie prenante	OUI
La modélisation doit pouvoir être écrite par les personnes qui expriment l'exigence	OUI
La modélisation doit permettre de générer des tests	NON
Les tests générés doivent être des tests fonctionnels	NON
N'importe quel domaine peut modéliser avec cet outil.	OUI
Le domaine est au coeur de la modélisations	NON

TABLE 1.2 – Évaluation du diagramme de classe selon les critères défini

Les diagrammes de flux de données

Ce diagramme popularisé à la fin des années 1970 permet de représenter graphiquement comment l'information circule dans un processus ou un système. Ce type de diagramme est composé de :

- Flux de données avec des étiquettes
- Des transformations de données (en cercle ou bulles) pour schématiser les processus
- Des magasins de données (lignes horizontales parallèles)
- Entités extérieures au système (rectangles)

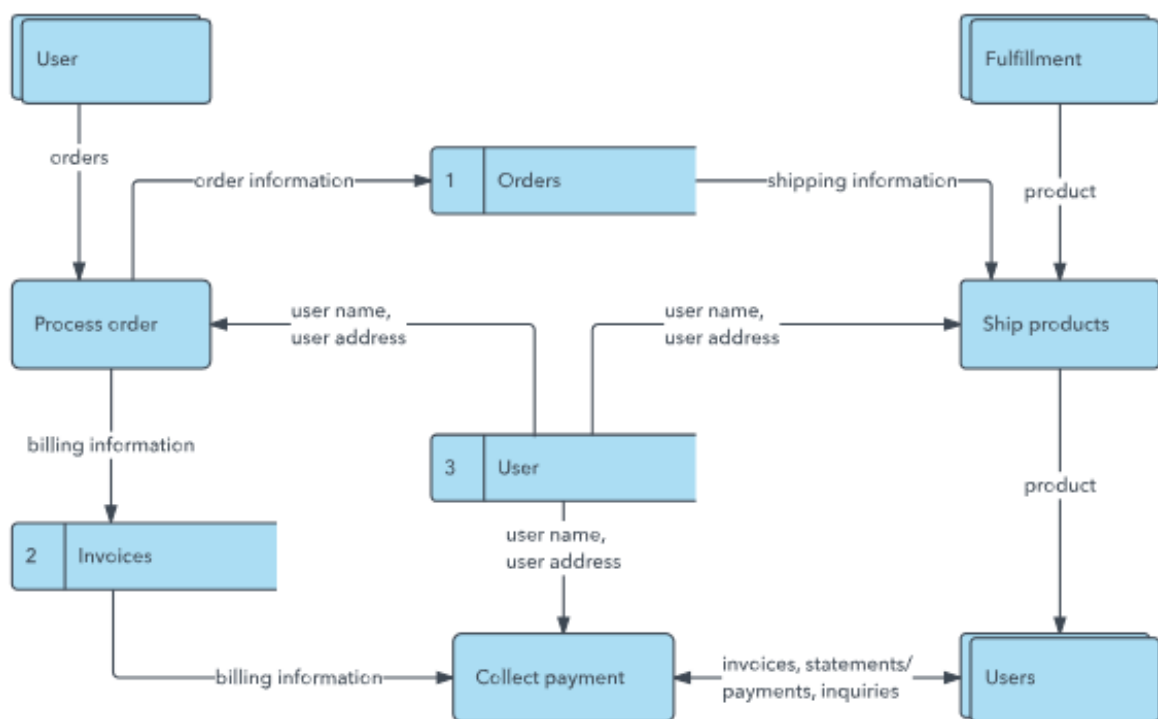


FIGURE 1.7 – Exemple de diagramme de flux de données

Les diagrammes de flux de donnée proposent des niveaux de détail numérotés 0, 1 ou 2 et vont parfois jusqu'à 3 ou plus.

Niveau 0

Ce diagramme est appelé diagramme de contexte. Il représente une vue globale du système, montrant un processus unique de haut avec ses relations externes. Ce diagramme est généralement compréhensible par toutes les parties prenantes mais de part son aspect général ne fournit pas assez de détails pour spécifier des tests d'acceptation.

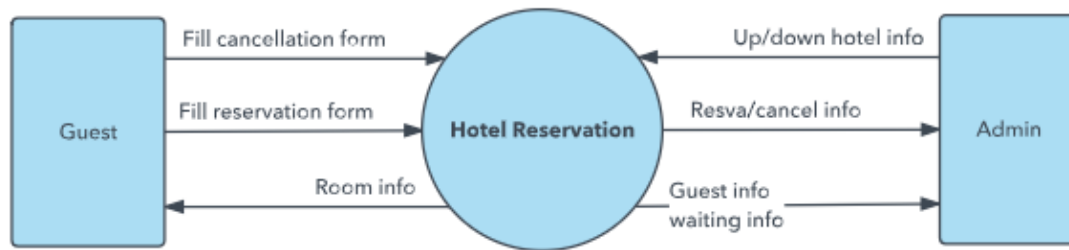


FIGURE 1.8 – Exemple de DFD de niveau 0

Niveau 1

Ce diagramme fournit des détails aux diagramme de context. Les fonctions principales du système sont mises en évidence. Le processus unique est découpé en sous processus.

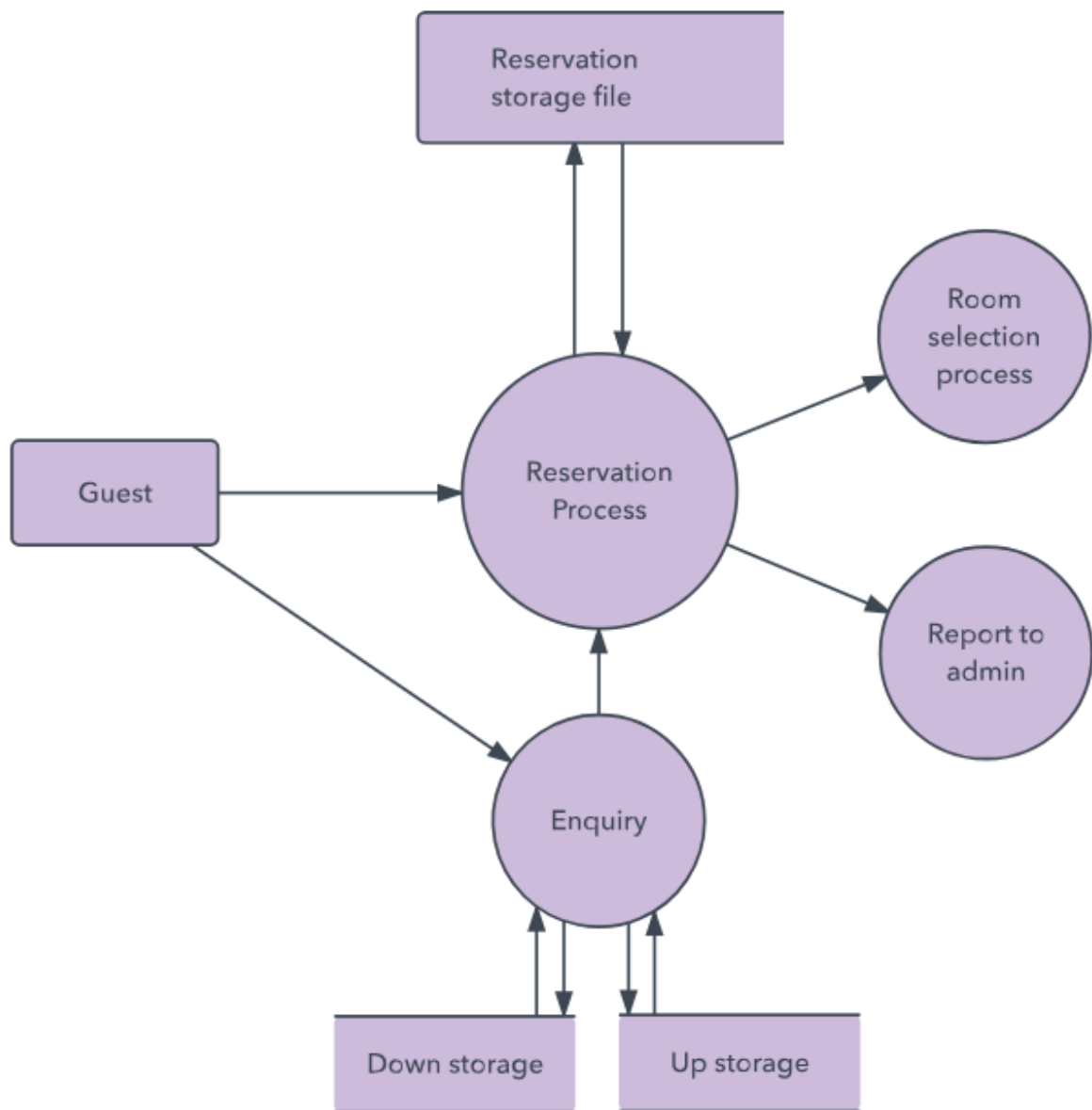


FIGURE 1.9 – Exemple de DFD de niveau 1

Niveau 2

Ce diagramme entre encore plus en détail dans la description des processus.

Les niveaux de détails 3 et 4 sont possibles à mettre en place mais restent assez rares.

Avec un diagramme assez détaillé, les développeurs peuvent l'utiliser pour commencer à rédiger du pseudocode. Un DFD peut fournir un bon point de départ pour modéliser les

exigences mais lors du système réel, il n'est pas suffisant pour les testeurs. Les DFD sont même moins précis que le langage naturel pour les développeurs. Ce type de modélisation est intéressant pour visualiser les fonctions et les interfaces mais n'est pas suffisant pour éliciter correctement une exigence. [Elizabeth Hull 2010]

Critère	Répond au critère
La modélisation doit pouvoir exprimer l'exigence	NON
La modélisation doit être lisible par n'importe quelle partie prenante	OUI
La modélisation doit pouvoir être écrite par les personnes qui expriment l'exigence	OUI
La modélisation doit permettre de générer des tests	NON
Les tests générés doivent être des tests fonctionnels	NON
N'importe quel domaine peut modéliser avec cet outil.	NON
Le domaine est au coeur de la modélisations	NON

TABLE 1.3 – Évaluation du diagramme de flux de données selon les critères défini

Les modélisations qui représentent une vue beaucoup trop globale et seulement structurale des exigences semblent ne pas être suffisantes pour remplir tous les critères que nous avons défini. Il semble également nécessaire de représenter le comportement des exigences, d'ajouter une partie dynamique à l'expression de ces exigences.

Modélisation comportementale

Le diagramme de cas d'utilisation

Ce diagramme permet de décrire les différents acteurs du système et les fonctionnalités que chacun d'entre eux doit pouvoir réaliser. Chaque fonctionnalité est généralement représentée par un ovale dans lequel l'action est décrite. Les traits entre les acteurs et les cas d'utilisations sont des associations et/ou des inclusions.

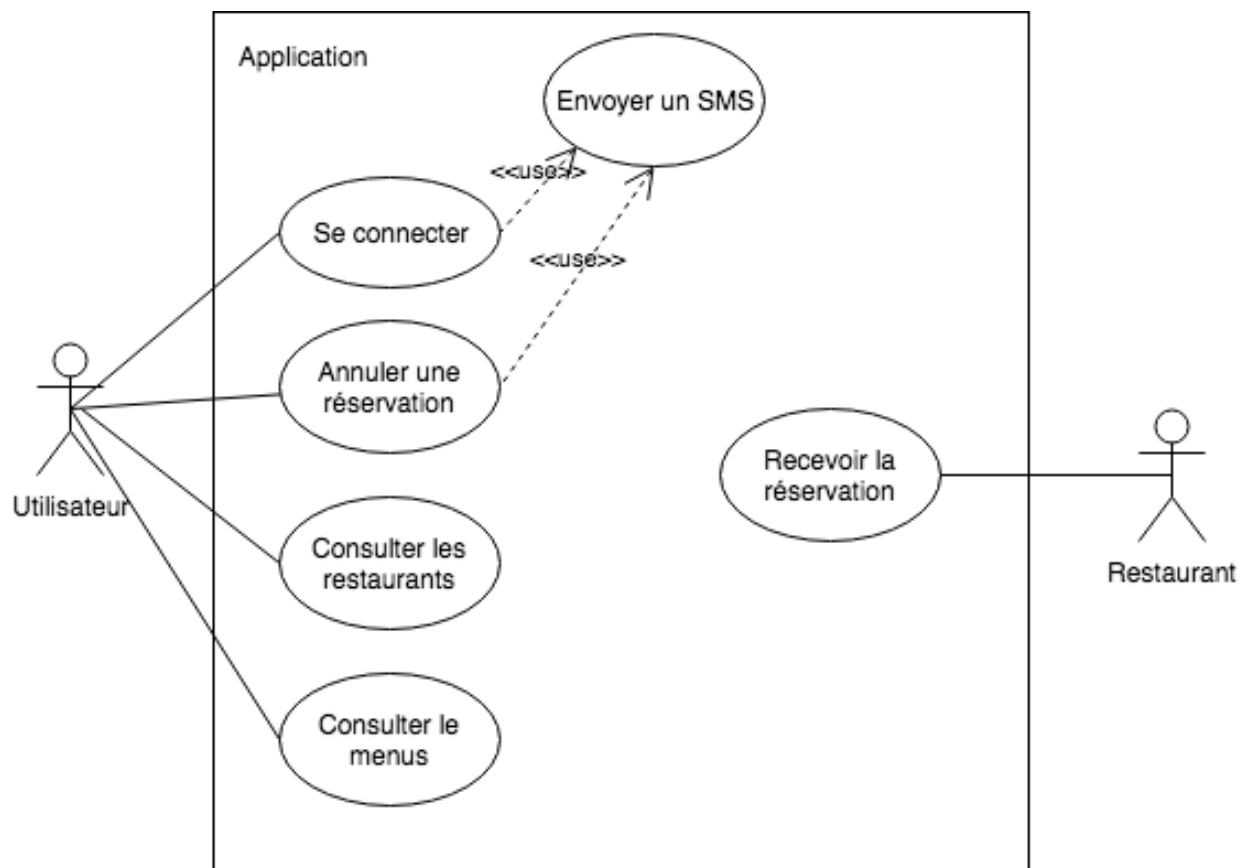


FIGURE 1.10 – Diagramme de cas d'utilisation de notre cas

Ce diagramme permet d'avoir une vision très global de ce que doit pouvoir faire un acteur mais ne prend pas en compte les différents scénarios qui peuvent exister et qui peuvent être en eux même des exigences à spécifier.

Critère	Répond au critère
La modélisation doit pouvoir exprimer l'exigence	NON
La modélisation doit être lisible par n'importe quelle partie prenante	OUI
La modélisation doit pouvoir être écrite par les personnes qui expriment l'exigence	OUI
La modélisation doit permettre de générer des tests	NON
Les tests générés doivent être des tests fonctionnels	NON
N'importe quel domaine peut modéliser avec cet outil.	OUI
Le domaine est au coeur de la modélisations	NON

TABLE 1.4 – Évaluation du diagramme d'utilisation selon les critères défini

Le scénario

Le scénario est une description textuelle d'un cas d'utilisation. Décrire un cas d'utilisation permet notamment de déterminer quelle action arrive avant ou après une autre action, de bien comprendre comment la fonctionnalité doit se dérouler ou encore de connaître les contraintes relatives à ce cas. Un scénario est représenté par une fiche descriptive composé de l'identification, de la description du scénario, la et les post-conditions et les compléments.

Cas n° 1
<p>Nom : Consulter catalogue produit (package « Gestion des achats »)</p> <p>Acteur(s) : Acheteur (client ou commercial)</p> <p>Description : La consultation du catalogue doit être possible pour un client ainsi que pour les commerciaux de l'entreprise.</p> <p>Auteur : Carina Roels</p> <p>Date(s) : 10/11/2013 (première rédaction)</p> <p>Pré-conditions : L'utilisateur doit être authentifié en tant que client ou commercial (Cas d'utilisation « S'authentifier » - package « Authentification »)</p> <p>Démarrage : L'utilisateur a demandé la page « Consultation catalogue »</p>

FIGURE 1.11 – Exemple de fiche descriptive d'une scénario

Avec ce type de fiches descriptive que l'on peut standardiser, nous avons un début de

formalisme d'une exigence. Toutefois, la description de l'exigence reste purement textuelle. Il est donc compliqué à partir de cette fiche de générer entièrement un test. Il existe une façon plus schématique et moins textuel de représenter un scénario : le diagramme de séquence.

Critère	Répond au critère
La modélisation doit pouvoir exprimer l'exigence	OUI
La modélisation doit être lisible par n'importe quelle partie prenante	OUI
La modélisation doit pouvoir être écrite par les personnes qui expriment l'exigence	OUI
La modélisation doit permettre de générer des tests	NON
Les tests générés doivent être des tests fonctionnels	NON
N'importe quel domaine peut modéliser avec cet outil.	OUI
Le domaine est au coeur de la modélisations	OUI

TABLE 1.5 – Évaluation du scénario selon les critères défini

Le diagramme de séquence

Le diagramme de séquence permet de décrire de façon détaillée les interactions entre les différentes instances du système pour un scénario d'utilisation donné.

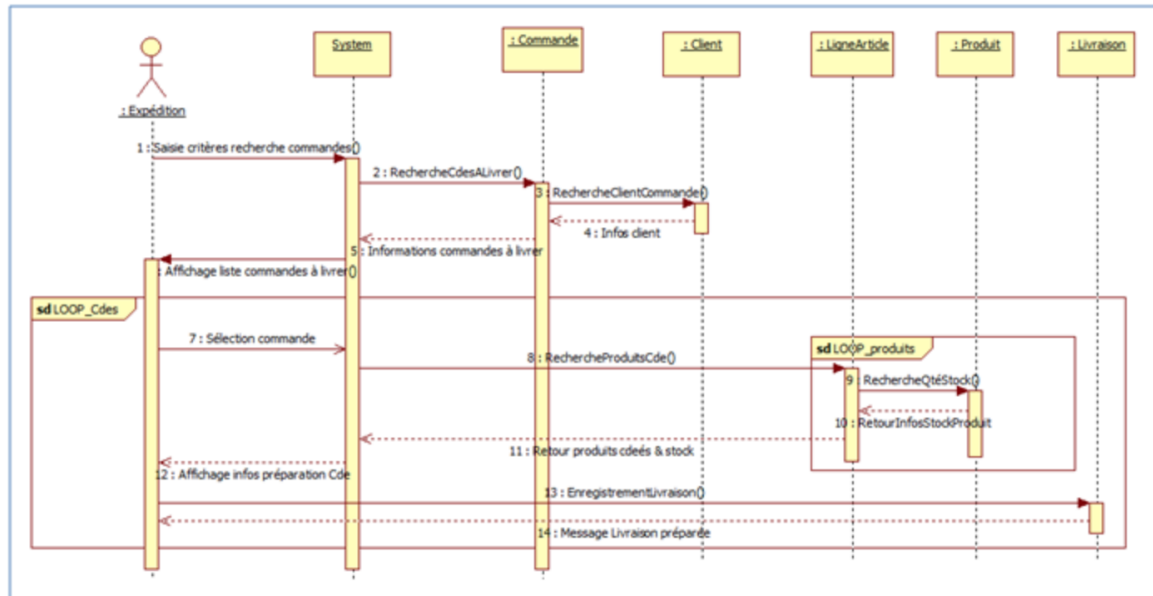


FIGURE 1.12 – Exemple de diagramme de séquence

Il existe plusieurs approches qui ont été développées dans plusieurs articles de recherches pour générer et dériver des cas de tests à partir d'un diagramme de séquence. [M. Dhineshkumar 2014, Vikas Panthi 2012]

Critère	Répond au critère
La modélisation doit pouvoir exprimer l'exigence	OUI
La modélisation doit être lisible par n'importe quelle partie prenante	NON
La modélisation doit pouvoir être écrite par les personnes qui expriment l'exigence	NON
La modélisation doit permettre de générer des tests	OUI
Les tests générés doivent être des tests fonctionnels	OUI
N'importe quel domaine peut modéliser avec cet outil.	OUI
Le domaine est au coeur de la modélisations	NON

TABLE 1.6 – Évaluation du diagramme de séquence selon les critères défini

Le diagramme d'activité

Ce diagramme décrit le déroulement des différentes actions successives d'un système sans utiliser les objets ce qui permet plus de clarté pour les clients qui ont une vision domaine et non technique.

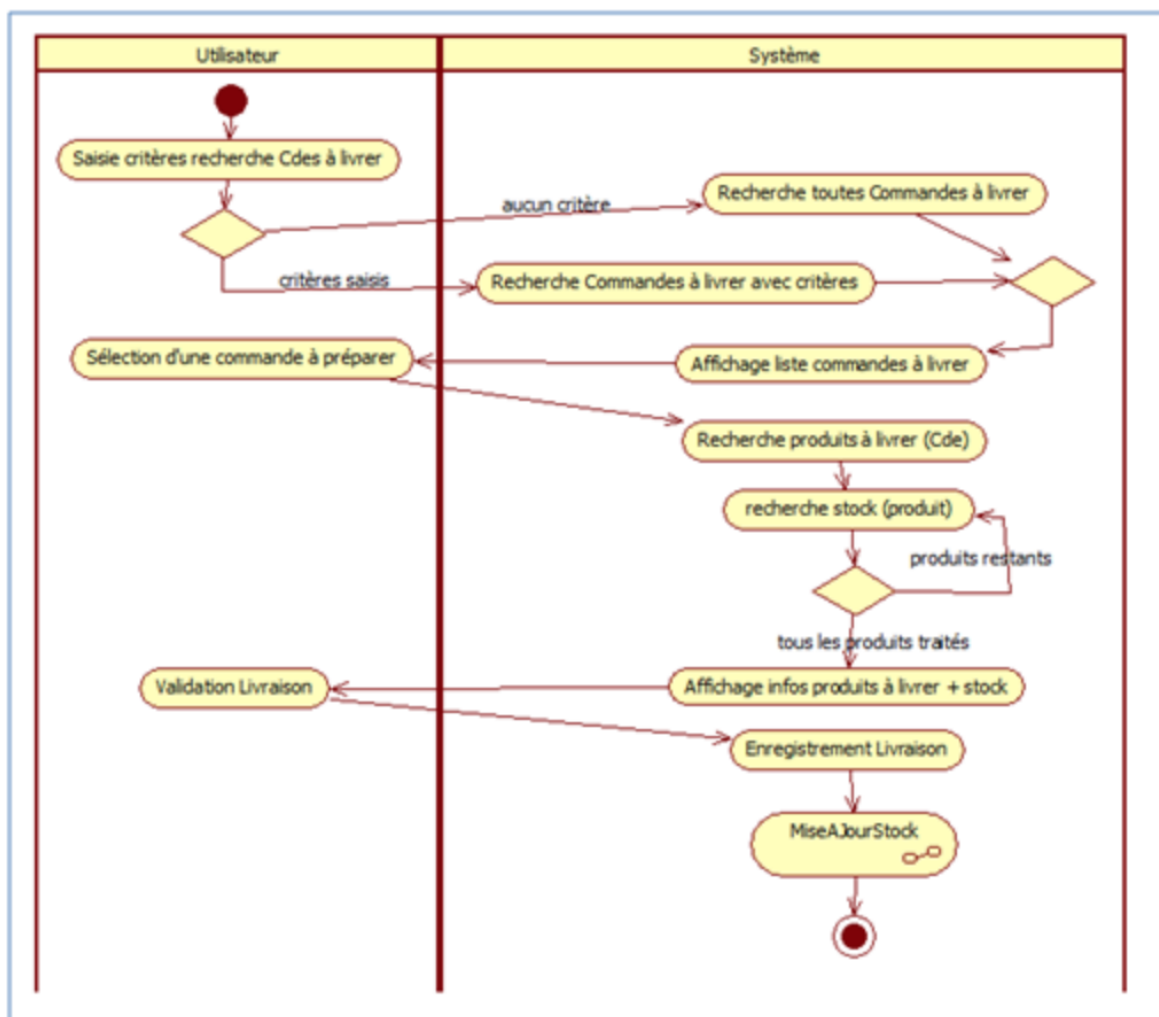


FIGURE 1.13 – Exemple de diagramme d'activité

Des approches ont également été proposées [Kundu 2008] pour générer des cas de test à partir de ce types de diagrammes.

Critère	Répond au critère
La modélisation doit pouvoir exprimer l'exigence	OUI
La modélisation doit être lisible par n'importe quelle partie prenante	OUI
La modélisation doit pouvoir être écrite par les personnes qui expriment l'exigence	NON
La modélisation doit permettre de générer des tests	OUI
Les tests générés doivent être des tests fonctionnels	OUI
N'importe quel domaine peut modéliser avec cet outil.	OUI
Le domaine est au coeur de la modélisations	NON

TABLE 1.7 – Évaluation du diagramme d'activité selon les critères défini

SysML : le diagramme d'exigence

SysML doit permettre à des acteurs de corps de métiers différents de collaborer autour d'un modèle commun pour définir un système. [sys]. A l'instar d'UML, SysML est un langage constitué de plusieurs diagrammes. La majorité de ces derniers sont communs à UML. Par conséquent, nous présenterons dans cette sous partie que le diagramme d'exigences. Il s'agit d'un diagramme qui décrit les fonctions du logiciel. Il décrit les spécifications fonctionnelles.

Diagramme des exigences

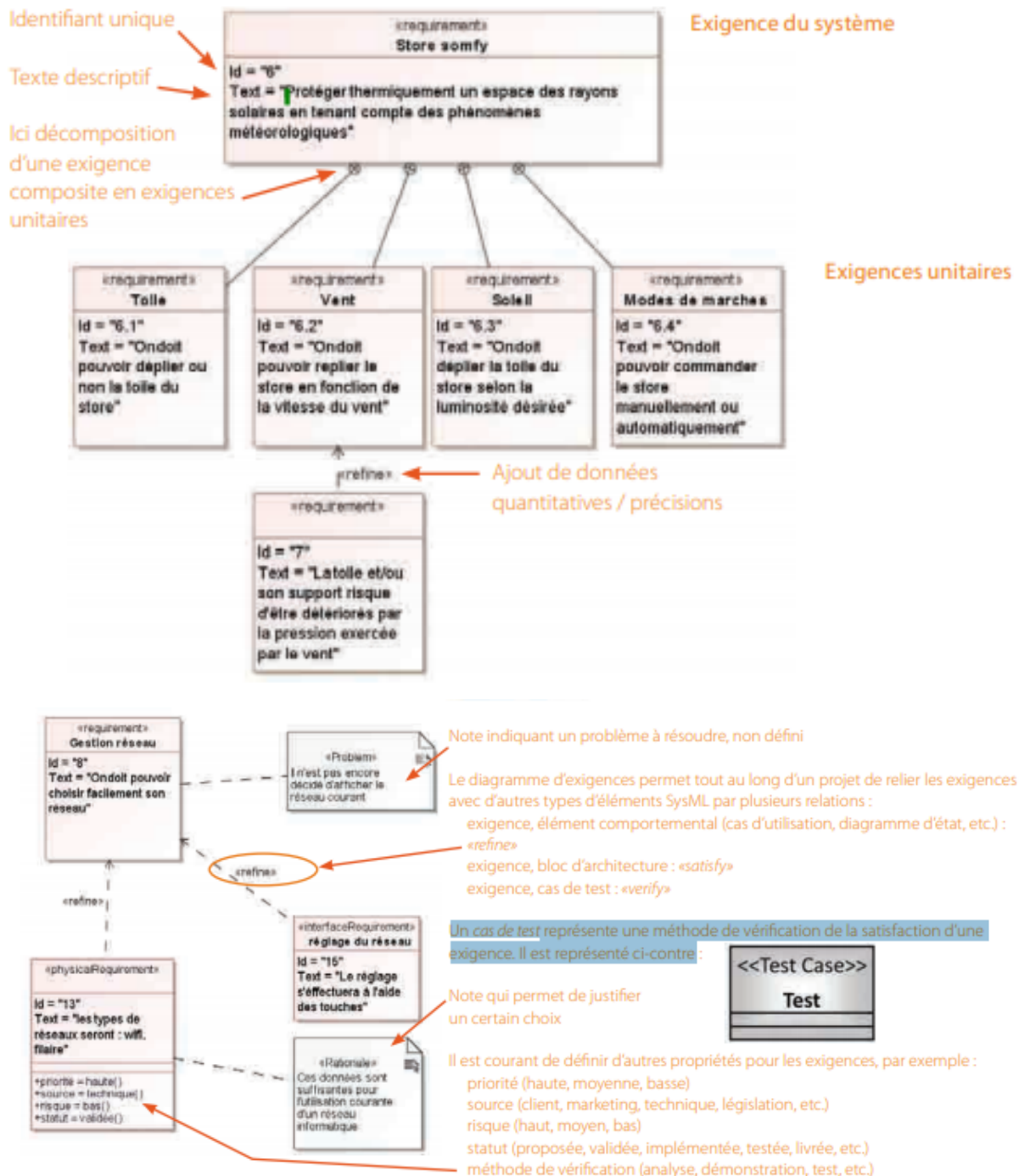


FIGURE 1.14 – Exemple de diagramme d'exigences [dia]

Bien qu'il soit possible d'importer ce diagramme dans des outils d'automatisation de test pour générer des scripts exécutables de test, il n'en reste pas moins que comme pour UML les spécificités du domaine et le langage utilisé n'est pas simple d'utilisation pour toutes les personnes impliquées dans le projet.

Critère	Répond au critère
La modélisation doit pouvoir exprimer l'exigence	OUI
La modélisation doit être lisible par n'importe quelle partie prenante	NON
La modélisation doit pouvoir être écrite par les personnes qui expriment l'exigence	NON
La modélisation doit permettre de générer des tests	OUI
Les tests générés doivent être des tests fonctionnels	OUI
N'importe quel domaine peut modéliser avec cet outil.	OUI
Le domaine est au coeur de la modélisations	NON

TABLE 1.8 – Évaluation du diagramme d'exigences selon les critères défini

Les méthodes formelles

Les Réseaux de Petri

«Les réseaux de Pétri sont un outil graphique et mathématique qui s'applique à un grand nombre de domaine où les notions d'évènements et d'évolutions simultanées sont importantes» [Valette 2000] Avec ce type de modélisation il est possible de générer des tests, notamment à partir des réseaux de pétri haut niveau comme les modèles de test [Xu 2011]. Bien que des tests puissent être générés ce type de modélisation est difficilement lisible et difficile à écrire pour les experts métier notamment.

La méthode B

La méthode B permet de modéliser les spécifications de manière abstraite. A l'instar des réseaux de Pétri, cette modélisation est beaucoup trop complexes pour les différentes parties prenantes.

Malgré la possibilité à partir des méthodes formelles de générer des cas de tests, il n'en reste pas moins que ces méthodes ne sont pas adapté à la compréhension de tous.

Les Domain Specific Languages

Au delà de la capacité pour un type de modélisation à générer des tests, il est important que toutes les exigences fonctionnelles puissent être correctement modélisées. Nous avons vu que les modélisations comme UML ou encore les diagrammes de flux permettent d'avoir une vision globale et/ou détaillée des exigences. Toutefois, le formalisme de ces diagrammes impose des notations, des abstractions et conception. Or, certains domaines nécessitent une flexibilité quant à l'expression des exigences fonctionnelles. De plus, ces modélisations sont souvent pas les meilleures pour les utilisateurs finaux désirant une notation plus familière [Barret R.Bryant 2010]. En effet, UML peut être difficile à appréhender pour les clients. Cette sous partie présente les DSL qui sont des langages de modélisations spécifiques à un domaine. Avec ces langages déclaratifs [Arie van Deursen 2000] il est possible d'une part de fournir des abstractions au niveau du domaine et d'autre part de permettre aux clients d'exprimer les exigences avec un langage dédié à leur domaine qui leur est compréhensible.

Un langage spécifique au domaine (DSL) est un langage de programmation ou un langage de spécification exécutable qui offre, par des notations et des abstractions appropriées, un pouvoir expressif axé sur, et généralement restreint à, un domaine particulier de problème [Arie van Deursen 2000].

Les DSL ont de nombreux avantages pour la modélisation des exigences. Tout d'abord ils permettent une facilité d'utilisation et une meilleure expressivité par rapport aux modélisations à usage général. Ils permettent aux experts domaine de comprendre, valider, modifier et même parfois programmer les programmes [Arie van Deursen 2000, Barret R.Bryant 2010]. Les DSL offrent également l'avantage d'écrire des exigences précises et indépendantes des plateformes [Barret R.Bryant 2010]. D'autre part, les DSL améliorent la testabilité en suivant des approches telles que citées dans [Emin Gün Sirer 1999, Arie van Deursen 2000]. Il existe des outils qui permettent à partir d'un DSL prédéfini de générer des tests d'acceptation. Ces outils et méthodes sont décrits dans la partie suivante.

1.2.3 De la modélisation vers les tests d'acceptation - Les framework BDD

Cucumber

Cucumber est un outil qui permet de lancer automatiquement des tests d'acceptation et qui est créé en behaviour-driven development. Ce dernier est une méthode qui encourage la collaboration entre les différentes parties prenantes. L'outil est basé sur Gherkin qui est le format des spécifications Cucumber. Il est lisible par le client. Il s'agit d'un DSL qui permet à tout le monde de comprendre facilement le comportement attendu du logiciel.

Pour définir une structure, Gherkin a des espaces et des indentations.

Quelques exemples de la syntaxe Gherkin : Feature, Background, Scenario, Given, When, Then, And, But, Scenario outline, Examples, Scenario Templates. Les fichiers rédigés pourront ensuite générer des signatures de méthodes. Les développeurs devront rédiger le corps de la méthode de test.

```
1: Feature: Some terse yet descriptive text of what is desired
2:   Textual description of the business value of this feature
3:   Business rules that govern the scope of the feature
4:   Any additional information that will make the feature easier to understand
5:
6:   Scenario: Some determinable business situation
7:     Given some precondition
8:       And some other precondition
9:     When some action by the actor
10:      And some other action
11:      And yet another action
12:     Then some testable outcome is achieved
13:      And something else we can check happens too
14:
15:   Scenario: A different situation
16:     ...
```

FIGURE 1.15 – Exemple de syntaxe Gherkin

Cucumber offre un réel avantage : celui de la génération de test à partir d'exigences rédigés dans un langage domaine. Toutefois, il n'existe pas de retour des tests vers les spécifications initiales.

EasyB

Cet outil est un framework basé sur Groovy qui utilise un langage DSL pour la plateforme Java. Groovy est utilisé pour exprimer les scénarios.

```
scénario "L'utilisateur entre des informations d'identification valides", {  
    étant donné que "le compte d'utilisateur existe déjà", {  
  
    }  
    lorsque "utilisateur se connecte", {  
    }  
    alors "le système doit renvoyer un compte valide", {  
  
    }  
}
```

FIGURE 1.16 – Exemple de scénario Groovy

Un rapport textuel de scénario est créé à la fin de chaque execution de tests.

scénario utilisateur entre des informations d' identification valides
données compte d'utilisateur existe déjà
lorsque les connexions des utilisateurs
, le système retourne un compte valide

scénario utilisateur entre des informations d' identification non valides
donné compte d'utilisateur existe déjà
lorsque les connexions utilisateur avec mot de passe incorrect
alors un compte null doit être retourné

scénario Invalid login avec un mot de passe nul
compte compte d'utilisateur déjà existant
lorsque l'utilisateur se connecte avec mot de passe nul
puis une exception devrait être levée

FIGURE 1.17 – Exemple de rapport EasyB

1.2.4 JDave

Jdave est un framework qui permet de spécifier le comportement des classes. Un comportement défini le comportement d'une classe selon un certain contexte. Cet outil nous paraît beaucoup trop technique et intervient sur le code plutôt que sur les abstraction domaine.

Concordion

Cet outil permet d'écrire des scripts d'automatisation des tests d'acceptation dans les projets basés sur JAVA. Les spécifications doivent être écrites en HTML. Ceci présente plusieurs inconvénients : l'expert métier doit savoir écrire du HTML et le langage HTML ne permet pas de décrire correctement les exigences.

Ces deux derniers outils sont beaucoup trop spécifiques au langage de programmation.

1.2.5 Conclusion

Pour conclure, nous avons vu que les outils passant des exigences rédigées en langage naturel aux tests ne proposent pas de manière d'automatiser les tests d'acceptation. Nous avons donc ensuite présenté différentes façons de modéliser les exigences pour pouvoir dans un second temps générer des tests à partir de ceux-ci. Bien que les notations UMLs/SysML soient riches en diagrammes et proposent de la génération de code, il est nécessaire d'avoir une certaine connaissance en modélisation et les notions spécifiques d'un domaine peuvent être mal (ou pas du tout) représentées. Les Domain Specific Language semblent être une bonne alternative : ils proposent de mettre le domaine au coeur de la modélisation et d'en générer les tests. Les outils comme Cucumber montrent qu'il est possible à partir DSL orienté langage naturel de générer des tests.

Chapitre 2

Des exigences vers les tests et vice versa

2.1 Un DSL pour modéliser le domaine

Nous avons vu dans le premier chapitre de ce mémoire que le logiciel Cucumber permettait de rédiger des scénarios de tests fonctionnels avec un DSL qui s'apparente à un langage naturelle. Ceci permet donc la compréhension de toutes les parties prenantes de ces scénarios tout en impliquant le domaine. D'autre part, l'outil permet à partir de ce DSL de scénario de générer les tests fonctionnels associé.

Nous souhaitons conserver cette base pour apporter une solution à notre problématique.

2.2 Générer les tests

2.2.1 Génération des méthodes

2.2.2 Rédaction des commentaires

2.3 Analyser les commentaires

2.4 Critique de la solution

Bibliographie

- [Arie van Deursen 2000] Paul Klint Arie van Deursen et Joost Visser. *Domain-Specific Languages : An Annotated Bibliography*. Newsletter ACM SIGPLAN Notices Homepage archive Volume 35 Issue 6, Pages 26 - 36, 2000.
- [Barret R.Bryant 2010] Jeff Gray Barret R.Bryant et Marjan Mernik. *Domain-specific software engineering*. Conference Paper, 2010.
- [Barret R.Bryant 2011] Jeff Gray Barret R.Bryant et Marjan Mernik. *A domain specific Requirements Model for Scientific Computing : NIER Track*. IEEE, 2011.
- [C. Coulin 2005] D. Zowghi C. Coulin et “A Lightweight Workshop A. E. K. Sahraoui. *Centric Situational Approach for the Early Stages of Requirements Elicitation in Software Systems Development*. Proceedings of the International Workshop on Situational Requirements Engineering Processes (SREP 2005), France, 2005.
- [dia] *Diagramme d'exigences*.
- [Elizabeth Hull 2010] Jeremy Dick Elizabeth Hull Ken Jackson. Requirements engineering. Springer, 2010.
- [Emin Gün Sirer 1999] Brian N. Bershad Emin Gün Sirer. *Using Production Grammars in Software Testing*. 1999.
- [Felderer 2010] Michael Felderer, Philipp Zech, Frank Fiedler et Ruth Breu. *A Tool-Based Methodology for System Testing of Service-Oriented Systems*. vol. 0, pages 108–113, 08 2010.
- [fit] *FitNesse*.
- [fix] *Test Fixture*.
- [Ges] *Cours gestion des exigences*.
- [Iris Reinhartz-Berger 2011] Yair Wand Iris Reinhartz-Berger Arnon Sturm. Domain engineering. Springer, 2011.
- [Kundu 2008] Debasish Kundu et Debasis Samanta. *A Novel Approach to Generate Test Cases from UML Activity Diagrams*. 2008.
- [M. Dhineshkumar 2014] Galeebathullah M. Dhineshkumar. *An Approach to Generate Test Cases from Sequence Diagram*. Intelligent Computing Applications (ICICA), 2014 International Conference on, 03 2014.

- [Maurer 2004] Grigori MelnikKris ReadFrank Maurer. *Suitability of FIT User Acceptance Tests for Specifying Functional Requirements : Developer Perspective*. Extreme Programming and Agile Methods - XP/Agile Universe 2004, 2004.
- [Stéphane Badreau 2014] Jean-Louis Boulanger Stéphane Badreau. *Ingénierie des exigences : Méthodes et bonnes pratiques pour construire et maintenir un référentiel*. DUNOD. 2014.
- [sys] *SysML*.
- [Troyer 2014] Olga De Troyer et Erik Janssens. *A feature modeling approach for domain-specific requirement elicitation*. Dept. Computer Science Vrije Universiteit Brussel, 2014.
- [Valette 2000] Robert Valette. *Les Réseaux de Pétri*. LAAS-CNRS, 2000.
- [Vikas Panthi 2012] Durga Prasad Mohapatra Vikas Panthi. *Automatic Test Case Generation using Sequence Diagram*. 2012.
- [Xu 2011] Dianxiang Xu. *A Tool for Automated Test Code Generation from High-Level Petri Nets*. National Center for the Protection of the Financial Infrastructure, Dakota State University, 2011.

Résumé — Lorem ipsum dolor sit amet, consectetur adipiscing elit. Sed non risus. Suspendisse lectus tortor, dignissim sit amet, adipiscing nec, ultricies sed, dolor. Cras elementum ultrices diam. Maecenas ligula massa, varius a, semper congue, euismod non, mi. Proin porttitor, orci nec nonummy molestie, enim est eleifend mi, non fermentum diam nisl sit amet erat. Duis semper. Duis arcu massa, scelerisque vitae, consequat in, pretium a, enim. Pellentesque congue. Ut in risus volutpat libero pharetra tempor. Cras vestibulum bibendum augue. Praesent egestas leo in pede. Praesent blandit odio eu enim. Pellentesque sed dui ut augue blandit sodales. Vestibulum ante ipsum primis in faucibus orci luctus et ultrices posuere cubilia Curae; Aliquam nibh. Mauris ac mauris sed pede pellentesque fermentum. Maecenas adipiscing ante non diam sodales hendrerit. Ut velit mauris, egestas sed, gravida nec, ornare ut, mi. Aenean ut orci vel massa suscipit pulvinar. Nulla sollicitudin. Fusce varius, ligula non tempus aliquam, nunc turpis ullamcorper nibh, in tempus sapien eros vitae ligula. Pellentesque rhoncus nunc et augue. Integer id felis.

Mots clés : Lorem ipsum dolor sit amet, consectetur adipiscing elit. Sed non risus. Suspendisse lectus tortor.