Année : 2017 **Thèse N° : 75/ST2I**

École Nationale Supérieure d'Informatique et d'Analyse des systèmes
Centre d'Études Doctorales en Sciences des Technologies de l'Information et de l'Ingénieur

THÈSE DE DOCTORAT

Ingénierie des Systèmes Orientés Services Adaptables : Approche à base de DSLs

Présentée par
Mohammed LETHRECH

Le 13 Mai 2017

Formation doctorale : Informatique

Structure de recherche : Laboratoire SIME, Equipe Ingénierie des Modèles et Systèmes (IMS)

JURY

Professeur Rachid OULAD HAJ THAMI PES, ENSIAS, Université Mohammed V de Rabat	**Président**
Professeur Samir MBARKI PES, Faculté des Sciences, Université Ibn Tofail, Kenitra	**Rapporteur**
Professeur Rahal ROMADI PH, ENSIAS, Université Mohammed V de Rabat	**Rapporteur**
Professeur Mahmoud NASSAR PES, ENSIAS, Université Mohammed V de Rabat	**Examinateur**
Professeur Abdelaziz KRIOUILE PES, ENSIAS, Université Mohammed V de Rabat	**Directeur de thèse**
Professeur Adil KENZI PH, ENSA, Université Sidi Mohamed Ben Abdellah, Fès	**Co-Encadrant**

A feue ma mère Rabha

A feu mon père Kouider

Remerciements

En concrétisant l'un de mes rêves d'enfance, je tiens tout d'abord à remercier dieu seigneur de l'univers pour la bénédiction de succès.

J'exprime ma profonde gratitude et mes remerciements à mon directeur de thèse M. Abdelaziz KRIOULE, professeur de l'enseignement supérieur à l'Ecole Nationale Supérieure d'Informatique et d'Analyse des Systèmes (ENSIAS). Je le remercie très sincèrement pour sa confiance, ses directives, ses conseils et ses encouragements. Plus qu'un directeur de thèse, je vous considère une source d'inspiration et d'énergie qui m'a fortement influencée sur tous les plans.

J'adresse mes remerciements les plus sincères à mon co-encadrant M. Adil KENZI Professeur Habilité à l'Ecole Nationale des Sciences Appliquées de Fès pour son suivi de mon travail de recherche, ses conseils, sa patience et sa disponibilité tout au long des années de ma thèse. Je voudrais également le remercier pour son effort qu'il a généreusement consacré à l'évaluation de ce manuscrit.

Je suis également très reconnaissant envers les professeurs Samir MBARKI, Ahmed El KHADIMI et Rahal ROMADI pour avoir accepté de rapporter cette thèse malgré leurs multiples préoccupations.

Je tiens aussi à remercier le professeur Rachid OULAD HAJ THAMI d'avoir accepté de présider le jury de ma thèse et le professeur Mahmoud NASSER d'avoir pris la peine d'examiner ce travail.

Je remercie vivement les membres de l'Equipe de recherche IMS pour leurs conseils et leurs remarques pertinentes.

J'exprime ma profonde reconnaissance à feue ma mère et feu mon père qui nous ont quittés au cours de la réalisation de cette thèse. Je n'oublierai jamais que c'est grâce à vous que je suis arrivé jusqu'ici. Que dieu les bénisse et les accueille dans son vaste paradis.

J'adresse aussi mes chaleureux remerciements et ma gratitude envers les membres de ma famille et spécialement mon épouse FatimaZahra pour ses sacrifices, son encouragement et son soutien dans les moments les plus difficiles. A mes chers enfants Zainab et Abderrahmane, je leur demande grand pardon de les avoir privé de ma présence pour de longues périodes.

J'exprime également mes remerciements à mes frères Nabil, Aida et Jalal, à ma grande famille et aussi à mes amis pour leurs prières et encouragements.

Enfin, merci à toute personne ayant contribué de près ou de loin à la réalisation de ce travail.

Mohammed LETHRECH, Salé le 18/11/2016

Résumé

Les systèmes d'information sont constamment influencés par les avancées implacables de l'ingénierie du logiciel. Actuellement, cette dernière est essentiellement marquée par l'émergence des paradigmes : SOC (Service Oriented Computing) et CAC (Context-Aware Computing). Le premier vise la réorganisation du système d'information sous forme d'un ensemble de services faiblement couplés, cohésifs et réutilisables. Le deuxième traite l'intégration de la capacité d'adaptation aux systèmes informatiques. L'utilisation simultanée de ces deux paradigmes soulève beaucoup de défis, spécialement celui de l'ingénierie des Systèmes dits « Orientés Services Adaptables » (SOSA).

La plupart des approches de modélisation des SOSA étudiées au fil de notre travail de recherche sont basées sur des profils UML. Néanmoins, ce mécanisme a reçu plusieurs critiques de la part de la communauté de recherche (manque de richesse sémantique, la connaissance de base d'UML est obligatoire, etc.). Une alternative intéressante des profils UML réside dans l'utilisation de l'approche DSM. C'est une approche générative, basée sur des langages spécifiques (DSL). L'adoption de l'approche DSM par le processus de développement des SOSA a fait naître de nouveaux challenges d'ingénierie de ce type de systèmes.

Le but de cette thèse est de proposer une approche DSM pour le développement des SOSA. Dans cette visée, nous avons proposé une approche, intitulée CADSSOMA (***C****ontext-****A****ware, **D**omain **S**pecific and **S**ervice **O**riented **M**odeling **A**pproch*). Elle définit principalement : (i) Cinq DSLs pour l'étape de modélisation, (ii) un processus de développement, et (iii) un outil support facilitant la modélisation et la génération automatique du code source.

Les cinq DSLs proposés sont utilisés dans la phase de modélisation pour représenter : (i) les services spécifiques au domaine, (ii) la variabilité des services, (iii) le contexte, (iv) les règles d'adaptation, et (v) les règles métier du domaine spécifique.

Nous avons aussi proposé un processus DSM pour le développement des SOSA associé à notre approche. Il définit les phases, les activités et les artefacts nécessaires pour la transformation du métier d'un domaine spécifique en des services flexibles et adaptables.

Finalement, nous avons validé notre approche en proposant un environnement de développement intégré (*CADSSO**TB, T**ool**B**ox*) qui facilite la mise en œuvre de l'approche CADSSOMA. Après la spécification graphique des cinq modèles de la phase de modélisation, la boite à outils s'occupe de la génération automatique de la solution finale en se basant sur une seule transformation Modèle-à-Texte. L'étude de cas utilisée correspond au domaine de calcul et restitution d'impôts.

Mots clés : Modélisation Spécifique au Domaine (DSM), Langage Spécifique au Domaine (DSL), CADSSOMA, Informatique Orientée Services (SOC), Architecture Orientée Services (SOA), Variabilité de service, Contexte, Informatique Sensible au Contexte (CAC), Adaptabilité.

Abstract

Information systems are constantly influenced by the relentless advances in software engineering. Currently, the latter is essentially marked by the emergence of two paradigms: SOC (Service Oriented Computing) and CAC (Context-Aware Computing). The first aims at the reorganization of the information system as a set of loose coupled, cohesive and reusable services. The second deals with the integration of adaptability to computer systems. The simultaneous use of these two paradigms raises many challenges, especially the Adaptable Service-Oriented Systems (ASOS) engineering.

Most of the ASOS modeling approaches studied during our research are based on UML profiles. However, this mechanism has received a lot of criticisms from the research community (lack of semantic richness, basic knowledge of UML is mandatory, etc.). An interesting alternative of the UML profiles is the use of the DSM approach. It is a generative approach, based on Domain Specific Languages (DSL). The adoption of the DSM approach by the ASOS development process has created new engineering difficulties of such systems.

The objective of this thesis is to propose a DSM approach for the development of ASOS. To this aim, we have proposed an approach, called CADSSOMA (***C**ontext-**A**ware, **D**omain **S**pecific and **S**ervice **O**riented **M**odeling **A**pproach*) which mainly defines: i) Five DSLs for the modeling step, ii) a development process, iii) a software tool for modeling and automatic generation of source code.

The five proposed DSLs are used in the modeling phase to model: i) domain specific services, ii) services variability, iii) context, iv) adaptation rules and v) domain specific business rules.

We have also proposed a DSM process for the development of ASOS associated with our approach. It defines the necessary phases, activities and artifacts allowing the transformation of the domain specific business rules and entities into flexible and adaptable services.

Finally, we have validated our approach by proposing an integrated development environment (*CADSSO**TB, T**ool**B**ox*) which facilitates the implementation of the CADSSOMA approach. After the modeling stage which consists on the graphical modeling of the five models, the toolbox takes care of the automatic generation of the final solution based on a single Model-to-Text transformation. The used case study corresponds to the field of tax calculation and restitution.

Keywords: Domain Specific Modeling (DSM), Domain Specific Language (DSL), CADSSOMA, Service Oriented Computing (SOC), Service Oriented Architecture (SOA), Service variability, Context, Context-Aware Computing (CAC), Adaptability.

Table des matières

Remerciement 5
Résumé 7
Abstract 9
Table des figures 15
Liste des tableaux 18
Liste des abréviations 19
Chapitre I Introduction générale 21
I.1. Contexte de la thèse 21
I.2. Contexte du travail 21
I.2.1. Introduction 21
I.2.2. L'Informatique Orientée Services (SOC) 22
I.2.3. L'Informatique sensible au contexte (CAC) 23
I.2.4. Modélisation spécifique au domaine (DSM) 24
I.3. Problématique 25
I.4. Contributions 27
I.4.1. Approche d'ingénierie des systèmes orientés services adaptables à base de DSLs 27
I.4.1.1. Phase de modélisation 27
I.4.1.2. Phase de génération de code source 28
I.4.2. Formalisme du processus de développement des systèmes spécifiques au domaine 29
I.4.3. Processus de développement des SOSAs à base de DSLs 29
I.4.4. Outil support CADSSOTB 29
I.5. Organisation du mémoire 30
Chapitre II Etat de l'Art 32
II.1. Introduction 32
II.2. Notions et concepts de base 33
II.2.1. Le paradigme SOC 33
II.2.1.1. Définition du concept « Service » 34
II.2.1.2. Vue architecture 35
II.2.1.3. Vue technologique 36
II.2.1.4. Vue méthode de mise en place d'un SOS 36
II.2.1.5. Vue de composition de services 38
II.2.1.6. Synthèse 39
II.2.2. Modélisation Spécifique au Domaine (DSM) 39
II.2.2.1. Définitions 40
II.2.2.2. Principe de base 41
II.2.2.3. Architecture d'une solution spécifique au domaine 42
II.2.2.4. Apport de l'approche DSM 43
II.2.2.4.1. Productivité 43
II.2.2.4.2. Qualité 43
II.2.2.4.3. Valorisation de l'expertise métier 44
II.2.2.5. Quand utiliser l'approche DSM ? 44
II.2.2.6. Synthèse 44

II.2.3. SOS : Concepts d'adaptation et de contexte 45
II.2.3.1. Définitions 45
II.2.3.1.1. Définition du concept d'adaptation 45
II.2.3.1.2. Définition du contexte 45
II.2.3.1.3. Définition du système adaptable 46
II.2.3.2. Ingrédients de l'adaptation 46
II.2.3.2.1. La dimension « Why » ? 47
II.2.3.2.2. La dimension « What ? » 48
II.2.3.2.3. La dimension « How ? » 48
II.2.3.3. Catégorisation de l'adaptation 49
II.2.3.4. Mécanismes d'adaptation de service 50
II.2.3.4.1. Modélisation explicite de la variabilité 50
II.2.3.4.2. Le service multi-vue 53
II.2.3.4.3. Sélection de composition 53
II.2.3.4.4. Sélection de service 53
II.2.3.4.5. Différenciation de service 53
II.2.3.5. Synthèse 54
II.2.4. Conclusion 55
II.3. Approches de modélisation des SOSs adaptables 55
II.3.1. Approches à base de profils UML 56
II.3.1.1. Approches basées sur la modélisation du contexte 56
II.3.1.1.1. Descriptions 56
II.3.1.1.2. Discussion 61
II.3.1.2. Approches basées sur le profil SoaML 62
II.3.1.2.1. Descriptions 62
II.3.1.2.2. Discussion 66
II.3.1.3. Approches basées sur la modélisation de la variabilité de service 67
II.3.1.3.1. Descriptions 67
II.3.1.3.2. Discussion 73
II.3.1.4. Approches à base de vues 74
II.3.1.4.1. Descriptions 74
II.3.1.4.2. Discussion 76
II.3.2. Approches à base de DSLs 76
II.3.2.1. Domaine : adaptation et modélisation de services 76
II.3.2.2. Domaine : Sécurité 82
II.3.2.3. Domaine : Qualité de service 84
II.3.2.4. Domaine : orchestration de services 86
II.3.2.5. Discussion 89
II.3.3. Synthèse 91
II.4. Technologies d'implémentation 95
II.4.1. Implémentation des services 95
II.4.1.1. Pile des standards W3C / OASIS 95
II.4.1.2. Technologie REST 96
II.4.2. Implémentation de l'adaptation 97
II.4.2.1. Moteur de règles 97
II.4.2.2. Programmation Orientée Context (COP) 97

II.4.2.3. Programmation Orientée Aspect (AOP) 98
II.4.2.4. Programmation Orientée Caractéristique (FOP) 98
II.4.2.5. JCAF (Java Context Awareness Framework) 99
II.4.2.6. AWSDL et CWSDL 99
II.4.3. Synthèse 99
II.5. Conclusion générale 100
Chapitre III CADSSOMA : Une approche de modélisation des SOSA basée sur l'ingénierie des langages spécifiques au domaine 103
III.1. Introduction 103
III.2. Syntaxes abstraites des DSLs 104
III.2.1. Modélisation des services : méta-modèle générique 105
III.2.2. Modélisation de la variabilité de service 108
III.2.3. Modélisation du contexte 110
III.2.4. Modélisation des règles d'adaptation 113
III.2.5. Modélisation du métier du domaine spécifique 116
III.3. Etude de cas 117
III.3.1. Description de l'étude de cas : calcul et restitution d'impôts 117
III.3.2. Définition des méta-modèles 120
III.3.2.1. Méta-modèle des services spécifiques au domaine : services de calcul et de restitution d'impôts 120
III.3.2.2. Méta-modèle du métier du domaine spécifique : domaine de calcul et restitution d'impôts 122
III.3.3. Spécification des modèles 123
III.3.3.1. Modèle des services de calcul et de restitution de l'impôt sur les sociétés 123
III.3.3.2. Modèle de la variabilité de services 124
III.3.3.3. Modèle de contexte de l'impôt sur les sociétés 126
III.3.3.4. Modèle des règles d'adaptation 127
III.3.3.5. Modèle des règles de calcul et de restitution de l'impôt sur les sociétés 129
III.4. Conclusion 130
Chapitre IV Processus DSM pour le Développement des SOSAs 134
IV.1. Introduction 134
IV.2. Processus de développement des systèmes orientés services : phases, activités et artéfacts 135
IV.3. Développement des systèmes spécifiques au domaine : Equipe, Processus et cycle de vie 138
IV.3.1. L'Equipe DSM 139
IV.3.2. Processus de développement des systèmes spécifiques au domaine 139
IV.3.3. Cycle de vie d'une solution spécifique au domaine 145
IV.4. Processus DSM pour le développement des SOSAs 146
IV.5. Conclusion 152
Chapitre V CADSSOTB : Un Environnement de Développement des SOSAs Spécifiques au Domaine 155
V.1. Introduction 155
V.2. Environnements de méta-modélisation : Etude comparative 156
V.2.1. Pile de méta-modélisation de l'approche DSM 156
V.2.2. Critères de l'étude comparative des EMMs 158

V.2.3. Résultat de l'étude comparative des EMMs 159
V.2.4. Plateforme Eclipse : Un environnement de développement extensible 161
V.3. Mise en place de l'outil support CADSSOTB 163
V.3.1. Syntaxe abstraite 164
V.3.1.1. Syntaxe abstraite du langage de modélisation des services de calcul d'impôts .. 167
V.3.1.2. Syntaxe abstraite du langage de modélisation de la variabilité de service 168
V.3.1.3. Syntaxe abstraite du langage de modélisation du contexte 168
V.3.1.4. Syntaxe abstraite du langage de modélisation des règles d'adaptation (tables de décision) 169
V.3.1.5. Syntaxe abstraite du langage de modélisation des règles de calcul d'impôts 169
V.3.2. Syntaxe concrète 170
V.3.2.1. Choix de la notation des DSLs utilisés : Syntaxe concrète 170
V.3.2.2. GMF : Un outil de transformation d'une syntaxe abstraite à une syntaxe concrète graphique 172
V.3.2.3. Syntaxe concrète des langages de modélisation des services et de la variabilité de service de calcul d'impôts 174
V.3.2.4. Syntaxe concrète du langage de modélisation du contexte 177
V.3.2.5. Syntaxe concrète du langage de modélisation des règles d'adaptation 179
V.3.3. Environnement de développement CADSSOTB 181
V.3.4. Spécification et développement des règles de conception 182
V.3.5. Transformation des modèles au code source 185
V.4. Conclusion 188
Chapitre VI Conclusion générale 190
VI.1. Résumé de la contribution de la thèse 190
VI.2. Travaux en cours et perspectives 191
VI.2.1. Optimisation des règles d'adaptation 191
VI.2.2. Utilisation de l'adaptation dynamique 191
VI.2.3. Mesure de l'apport de l'approche DSM 192
VI.2.4. Amélioration de l'outil support CADSSOTB 192
VI.3. Liste des publications 193
VI.3.1. Revues internationales 193
VI.3.2. Conférences internationales 193
VI.3.3. Workshops 194
Références bibliographiques 195
Annexe A 206
Annexe B 209
Annexe C 212
Annexe D 214

Table des figures

Figure 1 : Vue d'ensemble de l'approche CADSSOMA 29
Figure 2 : Vision multi-vues du paradigme service (Boukadi, 2009) 34
Figure 3 : Méta-modèle du style d'architecture SOA (Arsanjani, 2004) 36
Figure 4 : Approche DSM : Projection entre l'espace problème et l'espace solution (Bonnet, 2005)..... 42
Figure 5 : Architectures d'une solution DSM (Kelly, et al., 2008a) 43
Figure 6 : Taxonomie de l'adaptation (Kazhamiakin, et al., 2010)..... 48
Figure 7 : Méta-modèle de CSOMA (Boukadi, 2009)..... 57
Figure 8 : Méta-modèle du contexte (Monfort, et al., 2009)..... 59
Figure 9 : Méta-modèle ContextUML (Sheng, et al., 2005) 60
Figure 10 : Diagrammes de contexte du scénario « logistique de voiture » (Raik, et al., 2012).... 61
Figure 11 : Partie « Services » du profil SoaML 1.0.1 (OMG, SoaML, 2012)..... 62
Figure 12 : Fragment du méta-modèle CASosML (Hafiddi, 2012) 64
Figure 13 : Méta-modèle de base des services sensibles au contexte (Hafiddi, 2012)..... 64
Figure 14 : Méta-modèle du contexte (Hafiddi, 2012)..... 65
Figure 15 : Méta modèle de la variabilité de service (Abu-Matar, et al., 2011b) 66
Figure 16 : Méta-modèle de service (Chang, et al., 2007b) 68
Figure 17 : Méta-modèle de variabilité (Kim, et al., 2005)..... 69
Figure 18 : Méta-modèle de variation (Narendra, et al., 2010)..... 70
Figure 19 : Méta-modèle de gestion de la variabilité (Clotet Martínez, et al., 2008) 70
Figure 20 : Méta-modèle d'une application SCA (Parra, et al., 2009)..... 72
Figure 21 : Méta-modèle des Assets sensibles au contexte (Parra, et al., 2009)..... 72
Figure 22 : Méta-modèle VxUML (Extension du diagramme d'activités UML) (He, et al., 2015) 73
Figure 23 : Méta-modèle VSoaML (Kenzi, 2010)..... 75
Figure 24 : Méta-modèle du langage de modélisation du contexte (Achilleos, 2009)..... 78
Figure 25 : Intégration des modèles d'un service sensible au contexte (Achilleos, 2009)..... 79
Figure 26 : Méta-modèle du DSL associé au PIM (López-Sanz, et al., 2012)..... 79
Figure 27 : Méta-modèle SOA DSL (Wada, et al., 2005c)..... 82
Figure 28 : Méta-modèle BPMN enrichi par des éléments de sécurité (Saleem, et al., 2012)..... 83
Figure 29 : Modèle de haut niveau du langage QUALA (Oberortner, 2011) 85
Figure 30 : Syntaxe concrète du langage de haut niveau QUALA (Oberortner, 2011) 85
Figure 31 : Modèle de bas niveau du langage QUALA (Oberortner, 2011)..... 86
Figure 32 : Architecture de la plateforme Swashup (Maximilien, et al., 2007) 87
Figure 33 : Structure globale d'un programme VCL (Rosenberg, et al., 2009b)..... 88
Figure 34 : Types de correspondance entre un problème d'un domaine spécifique et ses solutions (MetaCase, 2012) 92
Figure 35 : Pile de standards et langages des services Web (Boukadi, 2009)..... 95
Figure 36 : Le méta-modèle générique de services..... 107
Figure 37 : Méta-modèle de la variabilité de service 110
Figure 38 : Méta-modèle du contexte..... 113
Figure 39 : Eléments de base d'une table de décision (Pooch, 1974)..... 114
Figure 40 : Méta-modèle des règles d'adaptation (tables de décision) 116

Figure 41 : Phase de modélisation de l'approche CADSSOMA 120
Figure 42 : Méta-modèle des services de calcul et de restitution d'impôts 122
Figure 43 : Méta-modèle des formules de calcul d'impôts 123
Figure 44 : Modèle des services de calcul et de restitution de l'impôt sur les sociétés 124
Figure 45 : Modèle de variabilité de service de l'impôt sur les sociétés 125
Figure 46 : Modèle du contexte de l'impôt sur les sociétés 126
Figure 47 : Les *ServiceContextElements* des services de calcul et de restitution de l'impôt sur les sociétés 127
Figure 48 : Ensemble des méta-modèles de l'approche CADSSOMA 132
Figure 49 : Phase du processus de développement des SOS (Papazoglou, et al., 2006) 136
Figure 50 : Processus de développement des applications spécifiques au domaine 140
Figure 51 : Cycle de vie d'une solution spécifique au domaine 145
Figure 52 : Processus DSM pour le développement des SOSAs 147
Figure 53 : Cas d'utilisation du domaine spécifique : calcul et restitution d'impôts 148
Figure 54 : Services candidats du domaine de calcul et restitution d'impôts 150
Figure 55 : Relations entre les modèles de l'approche CADSSOMA 151
Figure 56 : Pile de modélisation de l'OMG (image adaptée de (Bonnet, 2005)) 157
Figure 57 : Architecture d'Eclipse (Eclipse, help, 2016) 162
Figure 58 : Caractère extensible de l'environnement de développement Eclipse 162
Figure 59 : Outils utilisés du projet de modélisation Eclipse 163
Figure 60 : Processus de mise en place d'une boite à outils d'un DSL 164
Figure 61 : Architecture d'EMF (Steinberg, et al., 2008b) 165
Figure 62 : Extrait du méta-méta-modèle Ecore (Steinberg, et al., 2008a) 166
Figure 63 : Modèle ECore du langage de modélisation des services de calcul d'impôts 167
Figure 64 : Modèle ECore du langage de modélisation de la variabilité de service 168
Figure 65 : Modèle ECore du langage de modélisation du contexte 168
Figure 66 : Modèle ECore du langage de modélisation des règles d'adaptation 169
Figure 67 : Modèle ECore du langage de calcul d'impôts 170
Figure 68 : Correspondance entre les concepts d'un domaine, la syntaxe abstraite et la syntaxe concrète (MataCase; Kelly, S., 2012) 172
Figure 69 : Flux de travail du composant Tooling Framework de GMF (Gronback, 2009) 173
Figure 70 : Exemple d'utilisation du composant Tooling Framework du GMF (modèle du contexte) 174
Figure 71 : Assistant GMF de transformation de la syntaxe abstraite (*.ecore*) à une syntaxe concrète (*.gmfgraph*) 175
Figure 72 : Modèle des services de calcul de l'impôt sur les sociétés 175
Figure 73 : Modèle de la variabilité de service du service *CTCalculationService* 176
Figure 74 : Spécification des points de variation du service *CTCalculationService* 176
Figure 75 : Modèle du contexte de l'impôt sur les sociétés 178
Figure 76 : Modèle des règles d'adaptation du service de calcul de l'impôt sur les sociétés *CTCalculationService* 179
Figure 77 : Liaison d'une *ConditionCell* à un *ContextParameter* 180
Figure 78 : Architecture de l'outil support CADSSOTB 181
Figure 79 : Plug-ins de l'outil support CADSSOTB 182
Figure 80 : Assistant de création des modèles de l'approche CADSSOMA 182
Figure 81 : Ajout de la méthode *validate* à la méta-classe *Service* 184

Figure 82 : Implémentation de la méthode *validate* de la classe *ServiceImpl.java* 184
Figure 83 : Exemple du résultat de validation du modèle des services de calcul d'impôts 185
Figure 84 : Génération de la solution finale : Service adaptable de calcul d'impôts 185
Figure 85 : Extrait du template Acceleo "*generateImplementations*" 187
Figure 86 : Extrait du résultat de la génération de code source du template Acceleo *« generateImplementations »* 187
Figure 87 : Modèle Ecore de base (Gronback, 2009) 206
Figure 88 : Les attributs Ecore (Steinberg, et al., 2008a) 207
Figure 89 : Les opérations et les paramètres Ecore (Steinberg, et al., 2008a) 207
Figure 90 : Génération d'un modèle Ecore : Modèle java de base 209
Figure 91 : Génération d'un modèle Ecore : Choix du générateur de modèles EMF 210
Figure 92 : Génération d'un modèle Ecore : Choix de l'utilisation d'interfaces Java annotées .. 210
Figure 93 : Génération d'un modèle Ecore : Choix du package 211
Figure 94 : Génération d'un modèle Ecore : Génération des fichiers *.ecore* et *.genmodel* 211
Figure 95 : Tableau de bord GMF des étapes de production de l'outil de modélisation du contexte 212
Figure 96 : Extrait du template Acceleo "generateInterfaces" 212
Figure 97 : Extrait du résultat de la génération de code source du template Acceleo « generateInterfaces » 213
Figure 98 : Template d'un atelier DSM (Kelly, et al., 2008a) 217

Liste des tableaux

Tableau 1 : Stéréotypes du profil SoaML 1.0.1 63
Tableau 2 : Stéréotypes du profil VSoaML 75
Tableau 3 : Principaux éléments de la syntaxe concrète du DSL du framework ArchiMeDeS (López-Sanz, et al., 2012) 80
Tableau 4 : Résultat de l'étude comparative des DSLs étudiés 89
Tableau 5 : DSL vs Profil UML 94
Tableau 6 : Règles d'adaptation du service "CTCalculationService" 127
Tableau 7 : Règles d'adaptation du service "GetCTRate" 128
Tableau 8 : Règles d'adaptation du service "CTAdvanceRestitution" 128
Tableau 9 : Règles d'adaptation du service "CTRestitutionProcess" 128
Tableau 10 : Description des opérandes des formules de calcul de l'impôt sur les sociétés 129
Tableau 11 : Résultat de l'étude comparative des environnements de méta-modélisation 159
Tableau 12 : Correspondance entre la syntaxe abstraite et la syntaxe concrète du langage de modélisation des services de calcul d'impôts 177
Tableau 13 : Correspondance entre la syntaxe abstraite et la syntaxe concrète du langage de modélisation du contexte 179
Tableau 14 : Correspondance entre la syntaxe abstraite et la syntaxe concrète du langage de modélisation des règles d'adaptation 180
Tableau 15 : Correspondance entre les éléments des modèles du domaine spécifique calcul et restitution d'impôt et JAX-WS 186
Tableau 16 : Correspondance entre MOF et EMF (Ecore) (Maddeh, et al., 2007) 207
Tableau 17 : Questionnaire facilitant la prise de décision concernant l'adoption de l'approche DSM pour un domaine donné (Kelly, et al., 2008a) 214

Liste des abréviations

A	
AOP	Aspect Oriented Programming
ATL	Atlas Transformation Language
B	
BPEL	Business Process Execution Language
C	
CAC	Context-Aware Computing
CADSSOMA	Context-Aware, Domain Specific and Service Oriented Modeling Approach
CADSSOTB	Context-Aware, Domain Specific and Service Oriented Toolbox
CBD	Component-Based Development
CC/PP	Composite Capability/Preference Profile
COM	Component Object Model
COP	Context Oriented Programming
CORBA	Common Object Request Broker Architecture
D	
DCOM	Distributed Component Object Model
DSL	Domain Specific Language
DSM	Domain Specific Modeling
DSML	Domain Specific Modeling Language
DSSD	Domain Specific Software Development
E	
EDI	Environnement de Développement Intégré
EDSD	Environnements de Développement Spécifiques au Domaine
EJB	Entreprise Java Bean
EMM	Environnement de Méta-Modélisation
EMPF	Eclipse Modeling Project Framework
F	
FOP	Feature-Oriented Programming
G	
GP	Generative Programming
H	
HTTP	HyperText Transfer Protocol
I	
IDM	Ingénierie Dirigée par les Modèles
J	
JAX-WS	Java API for XML Web Services
JEE	Java Enterprise Edition
M	
MDA	Model Driven Architecture
MOF	Meta Object Facility
M2M	Transformation Modèle-à-Modèle
M2T	Transformation Modèle-à-Texte
O	
OASIS	Organization for the Advancement of Structured Information Standards

OCL	Object Constraint Language
OMG	Object Management Group
OWL-S	Semantic Markup for Web Services
P	
PIM	Platform Independent Model
POO	Programmation Orientée Objet
PSM	Platform Specific Model
Q	
QoS	Quality of Service
R	
REST	REpresentational State Transfer
RMI	Remote Method Invocation
S	
SCA	Service Component Architecture
SI	Système d'Information
SOA	Service Oriented Architecture
SoaML	Service oriented architecture Modeling Language
SOC	Service Oriented Computing
SOSA	Système Orienté Services Adaptable
SOS	Système Orienté Services
SPLE	Software Product Line Engineering
U	
UDDI	Universal Description Discovery and Integration
UML	Unified Modeling Language
URI	Uniform Resource Identifier
V	
VUML	View Based Unified Modeling Language
W	
W3C	World Wide Web Consortium
WSDL	Web Services Description Language
X	
XMI	XML Metadata Interchange
XML	Extensible Markup Language

Chapitre I

Introduction générale

I.1. Contexte de la thèse

Ce document présente les travaux de thèse effectués au sein de l'équipe de recherche Ingénierie des Modèles et Systèmes (IMS) du laboratoire SIME/ENSIAS. Notre travail s'inscrit dans la continuité des travaux de l'équipe IMS. En effet, depuis les années 90 l'équipe IMS s'est intéressée au problème d'adaptation des systèmes en proposant des concepts, processus et outils facilitant leur mise en place. Ci-dessous, nous en citons les principales contributions :

- Proposition d'une méthode d'analyse et de conception des applications orientées objet basée sur le concept de vue : VBOOM (View Based Oriented Object Method) (Kriouile, 1995) ;
- Création d'un outil support de la méthode VBOOM appelé VBTOOL (Hair, et al., 1998) ;
- Définition du profil VUML (View Based Unified Modeling Language) (Nassar, et al., 2003). Ce profil a permis de situer la méthode VBOOM dans le cadre du standard UML ;
- Introduction du concept « composant multi-vues » au profil VUML (ElAsri, 2005) ;
- Définition du concept « service multi-vues » et proposition du profil UML VSoaML (Kenzi, 2010) ;
- Proposition d'une approche d'ingénierie des systèmes orientés services sensibles au contexte basée sur une extension du profil UML SoaML (Hafiddi, 2012).

I.2. Contexte du travail

I.2.1. Introduction

L'information, est une ressource stratégique qui devient progressivement un instrument de compétitivité pour l'entreprise. En fait, si l'on considère que l'entreprise est un corps social vivant, l'information en sera sans aucun doute un élément vital. Elle a un rôle essentiel dans le processus de prise de décision tant au niveau des décisions opérationnelles quotidiennes qu'au niveau des grandes décisions stratégiques. Le passage de l'information à l'intelligence

économique passe inévitablement par la mise en place d'un système d'information composé de plusieurs systèmes informatiques.

De nos jours, les systèmes d'information sont extrêmement influencés par les progrès incessants de l'ingénierie du logiciel. L'objectif principal de cette section est de poser un regard objectif sur l'histoire de l'évolution du génie logiciel, de se repérer par rapport aux travaux antérieurs, et d'appréhender le présent tout en développant un regard critique vis-à-vis des travaux de recherche contemporains. Comme dit l'adage : « *Ceux qui ne peuvent se souvenir du passé, sont condamnés à le répéter* » George Santayana, The Life of Reason (1905).

Actuellement l'ingénierie du logiciel est marquée spécialement par l'émergence des trois paradigmes suivants : l'informatique orientée services, l'informatique sensible au contexte et la modélisation spécifique au domaine.

I.2.2. L'Informatique Orientée Services (SOC[1])

Depuis la reconnaissance du développement logiciel en tant que discipline d'ingénierie lors de la conférence de l'OTAN (NATO, 1968), la communauté de recherche en génie logiciel n'a cessé de fournir de nouveaux paradigmes et approches de construction des logiciels en essayant de répondre aux défis de qualité, productivité, séparation des préoccupations, réutilisation, réduction des coûts, etc. L'histoire a commencé par la première transition de l'approche impérative vers la programmation structurée (ou procédurale) entamée par Dijkstra à travers son article "GO TO statement considered harmful" (Dijkstra, 1968). Ensuite et vers la fin des années soixante, le premier exemple de la Programmation Orientée Objet (POO) a vu le jour (Black, 2013). La POO consiste en la définition et l'assemblage de briques logicielles appelées *objets*. Le niveau très fin de granularité des objets ainsi que leur couplage fort (Bonnet, 2005) a accéléré l'apparition du développement à base de composants (CBD[2]). CBD favorise la réutilisation, en effet, les systèmes logiciels peuvent être développés en assemblant les composants appropriés, immédiatement disponibles (Cai, et al., 2000). Au début des années 2000, et avec l'accélération de la demande d'intégration des SIs et la promotion de l'internet et des technologies web (Kumar, et al., 2009), nous avons assisté à l'évolution de la pensée composant sous forme de service (Stojanovic, 2005). L'informatique orientée services (SOC) vise principalement la réorganisation des SIs sous

[1] *Service-Oriented Computing* dans le vocabulaire anglo-saxon.
[2] *Component-Based Development* dans le vocabulaire anglo-saxon. Ce concept est adopté par d'autres disciplines d'ingénierie plus anciennes comme le Génie civil ou le Génie mécanique.

forme de services cohésifs, réutilisables et faiblement couplés (Davis, 2009) (Kumar, et al., 2009) (Josuttis, 2007) (Caseau, 2011). Leur orchestration permet de produire des processus à l'image des procédures métiers de l'entreprise ce qui améliore l'alignement entre la logique métier et la logique applicative (Josuttis, 2007).

I.2.3. L'Informatique sensible au contexte (CAC[1])

Ces dernières années, le concept *contexte* attire de plus en plus l'attention de la communauté de recherche spécialisée dans le domaine de l'interaction homme-machine (Schmidt, et al., 1999). Cela est dû essentiellement aux avancées dans le domaine de l'électronique et des télécommunications, à la généralisation de l'utilisation des appareils mobiles et aussi au besoin croissant de l'informatique ubiquitaire[2] (Madkour, et al., 2013). Plusieurs définitions du concept *contexte* existent dans la littérature scientifique, l'une des plus référencées est celle de Dey (Dey, 2001). Elle considère qu'un contexte est « toute information qui peut être utilisée pour caractériser la situation d'une entité. Une entité est une personne, un lieu ou un objet qui est considéré comme pertinent pour l'interaction entre un utilisateur et une application, y compris l'utilisateur et l'application ».

Retenons que la première vague de l'informatique mobile, basée sur des ordinateurs portables à usage général, est principalement axée sur une sensibilité de la position. Une deuxième vague est maintenant basée sur des appareils ultra-mobiles (téléphone portable, tablette, capteur, etc.) avec des capacités de calcul et de stockage très importantes. Ces dispositifs mobiles suscitent un intérêt grandissant à les rattacher aux différents paramètres de leurs situations environnantes d'utilisation (Schmidt, et al., 1999).

L'informatique Sensible au Contexte (CAC) a récemment émergé comme un nouveau paradigme prometteur pour le développement des systèmes adaptables, c'est-à-dire, qui peuvent découvrir et profiter de l'information contextuelle (Chen, et al., 2000). En conséquence, une grande partie des paramètres d'entrée fournis par les utilisateurs seront remplacés par les informations du contexte.

L'autre face de la médaille de la sensibilité au contexte est le concept d'adaptation. Plusieurs catégories de ce concept existent dans la littérature scientifique, Bucchiarone et al. (Bucchiarone, et al., 2013) distinguent entre « *at run-time adaptation* » et « *designed adaptation* ». La première pour une adaptation à la volée, la deuxième nécessite l'analyse de toutes les alternatives d'adaptation lors de la phase de conception du système. En outre,

[1] *Context-Aware Computing* dans le vocabulaire anglo-saxon.

[2] Elle vise la suppression des limites traditionnelles concernant comment, quand et où une interaction homme machine peut avoir lieu (Kazhamiakin, et al., 2010).

Kazhamiakin et al. (Kazhamiakin, et al., 2010) classent l'adaptation selon l'objectif cible : i) adaptation perfective : vise à améliorer l'application ; ii) adaptation corrective : vise à supprimer un mauvais comportement ; iii) adaptation adaptative : adapte le comportement de l'application en réagissant aux changements affectant son environnement ; iv) adaptation préventive : vise à prévenir les défauts futurs avant qu'ils ne surviennent et v) adaptation d'extension : étend l'application en ajoutant de nouvelles fonctionnalités. Il y a lieu de signaler que nous nous intéressons dans notre travail de recherche aux types d'adaptation *designed* et *adaptative* (précisément sensible au contexte).

I.2.4. Modélisation spécifique au domaine (DSM[1])

Tout au long de l'histoire du développement logiciel, les chercheurs ainsi que les développeurs ont toujours essayé d'améliorer la productivité en augmentant le niveau d'abstraction des langages de développement (Kelly, et al., 2008a). Booch, l'un des fondateurs du langage de modélisation UML, considère que l'histoire entière du génie logiciel est résumée dans l'élévation des niveaux d'abstraction[2]. Le nouveau niveau d'abstraction est automatiquement transformé à l'ancien niveau. En d'autres termes, si le code source représente l'implémentation pour un développeur, il est considéré comme étant une spécification pour un compilateur (Bonnet, 2005).

Force est de constater le rôle important joué par les langages de programmation traditionnels, communément catégorisés en troisième génération (3GLs), dans l'amélioration de la productivité (Luoma, et al., 2004). Cela sera plus visible si on compare le gain en productivité du passage de l'assembleur aux langages 3GLs[3]. En revanche, l'amélioration du niveau d'abstraction des langages de développement à usage général a atteint ses limites en termes d'amélioration de la productivité (Kelly, et al., 2008a).

L'approche modélisation spécifique au domaine (Domain Specific Modeling, DSM) est l'acteur principal d'une nouvelle révolution dans l'élévation du niveau d'abstraction (Kelly, et al., 2008a) (Kelly, S.; MetaCase, 2009a). Un langage spécifique au domaine (DSL[4]) est plus proche du langage naturel que les 3GLs. Cela est dû au fait qu'un DSL est par défaut sémantiquement riche, puisqu'il utilise les concepts et les règles métier d'un domaine spécifique. En plus de l'amélioration de la productivité[5], l'adoption de l'approche DSM dans

[1] *Domain Specific Modeling* dans le vocabulaire anglo-saxon.
[2] Présentation en 2002 intitulée "The Limits of Software".
[3] Estimée à environ 450% selon Jones (Jones, 1996).
[4] *Domain Specific Language* dans le vocabulaire anglo-saxon.
[5] Estimée entre 300% et 1000% (MetaCase, 2012).

l'ingénierie du logiciel fournit plusieurs autres avantages, essentiellement un code source sans bugs et une meilleure réactivité aux changements des règles métier et des technologies. En sus, l'approche DSM est une approche générative (Bonnet, 2005), les modèles conçus en utilisant la syntaxe concrète d'un DSL seront directement transformés au code source final. L'ensemble des modèles d'un domaine spécifique constitue un référentiel métier de l'entreprise qui est protégé contre les permanentes évolutions des technologies d'implémentation.

Il convient de signaler que l'OMG propose deux mécanismes pour produire un DSL (OMG, MOF, 2016) : i) le premier consiste en l'extension d'UML (profil UML) ; ii) le deuxième se base sur la définition d'un langage spécifique conforme au MOF. Dans le reste de ce manuscrit, nous considérons qu'un DSL est équivalent à un langage spécifique indépendant d'UML (Figure 56).

I.3. Problématique

L'utilisation simultanée des deux paradigmes SOC et CAC a généré plusieurs défis d'ingénierie du logiciel, essentiellement la définition des approches de modélisation, processus, outils et techniques pour faciliter la mise en œuvre de ce type de systèmes dits Orientés Services Adaptables (SOSA) (Papazoglou, et al., 2008). La plupart des travaux étudiés au fil de notre travail de recherche :

- N'intègrent pas l'ensemble des composants d'un système adaptable (le service, les variantes de service, le contexte et les règles d'adaptation) ;
- Présentent un couplage fort entre les composants d'un SOSA ;
- Nécessitent tous l'intervention manuelle des développeurs, et ce malgré leur adoption de l'ingénierie dirigée par les modèles.

Par ailleurs, la nécessité d'un niveau d'abstraction plus élevé est toujours présente, surtout avec l'utilisation de l'ingénierie dirigée par les modèles, qui vise la génération totale de la solution finale à partir de modèles (Biehl, et al., 2014). Selon Kelly (Kelly, et al., 2008a), le développement dirigé par les modèles ne peut être que spécifique au domaine. Dans ce sens, plusieurs travaux ont essayé de proposer des langages de modélisation spécifiques aux SOSA en se basant sur le langage de modélisation universel UML (Boukadi, 2009), (Monfort, et al., 2009), (Abu-Matar, et al., 2011a), (Hafiddi, 2012), etc. Néanmoins, plusieurs critiques ont été apportées à l'utilisation des profils UML :

- Le manque de richesse sémantique (Saleem, et al., 2012) ;
- L'utilisation d'un profil UML nécessite une connaissance de base d'UML ce qui n'est pas le cas pour les équipes fonctionnelles (Oberortner, et al., 2009) ;
- La génération totale du code source en utilisant les profils UML nécessite la maitrise d'un langage d'actions et l'utilisation d'OCL (d'autres langages à maitriser par l'équipe fonctionnelle) (Iseger, 2006) ;
- Un profil UML hérite d'UML même les caractéristiques indésirables, soit parce qu'elles ne sont pas pertinentes ou qu'ils doivent être limitées pour un profil donné (López-Sanz, et al., 2012) ;
- La validation des modèles par rapport à la sémantique du domaine n'est pas permise par les environnements de modélisation (Dalgarno, et al., 2008).

L'extension d'un langage de modélisation universel est considérée comme étant une mauvaise pratique de la modélisation spécifique au domaine (Kelly, et al., 2009b). Par contre, la création d'un nouveau langage est plus facile que de personnaliser un langage existant (enlever des parties, ajouter de nouveaux concepts et sémantiques, etc.) (Kelly, et al., 2008a).

Malgré son apparition depuis la fin des années 1990, l'approche DSM n'a été utilisée que par de grandes compagnies (Iseger, 2010) spécialisées surtout dans la fabrication automobile (Long, et al., 1998) et les télécommunications (Kieburtz, et al., 1996). Cela était dû essentiellement au coût élevé de mise en place des solutions DSMs. A partir des années 2000, nous avons progressivement assisté à une démocratisation de l'utilisation de l'approche DSM. En effet, avec l'amélioration des plateformes et environnements de méta-modélisation facilitant la création des outils supportant l'approche DSM, le coût de mise en place d'une solution DSM a été nettement réduit (Iseger, 2006) (Kelly, et al., 2009b) (Achilleos, 2009) (ISIS, MIC, 2011).

A ce sujet, il importe de signaler que malgré l'accessibilité et les avantages de l'adoption de l'approche DSM (haut niveau d'abstraction, amélioration de la qualité et de la productivité, etc.), nous avons remarqué lors de notre étude bibliographique que cette approche est marginalement utilisée par les approches d'ingénierie des SOSA.

Il découle de ce qui précède que l'utilisation des langages de modélisation spécifiques indépendants de tout langage de modélisation universel, représente un choix prometteur pour l'ingénierie du logiciel en l'occurrence l'ingénierie des SOSA. Dans ce sens nous avons constaté un manque de :

- Approche d'ingénierie des SOSAs à base de DSLs ;
- Formalisme du processus de développement des systèmes spécifiques au domaine ;
- Processus de développement des SOSAs à base de DSLs.

I.4. Contributions

I.4.1. Approche d'ingénierie des systèmes orientés services adaptables à base de DSLs

L'objectif principal qui préside à notre thèse est la définition d'une approche d'ingénierie des systèmes orientés services adaptables à base de DSLs, une approche :

- Qui intègre l'ensemble des composantes d'un SOSA ;
- Caractérisée par un faible couplage entre les différentes vues d'un SOSA ;
- Générative, qui remplace pratiquement la notion de code source par modèles source.

Notre proposition intitulée CADSSOMA (***C**ontext **A**ware **D**omain **S**pecific and **S**ervice **O**riented **M**odeling **A**pproach*) est divisée en deux phases : la phase de modélisation et la phase de génération de code (Figure 1).

I.4.1.1. Phase de modélisation

La séparation des préoccupations représente la pierre angulaire de la flexibilité et de la gestion de la complexité des systèmes. Dans cette optique, notre approche assure une séparation entre la variabilité de service, le contexte et les règles d'adaptation. La phase de modélisation est basée sur cinq modèles (Figure 1) :

- Modèle des services spécifiques au domaine **(a)** : représente les services et les concepts du domaine spécifique. Nous avons défini une syntaxe abstraite générique, à étendre par les services et les concepts du domaine spécifique. Cela permet de produire la syntaxe abstraite du langage de modélisation des services spécifiques au domaine (Lethrech, et al., 2013) ;
- Modèle de la variabilité des services **(b)** : la modélisation de l'adaptation de service requière la prise en charge de la modélisation des variantes de services au niveau des premières étapes d'une approche de modélisation. Pour ce faire, nous avons adopté le mécanisme de modélisation explicite de la variabilité de service ;
- Modèle du contexte **(c)** : le type d'adaptation traité par notre approche est *designed* (Bucchiarone, et al., 2013) et sensible au contexte (Kazhamiakin, et al.,

2010). Par conséquent, nous avons opté pour un modèle dédié spécifiant les éléments du contexte qui influencent l'adaptation des services ;

- Modèle des règles d'adaptation **(d)** : l'adaptation traitée par notre approche est une adaptation manuelle (Canal, et al., 2006), c'est-à-dire, que la logique d'adaptation doit être spécifiée et mise en œuvre manuellement par des développeurs. Le modèle des règles d'adaptation permet de relier le modèle de la variabilité de service et le modèle du contexte. En d'autres termes, le modèle des règles d'adaptation permet de déterminer les variantes des services à utiliser pour une configuration donnée du contexte. Le modèle des règles d'adaptation est sous forme de tables de décision complètes à entrées mixtes (Pooch, 1974) ;
- Modèle du métier du domaine spécifique **(e)** : utilisé pour modéliser les règles métier du domaine cible. Il joue un rôle important dans la génération de la logique interne des services.

Les développeurs de la solution DSM doivent produire le méta-modèle (syntaxe abstraite) du métier du domaine spécifique, et le méta-modèle des services spécifiques au domaine. Ce dernier est une spécialisation de notre méta-modèle générique des services (Lethrech, et al., 2013). Retenons que les outils de modélisation de ces deux langages sont à développer et à intégrer dans l'outil support CADSSOTB (voir la section I.4.4).

I.4.1.2. Phase de génération de code source

Notre approche CADSSOMA adopte les concepts et les principes de la modélisation spécifique au domaine (DSM). En réalité, les modèles produits au niveau de la phase précédente représentent la matière première de cette phase. Le générateur de code, développé en utilisant un langage de transformation Modèle-à-Texte, transforme les cinq modèles de la phase de modélisation au code source final (Figure 1).

Il est à noter que les développeurs de la solution DSM doivent produire la syntaxe concrète du langage de modélisation du métier du domaine spécifique, ainsi que celui des services spécifiques au domaine. En outre, le générateur de code source de l'outil support CADSSOTB (voir la section I.4.4) doit être complété par la partie prenant en charge la transformation des modèles de ces deux DSLs.

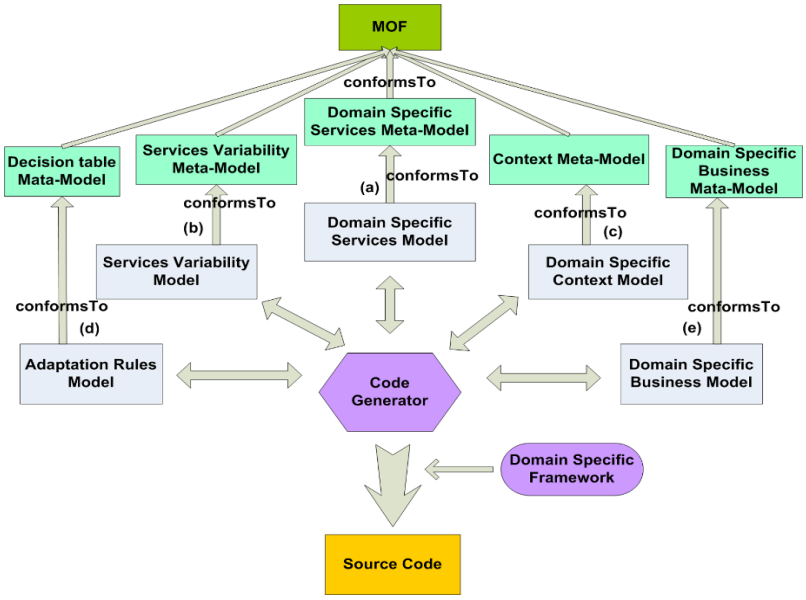


Figure 1 : Vue d'ensemble de l'approche CADSSOMA

I.4.2. Formalisme du processus de développement des systèmes spécifiques au domaine

Le processus de développement des systèmes spécifiques au domaine est très présent dans la littérature scientifique (Kelly, et al., 2008a) (Kelly, S.; MetaCase, 2009a) (Achilleos, 2009). En revanche, aucun formalisme regroupant l'ensemble des phases et artefacts de ce processus n'a été proposé. C'est pour cette raison que nous avons jugé intéressant de proposer un formalisme du processus DSM permettant d'initier les développeurs à l'ingénierie des langages spécifiques au domaine.

I.4.3. Processus de développement des SOSAs à base de DSLs

Il va sans dire que l'approche CADSSOMA joue un rôle éminent dans la construction des systèmes orientés services adaptables et spécifiques au domaine. Toutefois, un processus de développement normalisant leur développement n'en est pas moins appréciable. Le processus de développement des SOSA à base de DSLs proposé définit les phases, les artefacts et les activités nécessaires pour transformer la sémantique et les règles métier du domaine spécifique à des services adaptables, et ce conformément à l'approche DSM.

I.4.4. Outil support CADSSOTB

Finalement, nous avons validé notre approche en proposant un environnement de développement intégré (*CADSSO**TB, T**oolB**ox*) qui facilite la mise en œuvre de l'approche CADSSOMA. Après la spécification graphique des cinq modèles de la phase de modélisation, la boite à outils s'occupe de la génération automatique de la solution finale en se basant sur

une seule transformation Modèle-à-Texte. L'étude de cas utilisée correspond au domaine de calcul et restitution d'impôts.

I.5. Organisation du mémoire

La suite de ce rapport de thèse est organisée en quatre chapitres :

Chapitre II- Etat de l'Art. Initié par une introduction, ce chapitre est composé de quatre autres sections. La deuxième comprend une description des approches SOC, DSM et le concept d'adaptation. La troisième section présente les différentes approches de modélisation des SOSA (à base de profils UML et à base de DSLs), des discussions et une synthèse mettant en exergue les limites des propositions actuelles. La quatrième section s'intéresse aux technologies d'implémentation des services et de l'adaptation. Enfin, le chapitre est clôturé par une conclusion.

Chapitre III- CADSSOMA : Une approche de modélisation des systèmes orientés services adaptables basée sur l'ingénierie des langages spécifiques au domaine. Ce chapitre décrit notre approche d'ingénierie des SOSA à base de DSLs, intitulée CADSSOMA. Il est divisé en quatre sections. Après une introduction, la deuxième section illustre la syntaxe abstraite des DSLs utilisés. La troisième section est une étude de cas suivie par une conclusion.

Chapitre IV- Processus DSM pour le développement des SOSAs. Ce chapitre comporte essentiellement deux propositions. La première est un formalise du processus de développement des systèmes spécifiques au domaine. La deuxième est un processus DSM pour le développement des SOSA associé à notre approche CADSSOMA.

Chapitre V- CADSSOTB : Un environnement de développement des SOSAs spécifiques au domaine. Ce chapitre présente une validation pratique de notre approche CADSSOMA. Il illustre l'outil support CADSSOTB utilisé pour spécifier les cinq modèles de notre approche, et aussi pour générer le code source de la solution finale.

Chapitre VI- Conclusion et perspectives. Nous clôturons ce mémoire par une conclusion synthétisant les résultats de notre thèse, et un ensemble de perspectives qui nous semblent être les meilleures suites à notre travail.

Chapitre II

Etat de l'Art

II.1. Introduction

La philosophie de l'ingénierie dirigée par les modèles représente l'une des principales révolutions de l'ingénierie du logiciel. En fait, la solution finale résulte de la transformation des modèles de la phase de modélisation. La mise en place de cette doctrine a rencontré plusieurs difficultés, essentiellement la suppression de l'intervention manuelle des développeurs (MetaCase, 2012). L'approche DSM a essayé de relever ce défi, et ce, en élevant le niveau d'abstraction en se spécialisant à un domaine spécifique. Encore plus, d'autres chercheurs considèrent que l'ingénierie dirigée par les modèles ne peut être que spécifique au domaine (Kelly, et al., 2008a).

Une autre quête importante de l'ingénierie du logiciel est de trouver une solution qui simplifie le développement, supporte le principe de la réutilisation et facilite la communication et l'intégration des systèmes distribués. Service Oriented Computing (SOC) représente la dernière découverte de cette quête illusoire (Davis, 2009). SOC est un paradigme prometteur favorisant la réorganisation du système d'information sous forme d'un ensemble de services faiblement couplés (Caseau, 2011). La prépondérance du paradigme SOC a plusieurs avantages, entre autres : i) l'amélioration de l'agilité et la flexibilité du métier, ii) la facilitation de la gestion des processus métiers, iii) la favorisation de la réutilisation, iv) la facilitation de la maintenance, etc (Davis, 2009) (Josuttis, 2007) (Kumar, et al., 2009).

Par ailleurs, les développeurs des systèmes informatiques s'intéressent de plus en plus à la capacité d'adaptation, surtout avec la propagation de l'utilisation des dispositifs mobiles (Dey, 2001) et l'émergence de l'informatique sensible au contexte (CAC).

Le premier objectif de ce chapitre est d'adopter ou proposer des définitions, de fournir des descriptions, et d'introduire les différents concepts et paradigmes utilisés au fil de notre travail de recherche. Cela nous a permis d'identifier nos choix sur le plan conceptuel, architectural et technique.

Le deuxième objectif de ce chapitre est d'étudier les différentes Approches de modélisation des systèmes orientés services adaptables, et ce afin d'identifier leurs limites,

leurs points faibles et aussi leurs points forts. En plus des profils UML nous nous sommes intéressés spécialement à l'utilisation de l'approche DSM dans la mise en place des Systèmes Orientés Services Adaptables (SOSA).

Toute approche de modélisation doit reposer sur une plateforme technologique qui permet sa concrétisation dans le monde réel. Le troisième objectif de ce chapitre s'intéresse aux différents choix d'implémentation de l'architecture orientée services et aussi de l'adaptation.

Le présent chapitre est organisé comme suit. La deuxième section a pour but de définir succinctement les concepts de base utilisés dans le cadre de notre travail, en l'occurrence SOC, DSM et Adaptation. La troisième section illustre les approches de modélisation des SOSAs étudiées. Cette dernière est divisée en deux sous sections, approches à base de DSL et approches à base de profils UML. Les technologies d'implémentation de l'architecture Orientée services et de l'adaptation sont traitées dans la quatrième section. Enfin, ce chapitre est clôturé par une conclusion générale.

II.2. Notions et concepts de base

L'Informatique Orientée Services (SOC), la Modélisation Spécifique au Domaine (DSM) et l'Adaptation sont les principaux concepts utilisés dans le cadre de notre travail de recherche. Nous en jetons quelque lumière dans cette section.

II.2.1. Le paradigme SOC

Le paradigme SOC a été initialement utilisé comme une technique visant l'intégration des systèmes d'information en essayant de résoudre le problème d'interopérabilité (Stojanovic, 2005). Ensuite, ce concept a été adopté même lors de l'urbanisation des nouveaux systèmes d'information, en les organisant sous forme de services autonomes, interopérables et faiblement couplés.

Certes, le paradigme SOC favorise la réutilisation, la flexibilité et le faible couplage des solutions informatiques, à l'instar de ses prédécesseurs POO et CBD. Par ailleurs, pratiquement beaucoup de différences existent entre le paradigme SOC et les autres paradigmes. Entre autres, nous citons le niveau de granularité, le type de communication, le type de couplage, la manière de consommation, la composition, la découverte et la sélection, la possibilité d'étendre la disponibilité à l'extérieur des frontières de l'entreprise, etc. Dans ce sens d'autres chercheurs (Stojanovic, 2005) considèrent que la notion de service est une évolution de la pensée composant.

Avant d'entamer la description du paradigme SOC il importe de signaler que la majorité de sa documentation étudiée (Davis, 2009) (Josuttis, 2007) (Kumar, et al., 2009) (Caseau, 2011) traite l'ensemble des aspects de ce paradigme (technologie, architecture, orchestration,…) en confondant SOC et SOA. Pour lever cette ambiguïté nous avons utilisé la structuration du concept SOC proposée par (Boukadi, 2009) (Figure 2). Cette dernière met le service au centre des autres volets. En effet, quatre vues complémentaires sont à détailler, à savoir : (i) la vue architecture, (ii) la vue technologique, (iii) la vue méthode de mise en place et (iv) la vue composition.

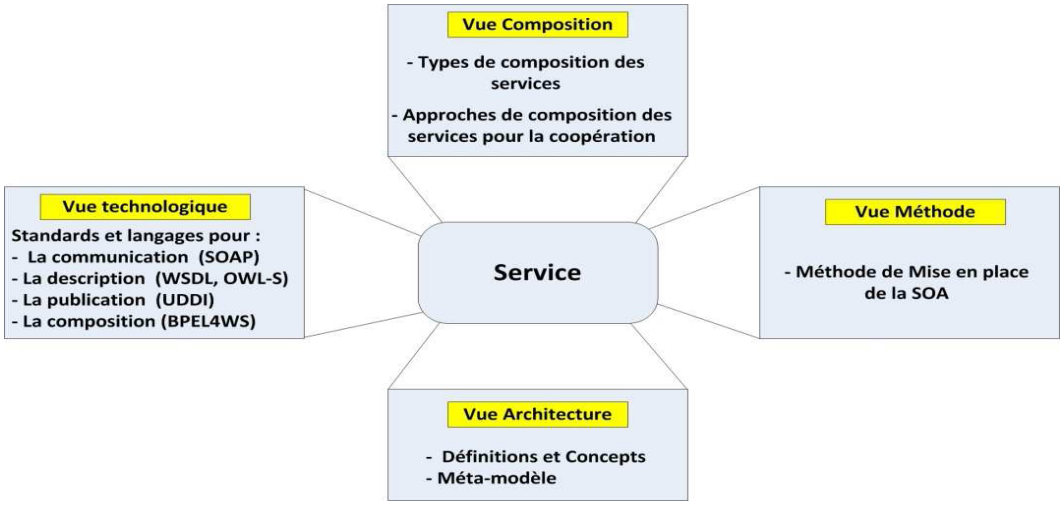


Figure 2 : Vision multi-vues du paradigme service (Boukadi, 2009)

Avant de décrire ces différents axes, nous allons présenter quelques définitions du concept service au niveau de la section suivante.

II.2.1.1. Définition du concept « Service »

Le but de cette section est de bien appréhender la notion de service et de fournir une définition de base qui sera adoptée dans le reste de ce manuscrit. Pour ce faire, nous avons catégorisé les définitions étudiées en deux visions : la vision métier et la vision informatique.

En ce qui concerne la vision métier, selon le dictionnaire Larousse un service est une activité professionnelle exercée dans une entreprise ou une administration. Pour (Cauvet, et al., 2008) un service est une unité réutilisable encapsulant un fragment d'un processus métier et visant la satisfaction d'un but métier.

Selon Kumar et al. (Kumar, et al., 2009) un service métier est un ensemble d'actions ou tâches offertes par une organisation à différents intervenants. Il précise aussi que le cycle de vie d'un produit ou d'un service implique les étapes suivantes: i) Création de produits ou

services avec les moyens de marchandisage[1] ; ii) Découverte de produits et services par les clients ; iii) Les fournisseurs et les clients confluent et négocient ; iv) Vente de produits et services ; v) Maintenance des produits et des services.

Quant à la définition informatique, OASIS (OASIS, 2006) considère un service comme un moyen permettant l'accès à une application en utilisant une interface. Pour Josuttis (Josuttis, 2007) un service est la réalisation informatique de certaines fonctionnalités métier indépendantes.

De ce qui précède découle que la définition du concept *service* diffère selon l'angle de vue de celui qui la propose. Nous soulignons que la vue métier et la vue technique permettent de démontrer que le concept service offre un meilleur alignement entre la logique métier et la logique applicative.

En conclusion, nous considérons qu'un service est un composant informatique indépendant qui réalise une fonctionnalité métier élémentaire et qui participe à une architecture Orientée services.

II.2.1.2. Vue architecture

L'Architecture Orientée Services (SOA) ne représente que le volet architectural du concept SOC. Plusieurs définitions et perceptions de SOA existent dans la littérature scientifique. Comme le confirme Boukadi (Boukadi, 2009), la plupart de ces définitions se focalisent principalement sur le volet technique d'une architecture Orientée services. Très peu de définitions qui considèrent le volet métier. Nous adoptons la définition d'Arsanjani (Arsanjani, 2004) qui considère que « SOA est un style architectural où les capacités logicielles sont exposées sous formes d'un ensemble de services ». Il affirme aussi que « SOA favorise l'alignement métier-IT, l'amélioration de l'agilité des entreprises et la réduction des coûts d'intégration ».

D'une façon générale trois rôles sont à distinguer dans une architecture SOA : le client du service, le fournisseur de services et le registre de service (Figure 3). Le fournisseur de services produit le service, ensuite il publie sa description au niveau du registre de service. La description d'un service présente les opérations offertes ainsi que leur mode de consommation. Le client effectue ses recherches dans le registre pour trouver le service

[1] Le marchandisage est l'orthographe francisée du merchandising, une branche du marketing spécialisée dans la grande distribution.

requis. La dernière étape consiste en l'établissement d'une liaison entre le consommateur et le fournisseur permettant l'invocation du service.

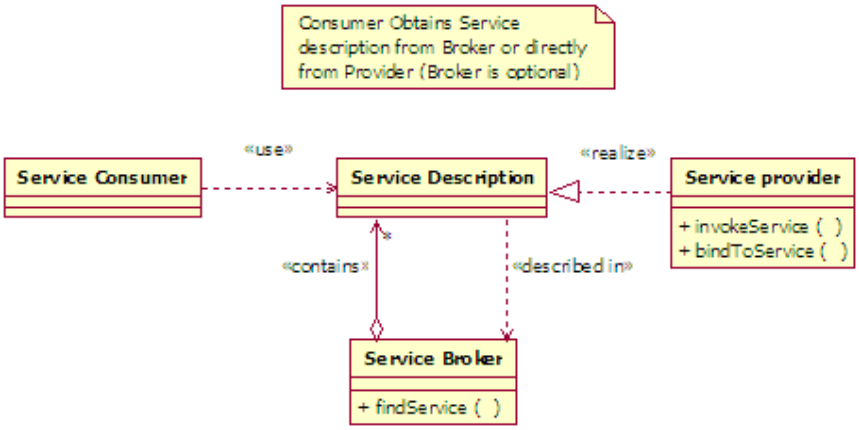


Figure 3 : Méta-modèle du style d'architecture SOA (Arsanjani, 2004)

II.2.1.3. Vue technologique

Comme le confirme Josuttis (Josuttis, 2007), plusieurs définitions confondent SOC et les services web. En fait, SOC est un paradigme et les services web représentent l'une de ses implémentations possibles. Il est possible de mettre en œuvre une SOA en utilisant d'autres technologies (CORBA, MQ, Tibco, etc.) et d'utiliser des services Web sans SOA. En d'autres termes, les services web offrent l'infrastructure technologique, mais l'architecture reste toujours à concevoir et à mettre en place (Josuttis, 2007).

Cela dit, les services web restent l'implémentation la plus prometteuse de l'architecture SOA, sa large utilisation dans le monde industriel en est le meilleur indicateur. Plus de détail sur cette technologie existe dans la section Technologies d'implémentation.

II.2.1.4. Vue méthode de mise en place d'un SOS

« La vue méthode de mise en place s'intéresse essentiellement à la manière avec laquelle il faut mener, suivant une démarche raisonnée, un processus de mise en œuvre d'une architecture Orientée services » (Boukadi, 2009). Nous exposons dans ce qui suit un résumé de quelques méthodes de mise en place de SOA.

SOMA (Arsanjani, et al., 2008), proposée par IBM, est divisée en trois phases :

- **Identification des services :** les trois démarches classiques d'identification des services sont utilisées, à savoir la démarche descendante (*Top-Down*) basée sur les cas d'utilisation et les processus métiers, la démarche ascendante (*Bottom-Up*) pour la réutilisation du patrimoine IT existant et la démarche hybride (*Middle-out*) qui

s'occupe de l'identification des services non pris en charge par les démarches précédentes.

- **Spécification :** la principale activité de cette phase consiste en l'identification des services candidats pour l'étape suivante. Et ce, en se basant sur des normes bien déterminées, entre autres nous citons: l'alignement métier, la facilité de l'implémentation, la factorisation en éliminant les redondances, la réutilisation, etc.
- **Réalisation :** implémentation des services.

(Papazoglou, et al., 2006) proposent un processus de développement des services Web. Il est composé de six phases :

- **La planification :** identifie le périmètre, la nature, le retour sur investissement, la faisabilité de la solution SOA dans le contexte de l'entreprise.
- **Analyse et de spécification :** permet de préparer les modèles de processus métiers existants « *as-is* » et à mettre en place « *to-be* ». la deuxième étape consiste en la spécification des services et des processus métiers.
- **Construction et test :** s'occupe de l'implémentation des services ainsi que la détermination de leurs interfaces.
- **Provisionning :** permet de tester les services lors de leurs utilisations.
- **Déploiement :** consiste en la publication des interfaces des services au niveau d'un registre de services.
- **Exécution et de supervision :** exploitation des services et évaluation des performances.

Zimmermann et al. (Zimmermann, et al., 2004) ont proposé une approche intitulée SOAD (*Service-Oriented Analysis and Design*). « L'originalité de leur approche réside dans l'utilisation des techniques déjà existantes comme la conception orientée objet, l'architecture d'entreprise et la modélisation des processus d'entreprise ain de bâtir une approche SOA au sein de l'entreprise » (Boukadi, 2009).

Les auteurs de cette méthode proposent d'utiliser une approche hybride pour l'identification des services. En effet, ils combinent à la fois l'analyse des processus métiers afin de définir les services nécessaires à leurs réalisations et l'analyse de l'existant pour déterminer les fonctions, modélisations, infrastructure existantes du système d'information.

Chang et al. (Chang, et al., 2007a) (Chang, et al., 2007b) proposent une approche de spécification et d'analyse pour le développement des services adaptables. Cette dernière est divisée en cinq phases :

- La première phase s'occupe de la définition des services cibles en se basant sur les processus métiers de l'entreprise.
- Phase 2 : cette phase vise la définition et la conception des services unitaires à partir du résultat de la première phase.
- Phase 3 : Trois types de services à identifier : les services offerts par le SI existant (ajout de façades ou de médiateurs), les services à développer et les services à consommer en utilisant des registres de service externes.
- Phase 4 : L'objectif de cette phase est de développer les nouveaux services, développer des adaptateurs pour exploiter les services existants et de considérer les services externes prêts pour utilisation.
- Phase 5 : cette phase prend en charge la composition des services identifiés par les phases précédentes, et ce afin de produire les processus métiers de l'entreprise.

Les auteurs associent au niveau de chaque phase les types de variabilités correspondantes (variabilité workflow, interface, logique, etc.).

II.2.1.5. Vue de composition de services

Comme dit l'adage "*Ayant divisé pour conquérir, nous devons réunir pour régner*" Jackson (1990) et comme le confirme Stojanovic (Stojanovic, 2005), n'importe quelle stratégie de séparation de préoccupations sous forme de services particuliers doit aussi fournir des mécanismes d'intégration homogène, cohérente et significative de ces services.

Vu le faible couplage des architectures orientées services, les services ont l'avantage d'être facilement composables. Casati (Casati, et al., 2002) définit la composition des services Web par l'aptitude de produire des services à forte valeur ajoutée, tout simplement, en combinant des services existants. Il s'agit d'identifier les services à invoquer, leur ordre d'invocation et les conditions d'exceptions.

A ce sujet il importe de signaler que le niveau de granularité des services a un impact direct sur leur composition. En réalité, plus la granularité est fine plus la composition devient complexe.

Plusieurs approches de composition de services existent dans la littérature (He, et al., 2015) (Rosenberg, et al., 2009b) (Baidouri, et al., 2012) (Sun, et al., 2010a) etc. Le détail de ces

approches qui bien que loin d'être inutile, semble tout de même moins important dans le contexte de notre travail.

II.2.1.6. Synthèse

Au niveau de cette section nous avons essayé de fournir des définitions et de lever quelques ambigüités concernant les différents aspects de l'informatique orientée services, l'un des principaux paradigmes utilisé dans le cadre de notre travail.

Dans le reste de ce manuscrit nous considérons que :

- SOC est un paradigme qui utilise les services comme éléments de base pour soutenir le développement rapide, à faible coût des applications distribuées et facilement composables.
- Service est un composant informatique indépendant qui réalise une fonctionnalité métier élémentaire.

En outre, il est à souligner que SOA n'est que la vue architecturale du paradigme SOC et les services web ne représentent que l'une de ses implémentations possibles.

En ce qui concerne les méthodes de mise en place, l'urbanisation d'un système d'information selon l'approche SOC doit impérativement passer par l'analyse et la conception des processus métiers. Cela permet de les optimiser, d'identifier les services candidats, et de construire un référentiel métier de l'entreprise. En revanche, l'approche ascendante qui tient compte de l'actif IT de l'entreprise est nécessaire dans le cas de l'intégration du concept service dans un SI existant. Dans ce sens il est judicieux d'étudier la possibilité d'exploiter, sous forme de services, les composants déjà implémentés.

Pour conclure le paradigme SOC offre plusieurs avantages tant sur le plan métier que sur le plan technique. En effet, il améliore l'agilité et la flexibilité du métier et permet un alignement métier-IT basé sur la modélisation des processus métiers. Ces derniers sont regroupés dans un manuel de procédures qui représente un référentiel métier de l'entreprise. D'autre part l'organisation du système d'information sous forme de service permet de réduire la complexité (faible couplage), favorise la réutilisation et facilite la maintenance et l'intégration des Systèmes d'information (Kumar, et al., 2009) (Davis, 2009).

II.2.2. Modélisation Spécifique au Domaine (DSM)

Dans un processus de développement traditionnel, appelé aussi ingénierie dirigée par le code, les modèles sont utilisés pour concevoir les systèmes, faciliter leur compréhension, et

documenter la solution finale. Les modèles et le code source sont totalement séparés, et assez souvent désynchronisés. En revanche, l'Ingénierie Dirigée par les Modèles (IDM[1]) accorde aux modèles une place importante dans le processus d'ingénierie du logiciel, on parle de modèle source au lieu de code source (Kelly, et al., 2008a). En fait, le code source exécutable n'est qu'un élément qui émane d'une transformation de modèles (Diaw, et al., 2008).

La modélisation, pratique qui ne date pas d'aujourd'hui[2], vise principalement l'un des deux objectifs suivants: abstraction d'une réalité existante ou bien spécification d'un système à réaliser. Dans le contexte de l'ingénierie dirigée par les modèles, un modèle est essentiellement utilisé pour générer du code et construire un référentiel métier de l'entreprise.

Plusieurs approches IDM ont été proposées par la communauté scientifique : MDA représentant la vison de l'OMG, les usines logicielles, exécutable UML, etc. L'une des dernières révolutions de l'IDM est la modélisation spécifique au domaine (DSM). Malgré son apparition depuis la fin des années 1990, l'approche DSM n'a été utilisée que par de grandes compagnies spécialisées essentiellement dans la fabrication automobile (Long, et al., 1998) et les télécommunications (Kieburtz, et al., 1996).

Dans la suite de cette section nous allons présenter une description des différentes facettes de l'approche DSM : définitions, principe de base, architecture, apport, etc.

II.2.2.1. Définitions

DSM est une méthodologie d'ingénierie du logiciel pour la modélisation et le développement des systèmes. Kelly (Kelly, et al., 2008a) définit l'approche DSM par:

Domain-Specific Modeling mainly aims to do two things. First, raise the level of abstraction beyond programming by specifying the solution in a language that directly uses concepts and rules from a specific problem domain. Second, generate final products in a chosen programming language or other form from these high level specifications.

Cette définition qui est proche de celle de DSM Forum (DSM Forum, 2016) considère que l'approche DSM vise principalement deux objectifs : i) l'élévation du niveau d'abstraction en modélisant la solution par un langage spécifique (DSL); ii) la génération des produits finaux à partir de ces spécifications de haut niveau.

[1] *Model Driven Engineering (MDE)* dans le vocabulaire anglo-saxon.
[2] En 1486 Leonard de vinci a modélisé la vis aérienne, considérée comme la base de l'hélicoptère (wikipedia).

La définition de l'approche DSM nous conduit à définir ce qu'est un langage spécifique au domaine (Domain Specific Language). Ci-dessous deux définitions retenues :

Définition proposée par Van Deursen :

"A domain-specific language (DSL) is a programming language or executable specification language that offers, through appropriate notations and abstractions, expressive power focused on, and usually restricted to, a particular problem domain" (Van Deursen, et al., 2000).

Définition proposée par Chen, et al. :

"A DSML is a modeling language that is tailored to particular constraints and assumptions of an application domain. The domain concepts are represented by language primitives, so the system can be modeled as it exists" (Chen, et al., 2004).

Nous retenons de ce qui précède qu'un DSL est un langage de programmation spécifique à un domaine restreint. Il utilise directement les concepts et les règles d'un problème lié au domaine spécifique. Il est naturellement très riche sémantiquement, ce qui rend la modélisation d'un système la plus proche possible du monde réel.

Il est à signaler qu'une confusion terminologique entre DSM et DSSD (Domain Specific Software Development) existe dans la littérature. DSM pour Sánchez-Ruíz et al. (Sánchez-Ruíz, et al., 2007) est le processus de construction des modèles d'un domaine spécifique, il ne concerne que la partie modélisation. Pour Kelly (Kelly, et al., 2008a) l'approche DSM fournit une solution complète du domaine spécifique, elle inclut le langage de modélisation, l'outil de modélisation, le générateur du code source et le framework du domaine spécifique. Selon Sánchez-Ruíz et al. (Sánchez-Ruíz, et al., 2007) cela s'appelle *Domain Specific Software Development* (DSSD). Il importe aussi de souligner que selon Bonnet (Bonnet, 2005) l'approche DSM est aussi équivalente à la « Programmation générative » (GP) visant la génération totale des systèmes à partir des modèles. Dans le reste de ce document nous considérons que DSM, DSSD et GP sont des synonymes.

II.2.2.2. Principe de base

Il a été reconnu depuis de nombreuses années qu'il existe un grand écart entre le domaine du problème d'une application et son code (Kelly, S.; MetaCase, 2009a) (Bonnet, 2005). Ce sont deux mondes différents, chacun avec son propre langage, ses experts, sa façon

de penser, etc (Figure 4). L'une des tâches les plus délicates pour un ingénieur logiciel est de construire un pont entre ces deux mondes, tout en résolvant les problèmes de chacun (MetaCase, 2012). L'approche DSM vise principalement la résolution de ce problème, et ce, en utilisant le mécanisme de transformation (Ortiz, 2012). L'espace problème représente un ensemble d'abstractions via une notation spécifique à un domaine et l'espace solution représente un ensemble d'abstractions technologiques, représentant l'implémentation du système spécifié et basé sur les artefacts de l'espace problème.

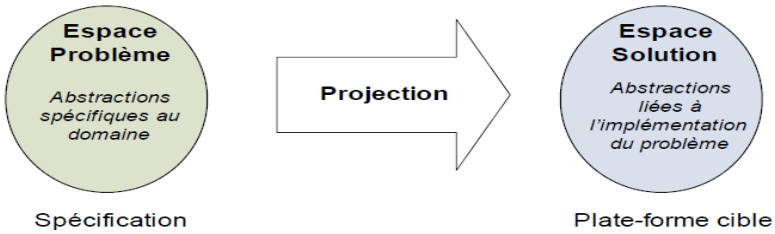


Figure 4 : Approche DSM : Projection entre l'espace problème et l'espace solution (Bonnet, 2005)

II.2.2.3. Architecture d'une solution spécifique au domaine

Une solution DSM est composée de trois principaux éléments (Figure 5) :

- Le langage spécifique au domaine (DSL) : Permet de modéliser les concepts du domaine spécifique. Un DSL est composé d'une syntaxe abstraite et d'une syntaxe concrète (Langlois, et al., 2007). La création d'un DSL nécessite aussi la création d'un outil de modélisation permettant de faciliter la création et la validation des modèles du domaine spécifique.
- Le générateur du code source : en général développé en utilisant un langage de transformation « Modèle-à-Texte ». Il permet de transformer les modèles spécifiques au code source de la plateforme cible.
- Le framework du domaine : permet de factoriser le code généré, dissimule l'environnement cible et facilite l'intégration avec le code existant. La taille du framework peut varier selon les cas (Figure 5).

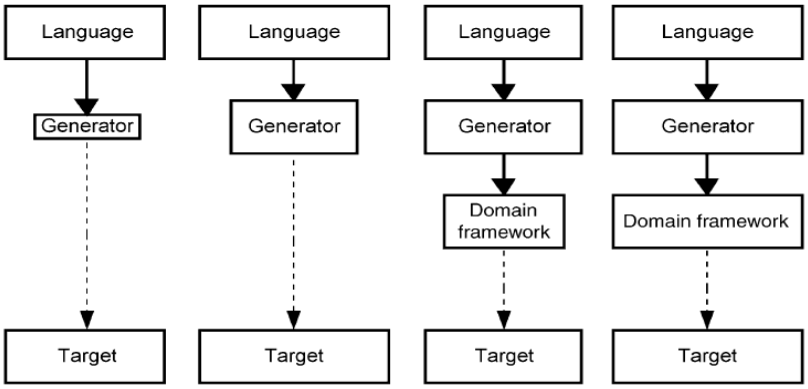


Figure 5 : Architectures d'une solution DSM (Kelly, et al., 2008a)

II.2.2.4. Apport de l'approche DSM

II.2.2.4.1. Productivité

L'approche DSM représente une réelle révolution de la productivité qui est similaire au passage de l'assembleur vers les langages de haut niveau. Elle permet une amélioration considérable de la productivité en comparaison avec les langages de modélisation universels et le développement manuel (Kärnä, et al., 2009) (MetaCase, 2012) (Iseger, 2006). Par exemple, pour UML, la meilleure amélioration de la productivité par rapport au développement manuel est estimée à 35% selon une étude d'un sponsor de l'approche MDA (Kelly, S.; MetaCase, 2009a).

Le principal fruit de l'augmentation de la productivité réside dans la réduction du délai de mise sur le marché[1], arme stratégique pour la compétitivité des entreprises.

II.2.2.4.2. Qualité

DSM permet de produire des applications avec une qualité meilleure, et ce pour deux raisons (Kelly, et al., 2008a) : i) Premièrement, le langage de modélisation peut inclure des règles de validation des modèles, permettant l'interdiction de création de spécifications illégales. Une fois un développeur expérimenté définit les règles du domaine au niveau d'un DSL, il impose les directives de développement et les meilleures pratiques pour tous les autres développeurs ; ii) Deuxièmement, l'automatisation de la correspondance entre le problème du domaine, représenté via des modèles spécifiques, et la solution qui est généralement sous forme de code source, permet de réduire le risque d'erreur de l'implémentation manuelle (MetaCase, 2012).

[1] *Time to market* dans le vocabulaire anglo-saxon.

II.2.2.4.3. Valorisation de l'expertise métier

Souvent, la capacité de recueillir et formaliser les connaissances du domaine est seule considérée comme précieuse, même en l'absence d'un générateur de code (Kelly, et al., 2008a). Cela permet de construire un référentiel métier de l'entreprise, facilitant ainsi l'intégration de nouveaux développeurs.

II.2.2.5. Quand utiliser l'approche DSM ?

Une solution DSM nécessite un investissement initial important. Cela implique que le travail au niveau du même domaine s'étalera sur une longue période. Encore plus, à l'instar de SPLE, l'approche DSM permet de raisonner non plus en systèmes individuels mais en familles de systèmes. Cela permet de placer concrètement la réutilisation au centre des préoccupations du processus de développement. DSM est donc une option moins probable pour les entreprises qui travaillent sur des projets à court terme sans avoir une visibilité sur les domaines des futures applications de leurs clients. De même, DSM est moins approprié pour des entreprises ayant focalisé leurs compétences au niveau d'un langage de programmation plutôt qu'un problème d'un domaine donné (Kelly, et al., 2008a). L'amortissement de l'investissement initial de la mise en œuvre d'une solution DSM ne sera visible qu'après quelques utilisations de la solution DSM. Pour illustrer ce qui précède, nous tenons à signaler qu'une ligne de produits ne sera rentable que si elle est capable de produire plus de trois variantes de produits (Weiss, et al., 1999).

Il importe de signaler, qu'avec l'amélioration des plateformes et environnements de méta-modélisation facilitant la création des outils supportant l'approche DSM, et ce à partir des années 2000, le coût initial de mise en place des solutions DSM a été nettement réduit (Iseger, 2006) (Achilleos, 2009) (ISIS, MIC, 2011). Entre autres nous citons : MetaEdit+ de Mata-Case, les outils du projet de modélisation d'Eclipse (EMF, GEF, GMF, etc.), Sirius, Spray.

II.2.2.6. Synthèse

DSM est une méthodologie d'ingénierie du logiciel pour la modélisation et le développement des systèmes en utilisant des langages spécifiques appelés DSLs. La syntaxe concrète de ces derniers représente les concepts du domaine spécifique. En outre, et comme déjà cité au niveau de la section II.2.2.1, DSM, DSSD, Ingénierie des langages spécifiques au domaine et programmation générative sont tous des synonymes. Nous utilisons dans le reste de ce manuscrit l'abréviation DSM.

Pour conclure, une forte liaison sémantique des langages avec le domaine en question est une condition sine-qua-none de l'élévation du niveau d'abstraction (Kelly, et al., 2008a). En outre, la focalisation sur un domaine d'intérêt restreint permet de faire une correspondance du langage qui soit la plus proche possible du problème traité. En plus, la génération complète du code source devient une tâche réaliste, chose qui est difficile, sinon impossible à réaliser avec des langages de modélisation universels. Selon Kelly (Kelly, et al., 2008a), le développement dirigé par les modèles ne peut être que spécifique au domaine. Il affirme aussi que l'ensemble des cas industriels où les modèles sont utilisés effectivement comme l'artefact de développement de base, les langages de modélisation étaient spécifiques au domaine et généralement non rendus publics.

II.2.3. SOS : Concepts d'adaptation et de contexte

Récemment, l'adaptation des logiciels, particulièrement ceux à base de services, a reçu beaucoup d'intérêt de la part de la communauté de recherche (Lane, et al., 2012) (Hafiddi, 2012) (Esfahani, et al., 2011) (Perez, et al., 2011) (Schmidt, et al., 1999) (Chen, et al., 2000). Cela est dû essentiellement à la croissance du besoin d'adaptation surtout avec l'utilisation massive des dispositifs mobiles qui sont fortement reliés à leurs situations contextuelles. Effectivement, l'adaptation d'un système passe inévitablement par l'utilisation de sous-systèmes capables d'analyser le contexte, de détecter ses changements et de déclencher les actions correspondantes. Dans le reste de cette section nous allons jeter quelques lumières sur le concept d'adaptation.

II.2.3.1. Définitions

II.2.3.1.1. Définition du concept d'adaptation

Le dictionnaire Larousse définit le verbe *adapter* par « rendre un dispositif apte à assurer ses fonctions dans des conditions particulières ou nouvelles ». En considérant la définition sémantique du terme, une adaptation correspond au processus de modification du système, indispensable pour permettre un fonctionnement adéquat dans un contexte donné.

II.2.3.1.2. Définition du contexte

Le concept *contexte* suscite un intérêt croissant de la part de la communauté de recherche spécialisée dans le domaine du développement sensible au contexte (Schmidt, et al., 1999). Plusieurs définitions de ce concept ont été proposées dans la littérature scientifique.

Chen et al. (Chen, et al., 2000) décrivent le contexte par un ensemble d'états et des paramètres de l'environnement qui déterminent le comportement d'une application ou au sein duquel un événement de l'application, intéressant pour l'utilisateur, se produit. L'encyclopédie Larousse définit le contexte par « un ensemble des circonstances dans lesquelles se produit un événement, se situe une action ».

Une autre définition qui est l'une des plus référencée dans la littérature scientifique est celle de Dey :

"Context is any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves" (Dey, 2001).

Une autre définition à laquelle nous nous sommes intéressés est celle de Winograd et al. (Winograd, 2001). Elle perçoit le contexte comme l'ensemble d'informations structuré et partagé qui évolue et sert l'interprétation. Elle précise que seule la manière dont l'information est utilisée permet de catégoriser une information comme élément du contexte.

II.2.3.1.3. Définition du système adaptable

Schilit et al. considèrent que « l'informatique sensible au contexte[1] permet de produire un système qui s'adapte en fonction de son lieu d'utilisation, des personnes et des objets à proximité, ainsi qu'aux changements affectant ces objets au fil du temps » (Schilit, et al., 1994). Plusieurs autres définitions ont été proposées. Nous citons par exemple celle de Dey qui est presque identique à la définition de Strang et al. (Strang, et al., 2003), il considère un système adaptable, « s'il utilise le contexte pour fournir des informations et/ou des services pertinents pour l'utilisateur, où la pertinence dépend de la tâche demandée par l'utilisateur » (Dey, 2001). Aussi Xiaohang stipule que « l'adaptation d'un système logiciel est sa capacité à acquérir, gérer, interpréter et répondre aux changements du contexte afin de fournir les services appropriés » (Xiaohang, 2003).

II.2.3.2. Ingrédients de l'adaptation

Plusieurs travaux ont essayé de proposer une vision globale des différents concepts d'adaptation (Boukadi, 2009) (Bucchiarone, et al., 2010a) (Bucchiarone, et al., 2010b) (Kazhamiakin, et al., 2010) (Ayed, et al., 2007) (Canal, et al., 2006). Kazhamiakin, et al. (Kazhamiakin, et al., 2010) ont proposé une classification des concepts d'adaptation structurées de manière à répondre aux questions «Why ?», «What ?» and «How ?» (Figure 6).

[1] *Context Aware Computing* dans le vocabulaire anglo-saxon.

- La dimension «Why ?» : fournit une description de la motivation de l'adaptation ;
- La dimension «What ?» : utilisée pour décrire le sujet de l'adaptation ;
- La dimension «How ?» : décrit la façon dont l'adaptation sera réalisée et mise en œuvre.

II.2.3.2.1. La dimension « Why » ?

Elle est divisée en cinq catégories :

- Adaptation perfective : vise à améliorer l'application même si elle fonctionne correctement. Par exemple, pour optimiser ses caractéristiques de qualité ;
- Adaptation corrective : vise à supprimer un mauvais comportement d'une application Orientée services, par exemple, en remplaçant un service par une nouvelle version qui fournit la même fonctionnalité ;
- Adaptation Adaptative : modifie le comportement de l'application en réponse aux changements affectant son environnement. Le besoin de ce genre d'adaptation des applications orientées service est dicté par i) la nécessité d'adapter l'application aux changements de son contexte d'exécution (context-aware adaptation); ii) la nécessité d'assurer l'interopérabilité entre les parties interagissantes en fournissant des médiateurs appropriés ; iii) la nécessité de personnaliser l'application selon les besoins et les exigences de l'utilisateur (Kazhamiakin, et al., 2010) (Bucchiarone, et al., 2010a) ;
- Adaptation préventive : vise à prévenir les défauts futurs avant qu'ils ne surviennent ;
- Adaptation d'extension : étend l'application en ajoutant de nouvelles fonctionnalités nécessaires.

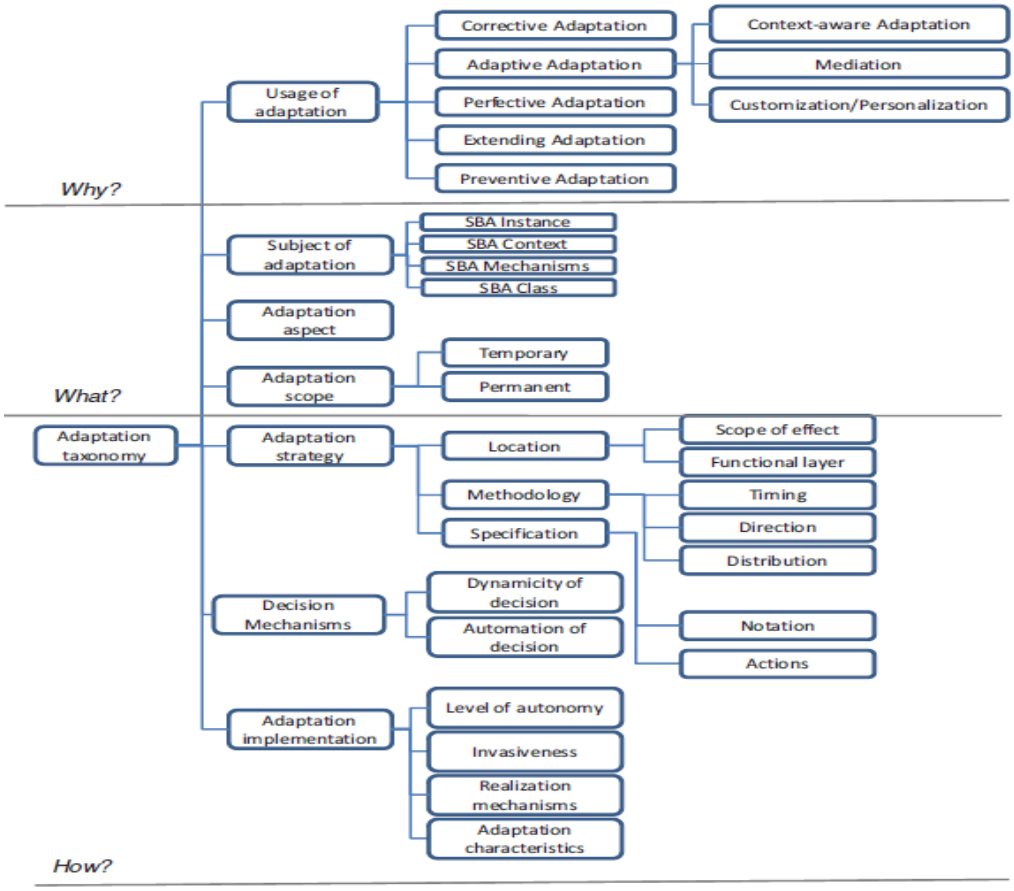


Figure 6 : Taxonomie[1] de l'adaptation (Kazhamiakin, et al., 2010)

II.2.3.2.2. La dimension « What ? »

Cette dimension est divisée en : Objet de l'adaptation (*Subject of Adaptation*), Aspect de l'adaptation (*Adaptation Aspect*) et Portée de l'adaptation (*Adaptation Scope*).

- Objet de l'adaptation : représente l'entité qui devrait être modifiée par le processus d'adaptation ;
- Aspect de l'adaptation : concerne une préoccupation particulière du processus d'adaptation, par exemple un modèle de qualité (sécurité, fiabilité, facilité d'utilisation, etc.), fonctionnalité donnée, etc ;
- Portée de l'adaptation : concerne la portée du processus d'adaptation (adaptation temporaire ou adaptation permanente).

II.2.3.2.3. La dimension « How ? »

Cette dimension est divisée en trois catégories :

[1] Désigne une méthode de classification des informations dans une architecture structurée de manière évolutive. Elle est à l'origine issue d'une branche de la biologie qui s'intéresse à la classification scientifique des espèces (wikipedia).

- Stratégie d'adaptation : représente la façon avec laquelle l'adaptation est accomplie (*re-configuration*, *re-binding*, *re-execution*, *re-planning*, etc.). Elle est classée comme suit :
  - Emplacement : elle concerne i) la portée de la modification qui est soit locale (service, adaptation spécifique à un utilisateur, etc.) soit globale (l'ensemble des utilisateurs d'un service, etc.) et ii) le lieu de la modification : la signature des services, les protocoles, l'environnement d'exécution, la composition de services, etc ;
  - Méthodologie d'adaptation : elle caractérise i) le moment de l'adaptation (réactive, proactive, post-mortem) ; ii) la distribution de l'adaptation (centralisée et distribuée) et iii) la direction de l'adaptation (vers un nouvel état du système, ou vers un état précédemment connu) ;
  - Spécification de l'adaptation : spécifie les actions particulières des stratégies d'adaptation. Actions concrètes à réaliser, la description des objectifs à atteindre (*goal-based*), variabilité explicite (points de variation, variantes), ou hybride.
- Mécanisme de décision : il est caractérisé par :
  - La dynamique de la décision : les auteurs distinguent entre *static selection* (la stratégie d'adaptation est prédéfinie), *dynamic selection* (la stratégie d'adaptation est choisie lors de l'exécution) et *evolution-based selection* (équivalente à la sélection dynamique avec exploitation de l'historique d'adaptation) ;
  - L'automatisation de la décision : elle caractérise le degré d'intervention humaine dans le processus de décision. Il peut varier de totalement automatique à interactive.
- Implémentation de l'adaptation : définit la façon avec laquelle un système adaptable est mis en œuvre. Elle caractérise le degré de l'intervention humaine, le couplage entre le sujet de l'adaptation et la plateforme d'exécution, les outils et les mécanismes permettant de faciliter la mise en place de l'adaptation, d'autres caractéristiques du processus d'adaptation tels que la sécurité, l'optimalité, le coût, la performance, etc.

II.2.3.3. Catégorisation de l'adaptation

En plus de la catégorisation de Kazhamiakin et al. (Kazhamiakin, et al., 2010) (voir la section II.2.3.2.1), d'autres catégorisations de l'adaptation sont proposées par la communauté de recherche. En effet, Bucchiarone et al. (Bucchiarone, et al., 2013) différencient entre « *at run-time adaptation* » et « *designed adaptation* ». La première pour une adaptation à la volée, la deuxième nécessite l'analyse de toutes les possibilités d'adaptation lors de la conception du système. Khouloud (Boukadi, 2009) quant à elle, a proposé trois types d'adaptation : adaptation réflexive, adaptation contrôlée par les politiques et adaptation par tissage d'aspects.

La première représente la capacité d'un système d'observer et d'agir sur lui-même lors de son exécution. La deuxième se compose de règles permettant au système d'identifier les actions à exécuter pour un contexte donnée. La troisième représente une solution technique pour implémenter l'adaptation.

En sus, (Ayed, et al., 2007) propose une catégorisation centrée sur le sujet de l'adaptation : i) Adaptation structurelle (*structural adaptation*) : modifie la structure de l'application (ajout /suppression de méthode, attribut, etc.) ; ii) adaptation architecturale (*architectural adaptation*) : ajout/suppression de composant, objet, service, etc ; iii) adaptation comportementale (*Behavioral adaptation*) : s'intéresse à la modification du comportement des éléments de l'application.

A l'instar de Bucchiarone et al. (Bucchiarone, et al., 2013), Canal et al. (Canal, et al., 2006) distinguent aussi entre : i) adaptation statique (*Static or Design Time Adaptation*): L'ensemble des adaptations possibles sont connues, planifiées avant l'exécution du système ; ii) adaptation dynamique (*Dynamic or Runtime Adaptation*) : le composant à adapter ainsi que l'adaptation sont inconnus avant le moment de l'adaptation. En outre, Canal et al. (Canal, et al., 2006) proposent une autre classification de l'adaptation basée sur la façon avec laquelle l'adaptation est gérée. Ils distinguent entre : i) Adaptation manuelle (*Manual Adaptation*) : la logique d'adaptation est spécifiée et mise en œuvre manuellement par des développeurs ; ii) Adaptation automatique (*Automatic Adaptation*) : la logique d'adaptation est générée automatiquement par des outils logiciels ; iii) Adaptation fonctionnelle (*Functional Adaptation*) : vise à faire des modifications des services offerts par le système ; iv) Adaptation technique (*Technical Adaptation*) : vise à modifier la façon dont les services sont offerts.

II.2.3.4. Mécanismes d'adaptation de service

Au niveau de cette section nous allons illustrer quelques stratégies d'adaptations des applications orientées service.

II.2.3.4.1. Modélisation explicite de la variabilité

Le concept de variabilité trouve ses origines dans le paradigme SPLE (Kazhamiakin, et al., 2010). SPLE concept inspiré de l'industrie traditionnelle[1], représente une autre avancée visant la promotion de la flexibilité, la réutilisation et l'amélioration de la productivité (Bonnet, 2005).

[1] Henry Ford a introduit le concept des lignes d'assemblage dans l'industrie automobile depuis 1908 (wikipedia).

Selon Kazhamiakin et al. (Kazhamiakin, et al., 2010) SPLE se base essentiellement sur la définition et la réalisation des points en commun[1] et des variabilités de la ligne de produits. Les points en commun comprennent les artefacts et les propriétés qui sont partagés par l'ensemble des produits fabriqués via une ligne de produits. La variabilité définit la façon dont les divers produits fabriqués par la ligne de produits peuvent varier. Il est aussi à noter qu'un point en commun peut aussi varier. C'est ce qui est appelé variabilité au sein d'un point en commun (Chang, et al., 2007a).

Nous citons aussi quelques définitions du concept de variabilité dans le contexte de l'ingénierie du logiciel. Selon Sinnema et al. (Sinnema, et al., 2006) la variabilité est la capacité d'un système logiciel ou d'un artefact à être étendu, modifié, personnalisé, ou configuré pour être utilisé dans un contexte spécifique. Weiss (Weiss, et al., 1999) définit la variabilité par la façon dont les membres d'une famille peuvent différer les uns des autres. Aussi (Chang, et al., 2007b) considère que la variabilité est une caractéristique qui peut varier d'une application à l'autre.

D'une façon générale la modélisation explicite de la variabilité d'un composant logiciel permet de définir les éventuels changements de l'application et la façon avec laquelle le système change (quand le changement doit avoir lieu et avec quel variant). Bonnet (Bonnet, 2005) considère que le concept de variabilité correspond à des décisions de conception retardées : « un point de variation dans un système définit un endroit pour lequel le choix entre plusieurs alternatives n'est pas fixé instantanément mais lié à une prise de décision ultérieure ».

Deux familles d'approches de modélisation de la variabilité existent dans la littérature. L'une propose l'intégration des informations de la variabilité dans des modèles existants, comme par exemple sous forme d'une extension d'UML (He, et al., 2015). La deuxième approche propose d'utiliser un modèle de variabilité dédié (Kim, et al., 2005).

Selon les travaux (Kazhamiakin, et al., 2010) (Sun, et al., 2010b) (La, et al., 2009) (Chang, et al., 2007b), un modèle de variabilité doit au moins contenir les informations suivantes :

- Point de variation (variation point) : lieu de la variation ;
- Variant : ensemble d'instances (alternatives) pour chaque point de variation ;
- Contraintes de variabilité : explicite les conditions d'utilisation de chaque variant.

[1] *Commonality* dans le vocabulaire anglo-saxon.

Dans le contexte de notre travail, la question suivante s'impose : comment SOA peut bénéficier des bonnes pratiques de SPLE, concernant la réutilisation et la gestion de la variabilité ? Boffoli (Boffoli, et al., 2009) a essayé de fournir des éléments de réponse à cette question. Leur travail représente l'une des utilisations du concept SPL dans SOA. Ils ont utilisé les concepts SPL pour produire des processus métier sensibles au contexte (Business Process Line). Chaque processus est composé de points en commun et de variabilités. A partir des besoins des utilisateurs une variante du processus est produite. Ce dernier est transformé en un système SOA. En outre, la modélisation de la variabilité est utilisée par de nombreux travaux s'intéressant à l'orienté services, visant la production de services hautement variables. L'étude (Mohabbati, et al., 2013) en est une parfaite illustration. Aussi dans le monde cloud, (La, et al., 2009) utilise la modélisation des points en commun et la variabilité pour produire des services adaptables.

Selon Mohabbati et al. (Mohabbati, et al., 2013), la modélisation de la variabilité dans le monde service diffère de celle de SPLE par différents aspects. En réalité, le monde service est constitué de plusieurs niveau d'abstraction : exigences métier, composition de services, interface de service, implémentation de service, etc. la variabilité peut se manifester dans un ou plusieurs de ces niveaux. En outre, en plus de la prise en charge de la variabilité des services internes il faut aussi tenir compte des variabilités de services externes. Aussi (Sun, et al., 2010b) ajoute que la modélisation de la variabilité des systèmes basés sur les services doit soutenir la flexibilité en cours d'exécution (*run-time*) et pas seulement lors de la conception ou la compilation (*compile-time*).

Dans notre travail nous avons opté pour une modélisation explicite de la variabilité, en produisant un modèle dédié de la variabilité de service. Nous adoptons aussi dans le reste de ce manuscrit la définition de variabilité de (Sinnema, et al., 2006) qui considère que la variabilité est la capacité d'un système logiciel ou d'un artefact à être étendu, modifié, personnalisé, ou configuré pour être utilisé dans un contexte spécifique.

Contrairement à (Northrop, 2008) qui considère que SPLE représente un successeur de SOA, nous considérons que ces deux paradigmes sont complémentaires et peuvent être conjointement utilisés. En effet, une ligne de produits peut produire des artefacts sous forme de service. Cela a été confirmé par plusieurs travaux qui ont adopté à la fois l'orienté services et SPLE (Papazoglou, et al., 2008) (La, et al., 2009) (Mohabbati, et al., 2013) etc.

II.2.3.4.2. Le service multi-vue

Il va sans dire que le concept de vue a fait ses preuves dans le domaine de l'informatique. Sa large adoption par les systèmes de gestion des Worflows, les systèmes de gestion de base de données et les approches orientées objet en est la meilleur preuve (Kenzi, 2010) (Achilleos, 2009). Pour illustrer ce concept nous nous sommes basés sur le travail de Kenzi (Kenzi, 2010). Un service multi-vues fournit/requiert un ensemble d'interfaces de service visuelles (*ViewServiceInterface*). Chaque interface visuelle de service permet de décrire ses capacités fournies/requises selon le profil de l'acteur interagissant avec le service. L'interface de service de base (*baseServiceInterface*) regroupe les interfaces communes. Elle est accessible à tous les acteurs interagissant avec le service.

II.2.3.4.3. Sélection de composition

Le principe de ce concept est intuitif, c'est la composition de services qui est adaptable aux utilisateurs. Swashup DSL (Maximilien, et al., 2007) en est un parfait exemple (voir la section II.3.2.4). Un autre exemple est celui de (Qiu, et al., 2007) où ils proposent une méthode de composition des services Web sensibles au contexte.

II.2.3.4.4. Sélection de service

Le choix du service dépend du contexte d'utilisation. Pour ce faire, en plus de la publication de la description du service, le fournisseur doit publier le contexte auquel le service peut répondre. Ce contexte inclut les conditions d'utilisation du service. Ces derniers dépendent en général des caractéristiques de l'utilisateur et de son environnement. L'exemple de (Suraci, et al., 2007), outre les conditions d'utilisation du service, utilise deux descriptions de service, la première est une description basique (*Basic Service Description*), exprimée dans un langage de bas niveau tel que XML ou WSDL. La seconde est une description sémantique (*Semantic Service Description*) exprimée en OWL-S.

II.2.3.4.5. Différenciation de service

Le travail de Tao et al. (Tao, et al., 2007) illustre parfaitement ce concept. Ils ont décomposé le processus métier en une partie noyau « *Core Business Process*, CBP » regroupant l'ensemble des activités accessibles à tous les utilisateurs, indépendamment des contextes d'utilisations, et une partie variable. Ces deux parties forment ce qu'ils ont appelé *processus métier configuré au contexte* « *Configured Context Business Process* CCBP ». La composition des fonctionnalités du processus s'adapte selon le contexte d'utilisation. Des

fonctionnalités spécifiques sont requises pour chaque ensemble d'utilisateurs. Une interface est associée à chaque CCBP, elle expose les fonctionnalités indispensables pour un groupe d'utilisateurs dans un contexte d'utilisation donné.

D'autres concepts d'adaptation de service existent dans la littérature scientifique spécialisée, entre autres nous citons : la réexécution (service indisponible), la renégociation du contrat de service, la substitution, etc (Bucchiarone, et al., 2010b) (Kazhamiakin, et al., 2010).

II.2.3.5. Synthèse

La définition du concept « contexte » adoptée est une jointure entre la définition de Dey (Dey, 2001) et celle de (Winograd, 2001). En fait, nous avons ajouté la précision de Winograd (Winograd, 2001), qui dicte que les éléments du contexte sont ceux utilisés dans l'adaptation d'une application (non limitée à l'interaction utilisateur application) à la définition de Dey (Dey, 2001) qui trace un espace infini et illimité de ce qui fait partie du contexte (utilisation du terme « toute information »). Cette jointure permet de limiter la généricité de la définition de Dey (Dey, 2001) critiquée aussi par (Boukadi, 2009).

Dans le cadre de nos travaux, et pour plus de séparation des préoccupations, l'un des principes fondamentaux de l'industrie du logiciel (Kazhamiakin, et al., 2010), nous faisons une distinction entre les systèmes adaptables, les systèmes de gestion de contexte et les systèmes d'aide à la décision. Ainsi, nous appelons système (ou service) adaptable tout système (ou service) qui s'adapte aux changements du contexte d'utilisation fourni par un système de collecte et de gestion de contexte et selon une logique d'adaptation. Pour le même souci de forte cohésion, nous adoptons la définition du concept d'adaptation proposée par David (David, 2005) qui distingue entre changement du système, décision de changement et le contexte d'exécution.

En se basant sur les catégorisations de l'adaptation déjà citées, nous pouvons classer l'adaptation traitée par notre approche par *designed* (Bucchiarone, et al., 2013) (ou statique (Canal, et al., 2006)), sensible au contexte (context-aware) (Kazhamiakin, et al., 2010) et manuelle (Canal, et al., 2006). Ayant égard aux succès de l'utilisation de la modélisation de la variabilité par les SPLE, et aux avantages que présente l'utilisation d'un modèle dédié de la variabilité en terme de séparation des préoccupations, et aussi en considérant notre choix d'utiliser le paradigme DSM au niveau des différentes facettes de notre approche, le choix du concept d'adaptation reposant sur la modélisation explicite de la variabilité de service s'impose.

II.2.4. Conclusion

Au niveau de cette section nous avons essentiellement survolé les trois principaux paradigmes utilisés par notre proposition, à savoir SOC, DSM et l'Adaptation.

L'adoption de l'architecture SOA n'est plus à justifier. Elle favorise la réutilisation, réduit la complexité (un faible couplage), facilite la gestion et la flexibilité des processus métiers (alignement IT / processus métier), etc. Nous avons abordé les différentes facettes du paradigme SOC, notamment la vue architecturale, la vue technologique, la vue méthodes de mise en place et la vue composition de services.

De plus, l'approche DSM représente une révolution dans l'ingénierie dirigée par les modèles. En effet, en plus de la modélisation de la solution du domaine en utilisant un langage spécifique, elle propose une seule transition de la solution du problème vers le code source. Les langages spécifiques sont par défaut sémantiquement très fort ce qui rend la génération totale du code source une tâche réaliste. La description de l'approche DSM a porté sur : des définitions et principe de base, l'architecture et l'apport.

Avec l'émergence et le développement des dispositifs mobiles, l'adaptation des systèmes n'est plus considérée comme étant un luxe. Pour illustrer cela, nous citons l'exemple de l'adaptation de l'affichage des interfaces graphiques selon le dispositif utilisé, qui est une caractéristique fortement recommandée. Le concept d'adaptation a aussi été traité dans cette section en fournissant les définitions, les ingrédients, les catégories et les mécanismes d'adaptation.

II.3. Approches de modélisation des SOSs adaptables

Au fil de notre travail de recherche nous avons étudié une panoplie d'approches visant l'encadrement de l'étape de modélisation des SOSs adaptables. Nous présentons dans cette section une description de l'ensemble des approches étudiées. Nous les avons catégorisés en approches à base de profil UML et approches à base de DSLs. Il y a lieu de signaler que nous considérons un DSL comme étant un langage spécifique conforme au MOF, indépendant d'UML (pas sous forme de profil UML) et dont le méta-modèle est positionné au même niveau d'abstraction que le méta-modèle d'UML (Figure 56).

II.3.1. Approches à base de profils UML

Les approches à base de profils UML ont été catégorisées en trois sous catégories : approches basées sur la modélisation du contexte, approches basées sur le profil SOAML (OMG, SoaML, 2012) et approches basées sur la modélisation explicite de la variabilité de service.

II.3.1.1. Approches basées sur la modélisation du contexte

Le concept « contexte » suscite un intérêt grandissant de la part de la communauté de recherche spécialisée dans le domaine de l'interaction homme-machine (Schmidt, et al., 1999). L'objectif de cette section est de présenter les différents travaux combinant la modélisation du contexte et la modélisation du service pour la mise en place de systèmes adaptables. La prise en charge de l'adaptation par les approches étudiées dans cette sous-section est principalement basée sur la modélisation explicite du contexte d'exécution du système orienté services. Quatre travaux sont étudiés : Boukadi (Boukadi, 2009), Monfort et al (Monfort, et al., 2009), Sheng et al. (Sheng, et al., 2005) et Raik et al. (Raik, et al., 2012).

II.3.1.1.1. Descriptions

Boukadi (Boukadi, 2009) a proposé une démarche méthodologique nommée CSOMA (**C**ontextual **S**ervice **O**riented **M**odelling and **A**nalysis) visant l'encadrement de la construction des systèmes orientés service. CSOMA permet de joindre la réutilisation du patrimoine applicatif existant à la pensée SOA naturellement orientée vers le les processus métiers de l'entreprise. Le méta-modèle de CSOMA est représenté au niveau de la Figure 7. Il est composé de trois couches :

- **La couche métier :** s'intéresse spécialement au métier de l'entreprise, précisément aux flux d'information et aux processus métiers ;
- **La couche service :** un service d'entreprise est spécialisé en deux concepts fils : les services métier et les services informatiques.

  Les services métier s'occupent des fonctionnalités métier et ils sont adaptables en fonction du contexte de leur utilisation. A leur tour, les services métier sont spécialisés en services fonctionnels et services domaines. Les services fonctionnels sont des services qui encapsulent une fonctionnalité métier élémentaire. Ils sont caractérisés par une granularité moyenne, un service de calcul du montant total d'un panier en est un exemple illustratif. Les services domaines sont des services de grande granularité. Ils encapsulent des processus métiers d'un domaine. En d'autres termes, les services domaines orchestrent des services fonctionnels. Ces derniers utilisent des services de

bas niveau appelés services informatiques. Ceux-ci regroupent des fonctionnalités existantes fournies par le système d'information existant ;

- **La couche applicative :** représente la totalité des applications élaborées pour implémenter les fonctions de l'entreprise. Cette couche modélise le patrimoine applicatif de l'entreprise.

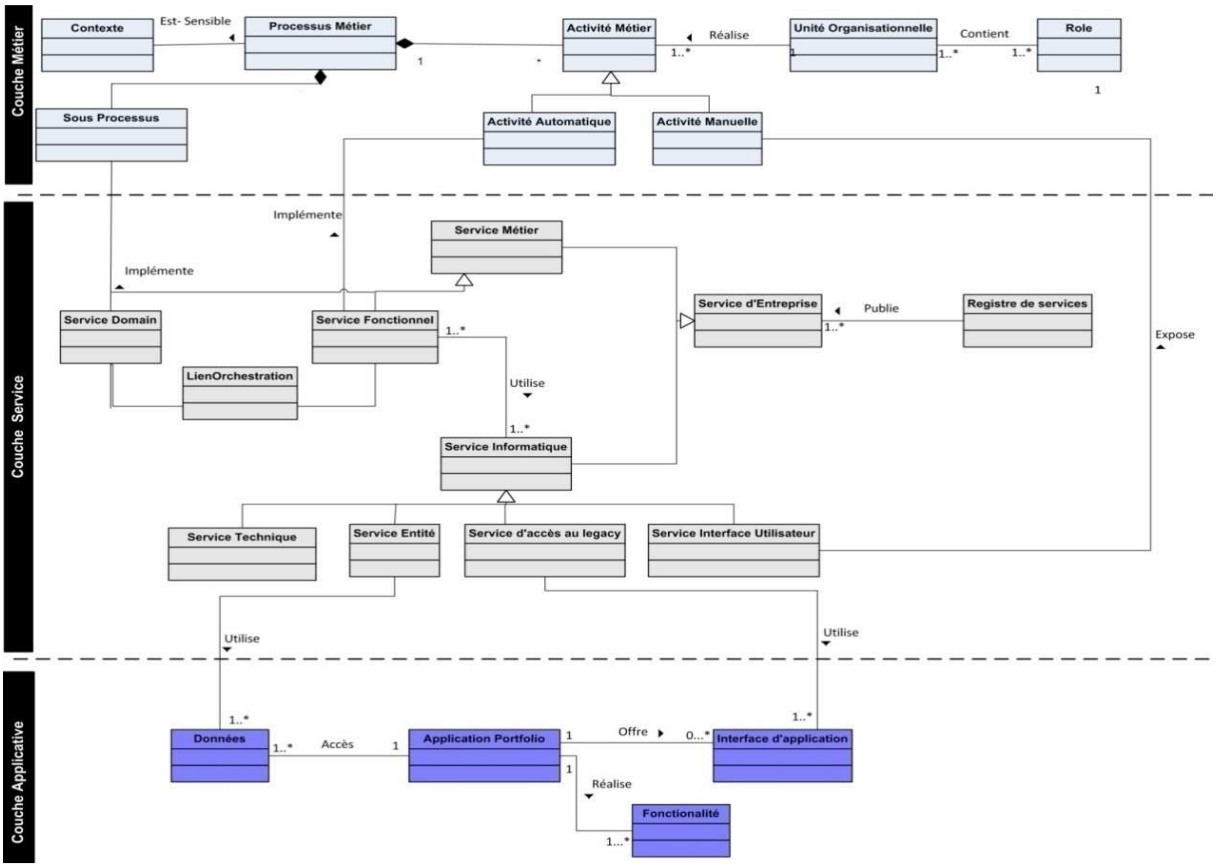


Figure 7 : Méta-modèle de CSOMA (Boukadi, 2009)

Boukadi a aussi défini des profils UML pour les services métiers et les services informatiques (Boukadi, 2009). Le premier décrit les éléments du service métier, à savoir, l'interface, le contrat et le comportement adaptable. Le deuxième catégorise les services informatiques en Services d'accès au système legacy (encapsulant le patrimoine applicatif de l'entreprise), Services d'interfaces utilisateur (permettant de gérer les communications et le dialogue avec les acteurs du système), Services entités (services de création, de consultation, mise à jour et suppression (*CRUD*[1]) des objets métier clés du système d'information de l'entreprise) et les services techniques (permettant de gérer l'infrastructure du système d'information de l'entreprise).

[1] CRUD est une abréviation anglo-saxonne qui signifie: *Create*, *Read*, *Update* and *Delete*.

D'autre part, Boukadi (Boukadi, 2009) a proposé une ontologie modélisant les informations contextuelles utilisées dans un processus coopératif. Elle est divisée entre trois catégories :

- *Service_Related_Context* : s'occupe des conditions des fournisseurs de services domaines qui participent à des processus coopératifs (considération de performance, considération métier et d'autres considérations en relation avec le fournisseur) ;
- *Customer_Related_Context* : représente les informations contextuelles permettant aux fournisseurs d'adapter leurs services. Cette catégorie est divisée en deux sous-catégories, le profil (*Entreprise_Profile*) et les préférences (*Entreprise_Preferences*) ;
- *Cooperation_Related_Context* : modélise les informations en relation avec une opportunité de coopération.

Ces trois dimensions forment ce qu'elle a appelé *Situation-Contextuelle.* La modification de la situation contextuelle déclenche la modification de la logique métier du service domaine (ou processus métier).

Dans la même visée, Monfort et al (Monfort, et al., 2009) ont proposé une approche pour améliorer l'adaptation des services web. Dans ce cadre ils ont défini un méta-modèle pour modéliser le contexte (Figure 8). Ce dernier est composé de six entités contextuelles génériques et quatre entités déduites spécifiques à des applications mobiles. *ContextView* regroupe l'ensemble des entités contextuelles. Elle possède deux types de relations avec l'élément *ContextEntity* : l'agrégation *involves* et l'association *belongsTo*. La première relation exprime que *ContextView* est composé de plusieurs *ContextEntity* qui sont associés à une application sensible au contexte. La deuxième relation *belongsTo* exprime l'utilisation des informations historiques du contexte. En effet, un élément donné du contexte peut avoir participé à différents *ContextViews* et peut aussi être utilisé dans la conception des futurs *ContextViews*. La deuxième entité générique est *ContextEntity*. Elle est spécialisée en trois éléments génériques : *Actor*, *ComputationalEntity* et *Environnement*. L'Acteur utilise des dispositifs de calcul (*ComputationalEntity*, Généralement un appareil mobile) pour accéder aux services et aussi pour capturer des informations contextuelles de l'environnement. Un environnement comprend différentes catégories d'informations comme : (i) des informations du contexte spatial (l'emplacement, ville, bâtiment, etc.) ; (ii) des informations du contexte temporel (la saison, la date, l'heure, etc.) ; (iii) Le climat (température, météo, etc.). La dernière entité est *Profile*, elle permet de décrire un acteur. Ce dernier peut avoir un profil statique et/ou un profil dynamique. Le profil statique recueille des informations valables

quel que soit le contexte d'utilisation, notamment : le nom, le sexe, la date de naissance. Par contre, le profil dynamique comprend des informations personnalisées en fonction du type d'application spécifique et/ou de l'acteur, entre autres : les objectifs, les préférences, les intentions, les désirs, les contraintes, etc.

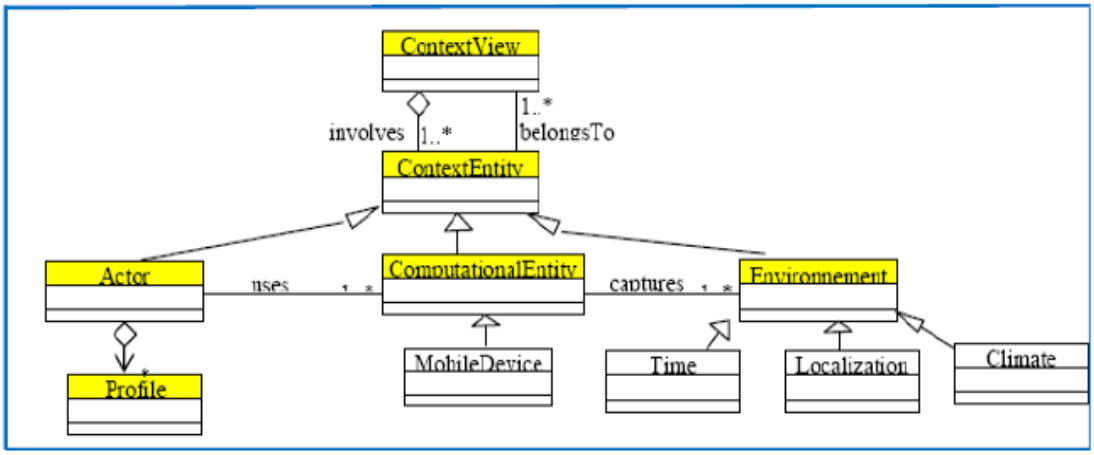


Figure 8 : Méta-modèle du contexte (Monfort, et al., 2009)

En outre, Sheng et al. (Sheng, et al., 2005) proposent un méta-modèle baptisé ContextUML, pour la modélisation des services Web sensibles au contexte. Comme illustré sur la Figure 9, ce méta-modèle est composé de plusieurs classes qui permettent de créer des services Web sensibles au contexte. La classe *Context* permet de décrire un contexte observable. La modélisation de la source d'un *AtomicContext* est prise en charge par le modelé. Ce dernier intègre aussi la notion de communauté de services pour le choix d'un collecteur approprié. Le méta-modèle ContextUML permet aussi la description des actions d'adaptation (*Action*) et les situations pertinentes (*ContextConstraint*) qui permettent de les déclencher à travers le mécanisme de *ContextTriggering*. Le mécanisme *ContextBinding* permet de définir les relations entre les objets sensibles au contexte et leurs paramètres contextuels.

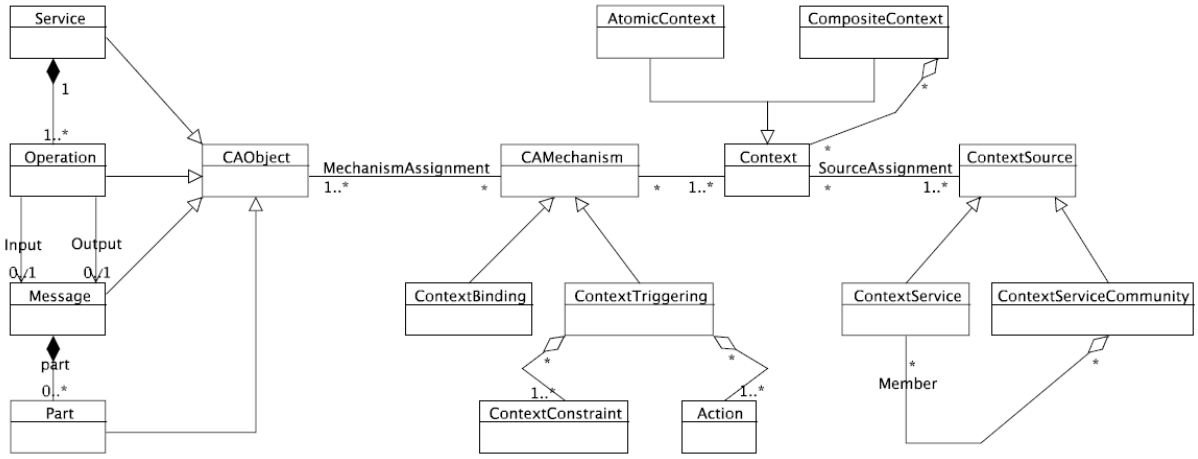


Figure 9 : Méta-modèle ContextUML (Sheng, et al., 2005)

Le mécanisme de sensibilité au contexte est assigné aux objets sensibles au contexte (*CAObject*) via la relation *MechanismAssignment. CAObject* est spécialisé en quatre sous-types : *Service, Operation*, *Message* et *Part*. Il est à noter que ces spécialisations sont adoptées à partir des spécifications WSDL.

Par ailleurs, Raik et al. (Raik, et al., 2012) ont proposé un framework pour définir et soutenir les processus métiers basés sur les services sensibles au contexte. Ce qui nous a intéressés dans ce travail est la partie modélisation du contexte. Raik et al. ont modélisé le contexte d'exécution sous forme de diagrammes d'état transition. Le scénario utilisé pour valider leur approche est basé sur le fonctionnement du port de Brème qui assure la liaison fabricants-détaillants de voitures. Initialement, une voiture est sur le navire, ensuite déchargée vers la zone de déballage, puis déplacée vers l'une des zones de stockage. L'emplacement de traitement mécanique est l'endroit où la voiture peut être réparée. De même, le diagramme *CarStatus* représente l'opérabilité d'une voiture. Le contexte est défini par un ensemble de propriétés (*context properties*), chacune décrivant un aspect particulier du domaine de l'application (par exemple l'emplacement actuel d'un véhicule, l'état d'un véhicule, la disponibilité de l'espace de stockage, etc (Figure 10)).

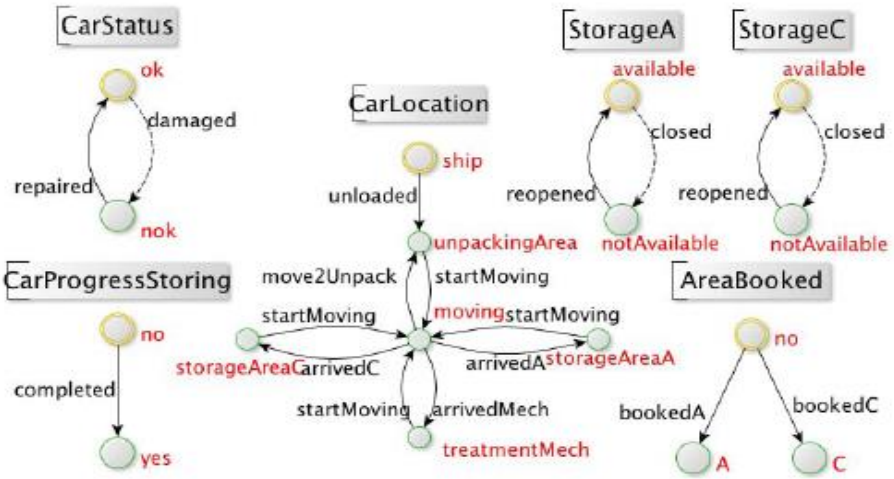


Figure 10 : Diagrammes de contexte du scénario « logistique de voiture » (Raik, et al., 2012)

Une propriété de contexte peut évoluer avec l'exécution du service, ce qui correspond au comportement normal du domaine (par exemple l'emplacement actuel de la voiture passe de « zone de déballage » à « zone de stockage A »).

Une configuration du contexte est une capture, à un instant donnée, des états actuels de l'ensemble des propriétés du contexte.

II.3.1.1.2. Discussion

Il est évident que les systèmes d'information existaient avant l'apparition du paradigme SOC. Dans ce sens l'un des points forts de l'approche CSOMA (Boukadi, 2009) réside dans l'utilisation de l'existant IT de l'entreprise (notion de service informatique). En effet, le fait d'exploiter le patrimoine IT existant permet de réduire le coût de l'adoption de l'architecture orientée services par les systèmes d'information existants. En outre, la séparation du méta-modèle de CSOMA en couches permet d'améliorer sa lisibilité. En revanche, l'auteur n'a pas explicité la relation entre le processus métier (service domaine) et le contexte. Nous avons aussi constaté qu'au niveau de l'ontologie proposée, prenant en charge la modélisation du contexte, les relations entre les éléments de ce dernier n'ont pas été considérées, de plus, le modèle proposé est un modèle spécifique à la coopération des entreprises en utilisant des services.

Certes le modèle proposé par Monfort et al (Monfort, et al., 2009) est un modèle extensible, par ailleurs aucune relation entre le contexte et les services adaptables n'a été proposée.

En ce qui concerne le profil UML ContextUML (Sheng, et al., 2005), il ne respecte pas le principe de séparation des préoccupations. En réalité, le même modèle est utilisé pour modéliser le contexte, les services et les règles d'adaptation. En outre, le méta-modèle ContextUML incorpore des détails d'implémentation spécifiques à la technologie service web.

L'utilisation du diagramme d'état transition pour modéliser le contexte (Raik, et al., 2012) permet de lier le contexte (état) et l'action (transition) appropriée. Pour ce faire, les valeurs des éléments contextuels doivent être connues au préalable. Chaque élément du contexte est représenté par un diagramme état-transition. Par contre, cette approche ne prend pas en charge la modélisation des relations entre les éléments du contexte.

II.3.1.2. Approches basées sur le profil SoaML

Les approches étudiées dans cette sous-section se basent sur le profil UML SoaML « Service oriented architecture Modeling Language » proposé par l'OMG (OMG, SoaML, 2012). Ce profil offre aux utilisateurs du langage UML, un sous langage facilitant la modélisation d'une architecture Orientée services. En plus du profil SoaML, deux travaux sont décrits dans cette sous-section : la proposition de Hafiddi et al. (Hafiddi, 2012) et la proposition d'Abu-matar et al. (Abu-Matar, et al., 2011a).

II.3.1.2.1. Descriptions

La version actuelle de SoaML 1.0.1 (version 2012) (OMG, SoaML, 2012) est une extension d'UML 2.1. La Figure 11 illustre sa partie services :

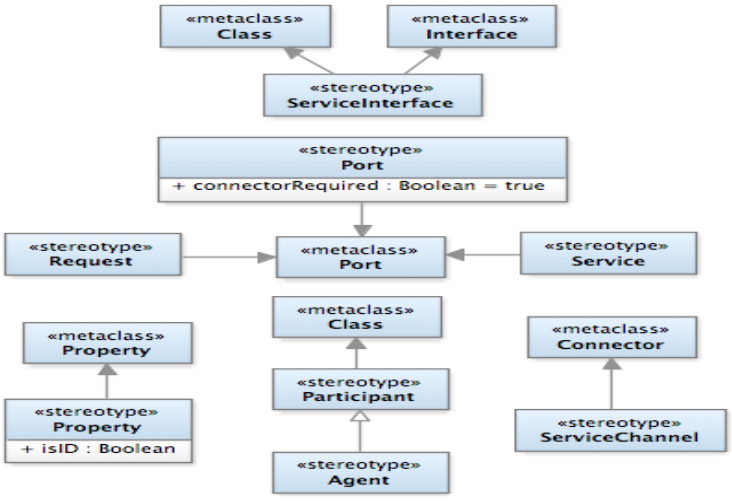


Figure 11 : Partie « Services » du profil SoaML 1.0.1 (OMG, SoaML, 2012)

Plusieurs stéréotypes ont été définis par le profil SoaML. Le Tableau 1 en décrit quelques-uns.

Tableau 1 : Stéréotypes du profil SoaML 1.0.1

Stéréotype	Métaclasse UML	Description
Agent	Class via Participant	Représente des Entités autonomes qui s'adaptent et interagissent avec leur environnement.
Participant	Class	Les participants sont en général les fournisseurs et les consommateurs des services
Request	Port	« désigne un port qui définit un point de connexion à travers lequel un participant répond à ses besoins via la consommation des services fournis par d'autres fournisseurs » (Kenzi, 2010)
Service	Port	Un *Service* est un port qui représente le point de connexion par lequel un fournisseur offre ses capacités à ses clients.
Service Interface	Class	*ServiceInterface* permet de définir un service. Il définit la spécification de l'interaction *Service* ou *Request*
ServiceChannel	Connector	*ServiceChannel* représente un chemin de communication entre les requêtes et les services

Plusieurs travaux de modélisation des systèmes orientés service adaptables se sont basés sur ce profil. Par exemple Hafiddi (Hafiddi, 2012) a proposé une architecture nommée ACAS « Architecture for Context-Awareness of Services » permettant de faciliter le développement des systèmes CASOS « Context-Aware Service Oriented Systems ». La modélisation des services (couche service de l'architecture ACAS) a été réalisée en utilisant une extension de SoaML (version 2009) (OMG, SoaML, 2009). En effet, il a intégré les concepts *CAServiceInterface*, *CAServicePoint* et *CARequestPoint* au niveau du méta-modèle SoaML (Figure 12).

Un participant possède la capacité de fournir et/ou consommer des services sensibles au contexte à travers ses ports *CAServicePoint* et *CARequestPoint*. L'interface d'un service sensible au contexte *CAServiceInterface* représente le mécanisme de typage (i.e., utilisée comme protocole) des ports *CAServicePoint* et *CARequestPoint* des participants.

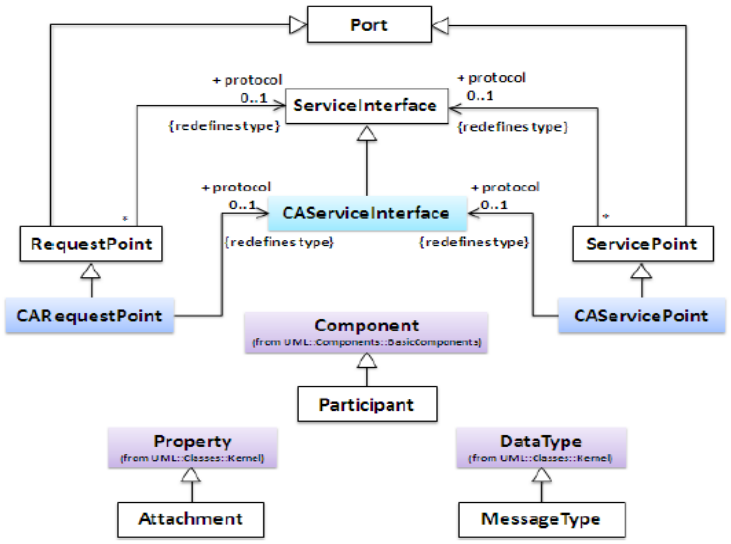


Figure 12 : Fragment du méta-modèle CASosML **(Hafiddi, 2012)**

Aussi, Hafiddi et al. (Hafiddi, 2012) ont proposés un méta-modèle des services sensibles au contexte (couche services sensibles au contexte de l'architecture ACAS) (Figure 13). Un service sensible au contexte (*ContextAwareService*) est un service spécifique qui possède des *ContextViews* à travers un *CASAdaptationStrategy*. Une stratégie d'adaptation (*CVSAdaptationStrategy*) indique quand (*AdaptationCondition*) et comment (*AdaptationRule*) un ensemble d'adaptations (*Adaptation*) s'applique sur le service de base afin d'adapter son comportement au contexte d'exécution. Le résultat de l'adaptation est représenté par l'élément *ContextViewService*. Ainsi, pour un service donné, l'ensemble de ses *ContextViewServices* (respectivement *CVSAdaptationStrategies*) constituent le service sensible au contexte (respectivement *CASAdaptationStrategy*). A l'exception de l'élément *ContextAwareService* qui hérite de la méta-classe *Operation*, tous les autres éléments héritent de la méta-classe *Class*.

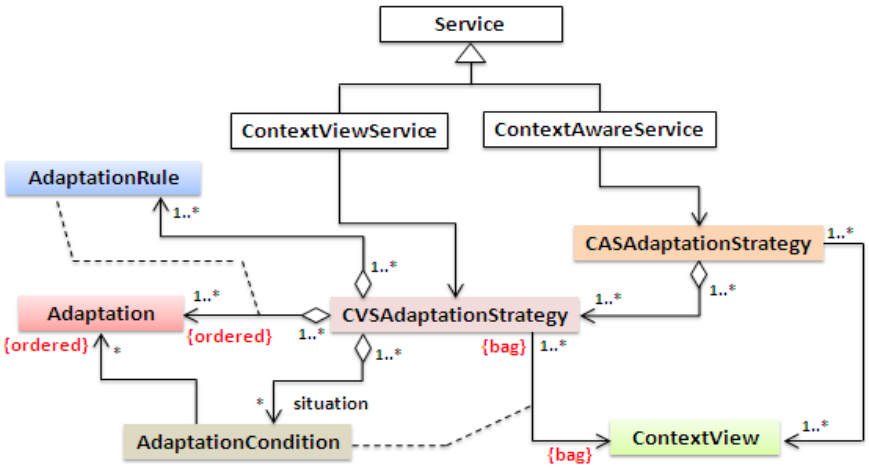


Figure 13 : Méta-modèle de base des services sensibles au contexte (Hafiddi, 2012)

De plus, Hafiddi et al. (Hafiddi, 2012) ont proposé un méta-modèle de contexte. Ce méta-modèle (Figure 14) considère que le contexte est un élément qui peut être décomposé en sous contextes (*SubContext*). De même Un sous contexte peut être organisé, récursivement, en catégories (*Category*). Les éléments *Parameter* (exemple : localisation, préférences, profil, etc.) et *Entity* (exemple : utilisateur, dispositif, etc) représentent les principaux composants d'un contexte/sous-contexte.

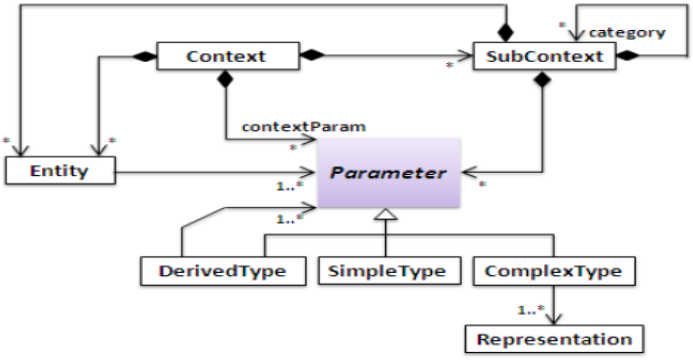


Figure 14 : Méta-modèle du contexte (Hafiddi, 2012)

Trois types de paramètres du contexte existent : complexe (*ComplexParam*), dérivé (*DerivedParam*) et simple (*SimpleParam*). Comme leur nom l'indique les paramètres dérivés sont obtenus en se basant sur d'autres paramètres. Les paramètres complexes disposent de représentations spécifiques (élément *Representation*). Finalement, une entité est décrite par un ou plusieurs paramètres. La liaison entre le contexte et un service sensible au contexte est réalisée via la relation d'agrégation entre *ContextView* est *Parameters.*

D'autre part Abu-matar et al. (Abu-Matar, et al., 2011a) (Abu-Matar, et al., 2011b) ont exploité les techniques de modélisation de la variabilité utilisées au niveau des lignes de produits logiciels pour modéliser la variabilité dans une architecture Orientée services. En effet, ils ont utilisé leur méta-modèle spécifique à la modélisation des « caractéristiques » (*features*) (Abu-Matar, et al., 2011a) pour intégrer la notion de variabilité au profil UML SoaML (OMG, SoaML, 2009). Une caractéristique est une exigence réutilisable qui peut être sélectionnée par n'importe quel membre d'une famille de services. Les caractéristiques *(features)* connexes peuvent être groupées en groupe de caractéristiques (*features group*) qui limitent la façon avec laquelle sont utilisées par un produit ou une ligne de produits.

Le méta-modèle de la variabilité de service est représenté dans la Figure 15. Les méta-classes SoaML utilisées sont : *ServiceContract*, *ServiceInterface*, et *Participant*. Afin de représenter les points en commun et les variabilités (*variability*) toutes les méta-classes sont

spécialisées en *kernel*, *optional*, et *variant*. Les caractéristiques *(Features)* sont spécialisées en *kernel*, *optional*, *alternative*, et *default*. *kernel* pour une caractéristique requise pour l'ensemble des produits. *Optional* pour une caractéristique requise par quelques produits. *Alternative* est une alternative des caractéristiques *Kernel* ou *Optional.* Elle est utilisée pour répondre à des exigences spécifiques de certains produits. *Default* pour une caractéristique choisie par défaut quand il fait partie d'un « *feature group* ». « *feature dependencies* » représente les relations entre les caractéristiques.

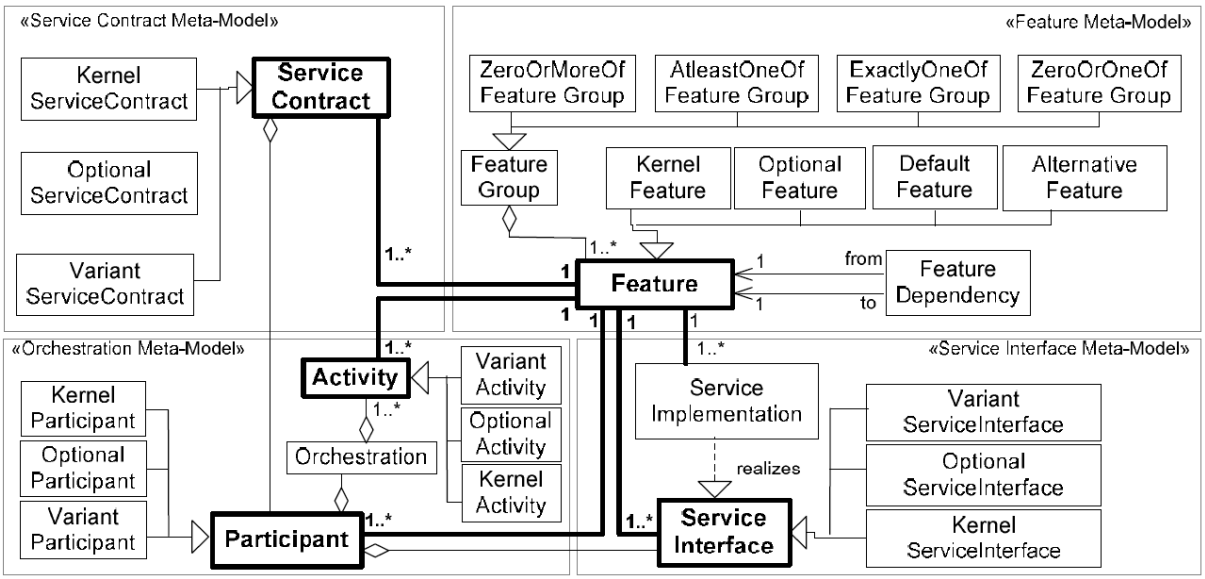


Figure 15 : Méta modèle de la variabilité de service (Abu-Matar, et al., 2011b)

II.3.1.2.2. Discussion

En ce qui concerne le travail de Hafiddi et al. (Hafiddi, 2012), sachant qu'un *ContextAwareService* est constitué de *ContextViewServices,* le faite de mettre ces deux éléments au même niveau héritant de l'élément *Service* prête à confusion. En plus, les auteurs n'ont pas explicité la relation entre l'extension proposée de SoaML et le méta-modèle de service sensible au contexte. Pour la partie contexte, force est de constater que le méta-modèle proposé est un modèle générique et extensible (indépendant des plateformes technologiques et des domaines d'application). En outre, il modélise même les relations entre les feuilles du modèle (via l'élément *DerivedType).* Le modèle proposé distingue entre les paramètres du contexte et les paramètres des entités (Entity) du contexte. En revanche nous remarquons qu'aucun attribut des éléments modélisés n'est proposé.

Abu-matar et al. (Abu-Matar, et al., 2011b) (Abu-Matar, et al., 2011a) ont défini le concept de variabilité de service orientée caractéristique (Feature Oriented Service

Variability). Ce dernier vise l'indentification des différences entre les produits sous forme de caractéristiques (*features*). Il permet aussi de les organiser sous forme d'un modèle de caractéristiques (*feature model*). Aucune transformation du modèle proposé vers le code source n'a été opérée, en plus aucun outil encadrant la mise en place de l'approche n'a été développé.

II.3.1.3. Approches basées sur la modélisation de la variabilité de service

En vue de soutenir le dynamisme et la flexibilité des systèmes orientés services, beaucoup de chercheurs ont commencé à explorer l'utilisation de la modélisation de la variabilité dans la conception de ce type de systèmes (Chang, et al., 2007b). Dans ce sens Clotet et al. (Clotet Martínez, et al., 2008) considèrent que l'utilisation simultanée de la modélisation de la variabilité et de l'orienté services est considérée comme une solution prometteuse pour atteindre à la fois la flexibilité et l'adaptabilité. Cette sous-section illustre cinq travaux basés la modélisation de la variabilité pour soutenir l'adaptation des SOSAs : Chang et al. (Chang, et al., 2007b), Narendra et al. (Narendra, et al., 2010), Clotet et al. (Clotet Martínez, et al., 2008), Parra et al. (Parra, et al., 2012) (Parra, et al., 2009), et He Xiao et al. (He, et al., 2015).

II.3.1.3.1. Descriptions

Nous allons commencer par le travail de Chang et al. (Chang, et al., 2007a) (Chang, et al., 2007b) (Kim, et al., 2005) qui ont proposé deux méta-modèles. Le premier pour modéliser les applications orientées services et le deuxième pour modéliser la variabilité (Figure 17). Ce dernier est utilisé comme une base pratique pour définir les types de variabilité de service.

La Figure 16 illustre le méta-modèle de service. L'entité *ServiceProvided* représente un service offert par un fournisseur de service. Elle remplit les exigences dictées par un *ServiceRequested.* Un *ServiceProvided,* qui est en générale d'une granularité importante, possède une composition de services appelée *ServiceComposition.* Ce dernier est composé de services plus fins, appelés *UnitService.*

Un *UnitService* est relié à une interface modélisée via l'élément *ServiceInterface*. Cette dernière spécifie son implémentation qui est sous forme d'un composant de service (*ServiceComponent).* Un *ServiceComponent* peut être implémenté en utilisant différentes technologies tel que EJB, COM, ou tout simplement en adaptant les composants existants.

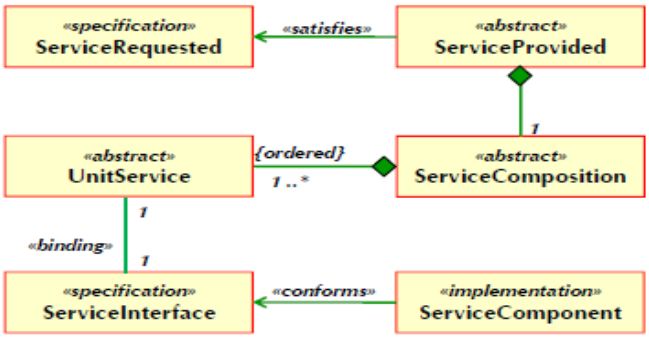


Figure 16 : Méta-modèle de service (Chang, et al., 2007b)

Comme déjà décrit au niveau de la section II.2.3.4.1, un point de variation (*Variation Point*) est un emplacement dans un composant logiciel où un changement peut survenir. Pour un contexte donné, un variant (alternative) est associé à un point de variation.

Pour Chang et al. (Chang, et al., 2007a) un point de variation possède un type (*Type*) et une portée (*Scope*). Cette dernière est spécialisée en trois catégories : i) *Binary* : deux variantes identifiées au préalable sont possibles pour un point de variation ; ii) *Selection* : plus que deux variantes identifiées au préalable sont possibles ; iii) *Open* : plusieurs variantes identifiées au préalable sont possibles en plus de variantes supplémentaires inconnues.

Cinq types de point de variation sont modélisés :

- *Workflow* : variation de la séquence d'exécution des services unitaires ;
- *Logic* : variation de l'implémentation (algorithme) d'un service unitaire ;
- *Interface* : variation de la signature des méthodes de l'interface de service (nombre et type des paramètres d'entrée / sortie, nom des méthodes, etc.) ;
- *Attribute* : la variation concerne le nombre, le type, la valeur (cas des constantes), etc, des attributs ;
- *Persistency* : variation du schéma physique de stockage, de la représentation des attributs persistants au niveau de l'espace de stockage, etc.

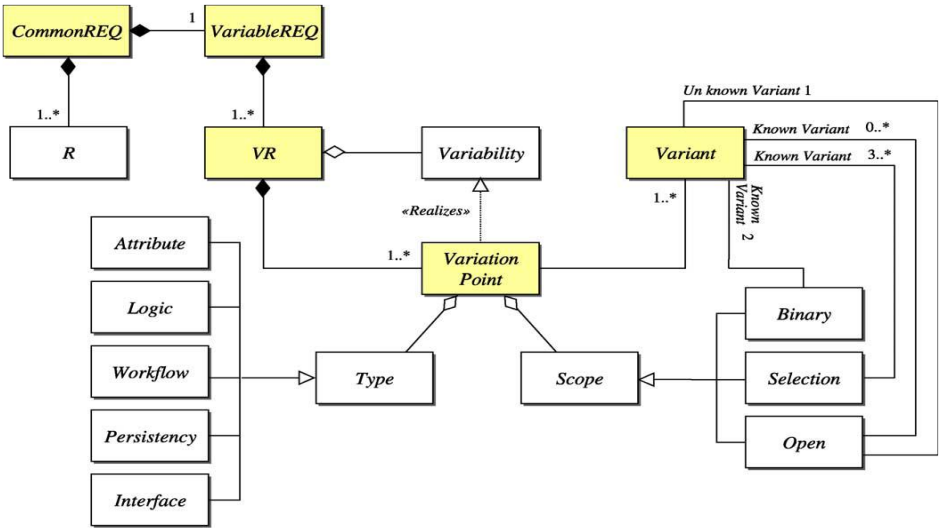


Figure 17 : Méta-modèle de variabilité (Kim, et al., 2005)

CommonREQ est l'ensemble des exigences logicielles communes entre les membres d'une famille (au moins deux membres).

VariableREQ est un sous ensemble de *CommonREQ*. Chaque fonction atomique *VR* appartenant à *VariableREQ* a une différence mineure avec des membres de la famille.

Dans le même sens, Narendra et al. (Narendra, et al., 2010) ont défini une approche d'ingénierie orientée variation [1]. Elle est composée de trois étapes : *variation-oriented analysis*, *variation oriented design* et *variation oriented implementation*. Dans le cadre de la première étape Nanjangud et al. (Narendra, et al., 2010) ont proposé un méta-modèle qui permet de modéliser séparément les parties statiques et variables de tout composant logiciel (service ou processus). Comme présenté dans la Figure 18 le méta-modèle proposé comprend les parties suivantes :

- « *Variation point* » : deux types de points de variation proposés, i) point de variation d'implémentation où seules les implémentations de certaines méthodes qui peuvent être modifiées sans affecter le comportement externe du composant logiciel ; ii) point de variation de spécification : affectent les interfaces des composants comme par exemple l'ajout ou la suppression de paramètres d'entrée/sortie ;
- « *Variation features* » : permettent d'affiner les points de variation en spécifiant la sémantique des actions de variation. Ils distinguent entre variation d'un service et variation d'un processus.

[1] *Variation Oriented Engineering* dans le vocabulaire anglo-saxon.

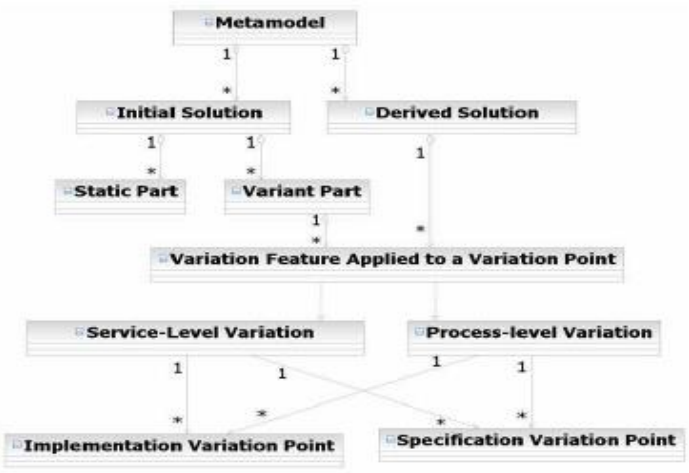


Figure 18 : Méta-modèle de variation (Narendra, et al., 2010)

Ce méta-modèle est à spécialiser pour instancier des modèles conceptuels modélisant la variation des services et des processus métiers.

Par ailleurs, Clotet et al. (Clotet Martínez, et al., 2008) ont utilisé le méta-modèle de variabilité de Dhungana et al. (Dhungana, et al., 2007) en l'adaptant aux systèmes orientés service (Figure 19). L'élément *Asset* décrit les composants du système et leurs relations. L'élément *Decision* décrit la variabilité du système à travers un ensemble de variables de décision qui sont utilisées pour adapter le système. La relation « *included if* » permet de déterminer les *Assets* qui vont former le système en fonction des valeurs des variables de décision. Clotet a proposé quatre types d'*Assets* : « *service goal* », « *service type* », « *service* » *and* « *service interface* ».

« *Service goal* » définit l'objectif d'un service (par exemple, "Offrez des voyages"). Différents « *Services types* » participent à l'accomplissement de cet objectif (par exemple, « un fournisseur de services de voyages»). Les services disponibles réalisant un « *Service type* » sont modélisés sous forme de *Services*. Enfin, les implémentations des services sont modélisées sous forme de « *Service instances* ».

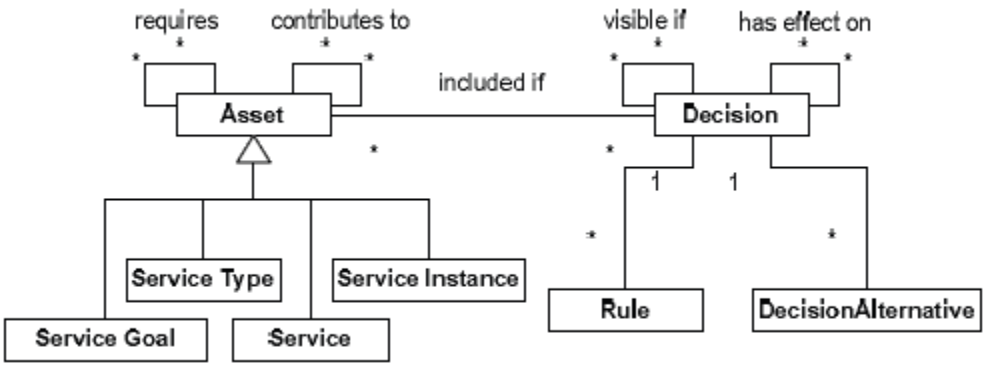


Figure 19 : Méta-modèle de gestion de la variabilité (Clotet Martínez, et al., 2008)

Le modèle de décision est utilisé pour modéliser la variabilité du système et pour décrire les dépendances entre les points de variation.

D'autre part, Parra et al. (Parra, et al., 2012) (Parra, et al., 2009) qui font partie de l'équipe Adam[1], proposent une ligne de produits dynamique, Orientée services et sensible au contexte nommée « CAPucine » (Context-Aware Dynamic Service-Oriented Product Line (DSOPL)). Leur travaux visent la construction des systèmes orientés services et sensibles au contexte en adoptant le paradigme SPLE. Leur approche se base essentiellement sur l'intégration dynamique des « *Assets*[2]» appropriés au niveau du système en cours d'exécution. La ligne de produits « CAPucine » est basée sur deux différents processus de dérivation de produit :

- Le premier utilise une composition des *Assets* pour obtenir le modèle d'application intégré, ensuite elle procède à la transformation du modèle pour l'enrichir par les concepts de la plateforme cible et le langage d'implémentation. Finalement, le produit est généré via une transformation modèle au code source.

  La plateforme cible adopte l'approche orientée services. Elle est basée sur une autre plateforme orientée SCA (*FraSCAti*) permettant la liaison des composants à l'exécution. Un méta-modèle a été proposé, il permet de décrire à partir d'un haut niveau d'abstraction, la structure et le comportement d'une application SCA (Figure 20). Les méta-classes *Element*, *Service*, et *Reference* permettent de décrire la structure d'une application orientée services. Ces méta-classes sont utilisées pour construire les composants SCA et leurs implémentations.

  Le comportement est représenté par deux méta-classes: *Activity* et *Connection*. Une activité (*Activity*) est essentiellement une liste de liens représentant un processus métier ;
- Le deuxième processus concerne l'adaptation dynamique. Ce processus introduit la notion d'Assets sensibles au contexte[3] qui opèrent à l'exécution selon l'état du contexte. Les *Assets* sensibles au contexte sont composés de trois types d'informations : le moment où les *assets* peuvent être modifiés (état du contexte), l'endroit où les *assets* doivent être appliqués et le changement qui doit être effectué (Figure 21).

[1] L'équipe ADAM : http://adam.lille.inria.fr/pmwiki.php/Adam/HomePage.
[2] Représentent les artefacts et les ressources réutilisables qui constituent la base du produit logiciel.
[3] *Context-Aware Assets* dans le vocabulaire anglo-saxon.

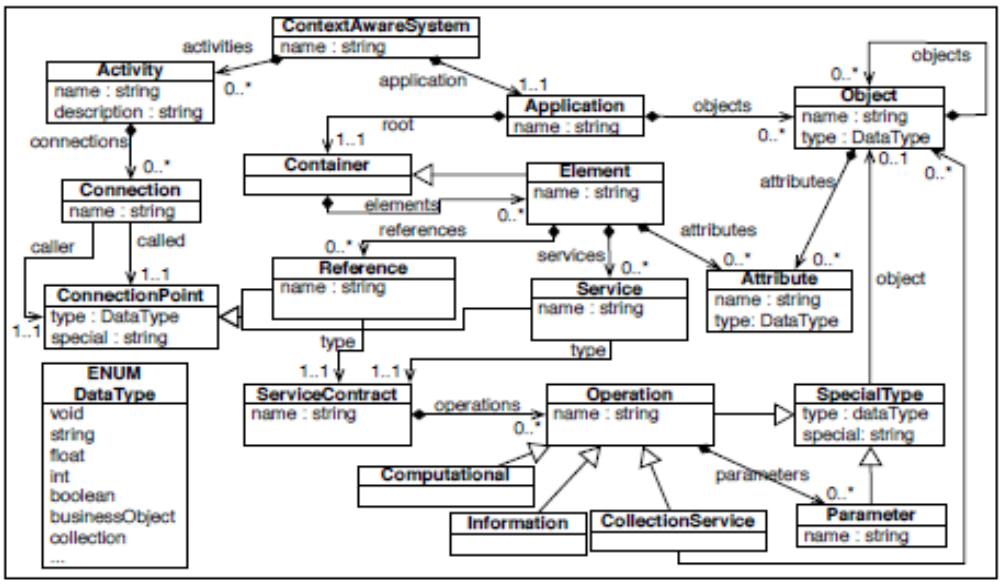


Figure 20 : Méta-modèle d'une application SCA (Parra, et al., 2009)

Les Assets sensibles au contexte différent des Assets classiques. En effet, la décision de leur intégration au système est prise dynamiquement (à l'exécution) et conduite par le changement du contexte.

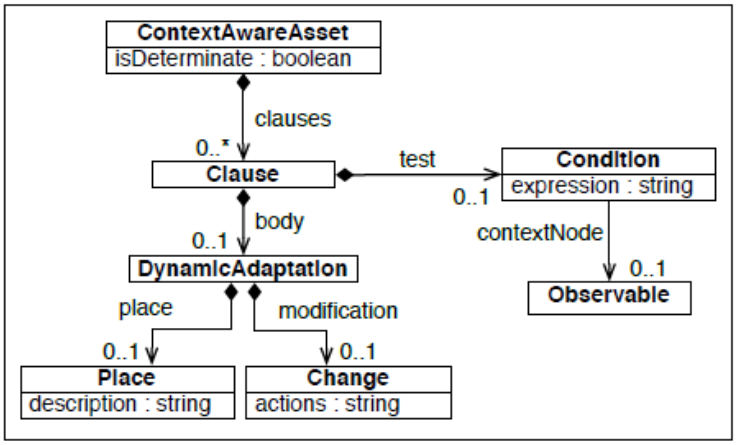


Figure 21 : Méta-modèle des Assets sensibles au contexte (Parra, et al., 2009)

La dernière approche étudiée dans cette sous-section est celle de He Xiao et al. (He, et al., 2015). Ils ont proposé une approche de composition flexible de services, basée sur la variabilité et dirigée par les modèles. La variabilité a été modélisée dans deux niveaux d'abstraction différents : au niveau du modèle de l'architecture en utilisant le profil UML VxUML (Figure 22), et au niveau de l'implémentation en utilisant une extension du standard BPEL (OASIS, BPEL4WS, 2007) baptisée VxBPEL (He, et al., 2015). Des règles de transformation permettent de convertir les modèles VxUML en VxBPEL. Les concepts de variabilités adoptés par le profil VxUML sont ceux du méta-modèle COVAMOF (Sun, et al., 2010a) (Sun, et al., 2010b). Ce dernier est un framework permettant de faciliter la

modélisation de la variabilité des systèmes. Les concepts « *variation point* » et « *variant* » feront l'objet d'extension des diagrammes de classe, d'activité, de séquence et de déploiement. En plus des deux concepts « *variation point* » et « *variant* », d'autres concepts sont modélisés via un nouveau diagramme appelé « Diagramme d'interaction des points de variation ». Ce dernier permet de modéliser les relations entre les points de variation.

Une partie du méta-modèle VxUML est représentée dans la Figure 22 (extension du diagramme d'activités).

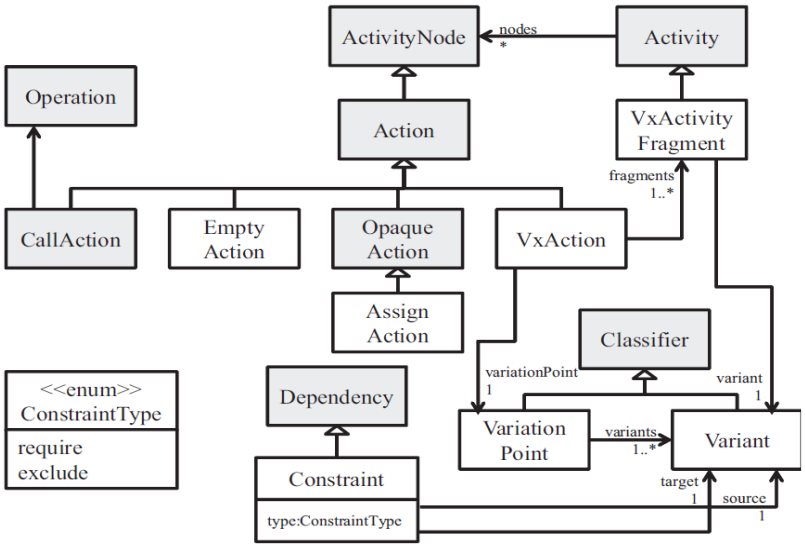


Figure 22 : Méta-modèle VxUML (Extension du diagramme d'activités UML) (He, et al., 2015)

Les éléments dont la couleur est blanche représentent les éléments proposés par le profil UML. VxUML supporte trois nouveaux types d'actions à savoir *AssignAction*, *EmptyAction* et *VxAction*. La première s'occupe de la conversion de données, la deuxième représente une tâche vide, la troisième symbolise les points de variation des processus métiers. Les alternatives (variantes) d'un *VxAction* sont définies en utilisant les éléments « *VxActivity Fragments* ».

II.3.1.3.2. Discussion

Le modèle de service proposé par Chang et al. (Chang, et al., 2007b) (Figure 16) est un modèle simpliste qui ne comporte aucune représentation du contexte d'exécution des services, élément important dans la modélisation des services adaptables. En revanche, le point le plus important dans le travail de Chang et al. est la catégorisation des variabilités de service représentée dans la Figure 17 (Kim, et al., 2005).

Narendra et al. (Narendra, et al., 2010) ont essayé de fournir une description globale de l'ingénierie orientée variation (*Variation Oriented Engineering*) des systèmes orientés service. L'un des points importants de l'approche de Narendra et al. réside dans la modélisation des composants statiques, ce qui permet de favoriser la réutilisation des composants déjà implémentés. Ce travail qui est largement basé sur le travail de Chang et al (Chang, et al., 2007a) (Chang, et al., 2007b) (Kim, et al., 2005) souffre aussi de l'absence de modèle de contexte. La même remarque est soulevée pour le travail de Clotet et al. (Clotet Martínez, et al., 2008) et He Xiao et al. (He, et al., 2015). Finalement, la proposition de Parra et al. (Parra, et al., 2012) (Parra, et al., 2009) ne traite que l'adaptation architecturale. Les autres formes d'adaptation ne font pas partie de leurs champs de recherche.

II.3.1.4. Approches à base de vues

Cette section traite les approches de modélisation des SOSAs basées sur le concept de vue. Ce dernier a été illustré par le travail de Kenzi (Kenzi, 2010).

II.3.1.4.1. Descriptions

Kenzi a proposé un profil UML 2.0 nommé VSoaML « *View based Service Oriented Architecture Modeling Language* » pour la modélisation des systèmes orientés services adaptables (Kenzi, 2010). Ce profil est bâti essentiellement en se basant sur le concept de Vue (voir la section II.2.3.4.2). La Figure 23 illustre le méta-modèle du profil VSoaML.

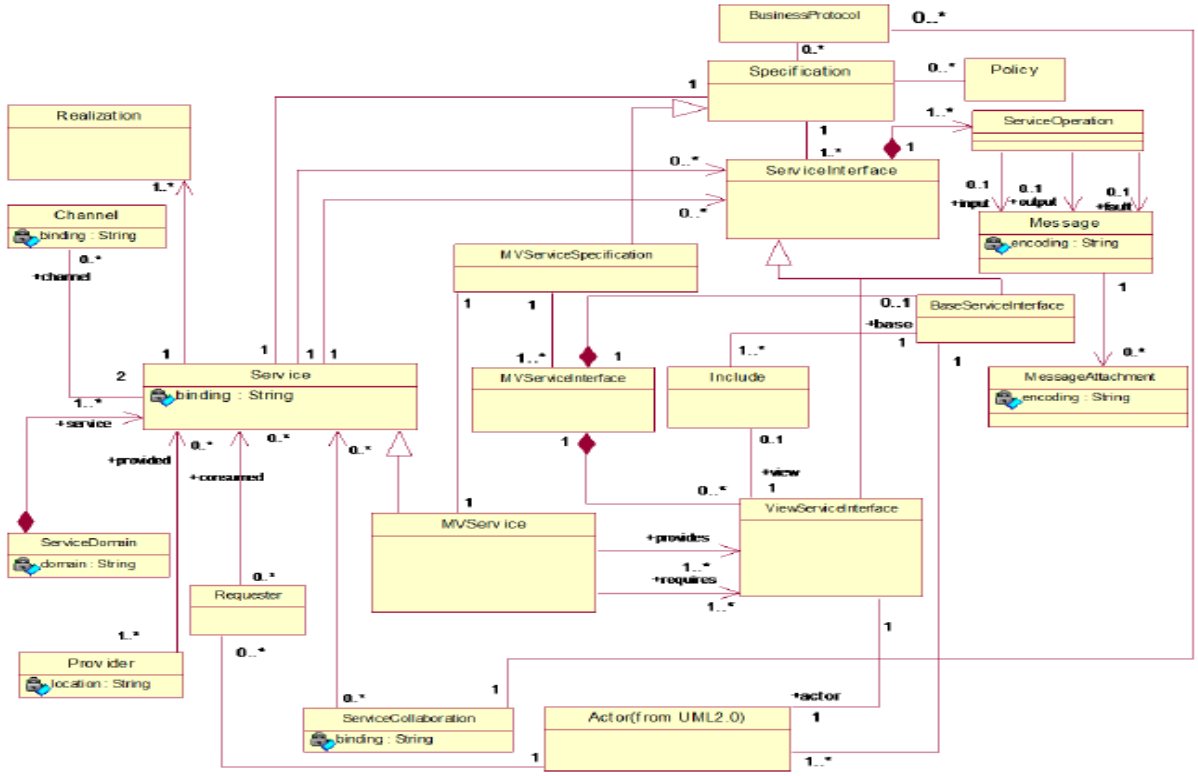


Figure 23 : Méta-modèle VSoaML (Kenzi, 2010)

Le Tableau 2 illustre quelques stéréotypes de ce profil ainsi que leurs méta-classes UML2.0 de base.

Tableau 2 : Stéréotypes du profil VSoaML

Stéréotype (VSoaML)	Metaclasse (UML)	Description
Service	**Port**	représente un point de communication.
MVService	**Port**	permet de représenter un service adaptable aux clients.
BaseServiceInterface	**Interface**	« permet de représenter un ensemble d'opérations accessibles à tous les clients d'un service » (Kenzi, 2010)
ViewServiceInterface	**Interface**	permet de représenter des interfaces propres à un client.
MVServiceInterface	**Package**	« permet de regrouper des fonctionnalités logiquement cohésives. Il est composé d'une interface de service de base et d'un ensemble d'interfaces de service vues » (Kenzi, 2010)

II.3.1.4.2. Discussion

Le concept multi-vues a fait ses preuves au niveau de plusieurs disciplines informatiques (voir la section II.2.3.4.2). Nous reprochons au travail de Kenzi le fait qu'il n'a proposé aucune liaison entre les services multi-vues et leur contexte d'utilisation, et ce afin d'expliciter l'adaptation du système. Encore plus, aucun modèle de représentation du contexte d'utilisation des services adaptables n'est proposé. En outre, l'orchestration de services représentée par l'élément *ServiceCollaboration* présente un manque concernant la spécification de l'ordre d'exécution des services, la correspondance des paramètres en entrée et en sortie des services qui composent la collaboration.

II.3.2. Approches à base de DSLs

L'utilisation de l'approche DSM par les systèmes orientés services attire de plus en plus l'attention de la communauté de recherche spécialisée dans l'ingénierie de ce type de systèmes. En effet, nous allons présenter dans cette section, une panoplie de DSLs s'intéressants à différents aspects de l'architecture orientée services, à savoir, l'adaptation et la modélisation de services ((Achilleos, 2009) (Ortiz, 2012), ArchiMeDeS (López-Sanz, et al., 2012), D-SOA (Parigot, et al., 2008)), la sécurité ((Saleem, et al., 2012), SOLJ (Bharadwaj, et al., 2007)), la qualité de service (QUALA (Oberortner, 2011) (Oberortner, et al., 2008) (Oberortner, et al., 2009)) et l'orchestration de services (VCL (Rosenberg, et al., 2009a) (Rosenberg, et al., 2009b), Swashup DSL (Maximilien, et al., 2007)).

II.3.2.1. Domaine : adaptation et modélisation de services

L'un des travaux les plus intéressants qui utilisent l'approche DSM pour produire des SOSAs est celui d'Achilleos (Achilleos, 2009). Achilleos et al. ont proposé un framework adoptant le paradigme développement dirigé par les modèles et permettant de créer des services sensibles au contexte (Achilleos, 2009). Pour ce faire, Achilleos a conçu trois DSLs : le premier pour la modélisation du contexte (Figure 24), le deuxième pour la modélisation de la présentation graphique, le troisième pour modéliser le comportement dynamique des services sensible au contexte. Ce dernier est sous forme d'un modèle de processus basé sur le formalisme des diagrammes de réseaux de Petri[1].

[1] « Les réseaux de Petri (Petri Net) sont apparus en 1962, dans la thèse de doctorat de Carl Adam Petri. Les réseaux de Petri sont des outils graphiques et mathématiques permettant de modéliser et de vérifier le comportement dynamique des systèmes à événements discrets comme les systèmes manufacturiers, les systèmes de télécommunications, les réseaux de transport » (wikipedia).

La Figure 24 illustre le méta-modèle du langage de modélisation du contexte proposé par Achilleas. La méta-classe *DocumentRoot* représente la racine du méta-modèle et le conteneur des éléments du modèle de contexte. La méta-classe *Entity* est une représentation abstraite du monde réel. Elle peut être une personne, un dispositif, etc. Une *Entity* peut être associée à d'autres *Entities* via la méta-classe *Relation*. En outre, une *Entity* peut avoir plusieurs *ContextSituation*. La règle de raisonnement est définie sous forme d'une contrainte OCL en utilisant la propriété *expression*. La variable *attribute* est mise à jour par vrai ou faux après l'évaluation de la contrainte *expression.* Les *ContextSituations* représentent les conditions qui conduiront l'exécution des comportements explicites des services sensibles au contexte si un événement de contexte survient.

En plus des relations entre entités, chaque *Entity* est associée à travers la méta-classe *ContextAssociation* à une variété d'informations contextuelles. *ContextAssociation* est la classe mère des méta-classes abstraites *Profiled*, *Sensed*, *Derived* et *Static*. Les valeurs des propriétés *multiplicity*, *multiplicityType*, *persistence* et *permission* sont énumérées en utilisant des énumérations. La propriété *expression* (exprimée en utilisant OCL) de la méta-classe « Derived » est utilisée pour définir la dépendance des autres informations du contexte.

Un modèle de contexte, est défini comme une instance de la méta-classe *Context* et peut être *Atomic* ou *Composite*. Chaque contexte atomique représente un objet d'information qui se caractérise par la propriété *nom* et contient des propriétés de base (*conproperties*) définies par la méta-classe *Property*.

En outre, la validité du contexte peut être déterminée en utilisant des contraintes temporelles représentées par la méta-classe *Constraint*.

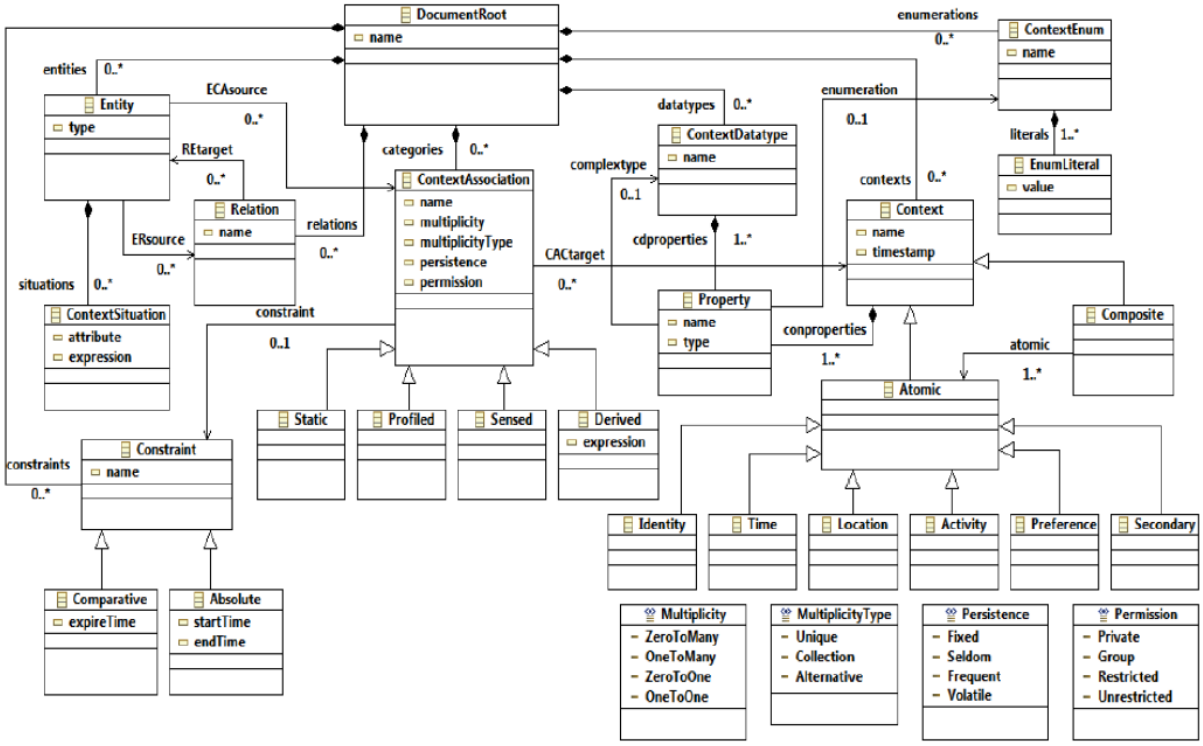


Figure 24 : Méta-modèle du langage de modélisation du contexte (Achilleos, 2009)

La Figure 25 présente un exemple de modèle de service sensible au contexte. Initialement les places p1 et p2 sont marqués avec des jetons d'initialisation "[]". Par conséquent, les transitions t1 et t2 sont activées[1] permettant de créer deux instances des objets *Display* et *Person*. Ces instances d'objets sont stockés sous forme de jetons dans les places p3 et p4, représentant l'état actuel de l'exécution du service. Dès que les places p3 et p4 détiennent les instances d'objet d et p l'arc t3 est activé permettant d'héberger le couple [D, P] au niveau de la place p5. Ce dernier dénote l'état final de l'exécution du service. En d'autres termes, le modèle du processus permet l'intégration des autres modèles.

[1] « Une transition est franchissable lorsque toutes les places qui lui sont en amont contiennent au moins un jeton. Le franchissement consiste à retirer un jeton de chacune des places d'entrée et à rajouter un jeton à chacune des places de sortie de la même transition » (wikipedia).

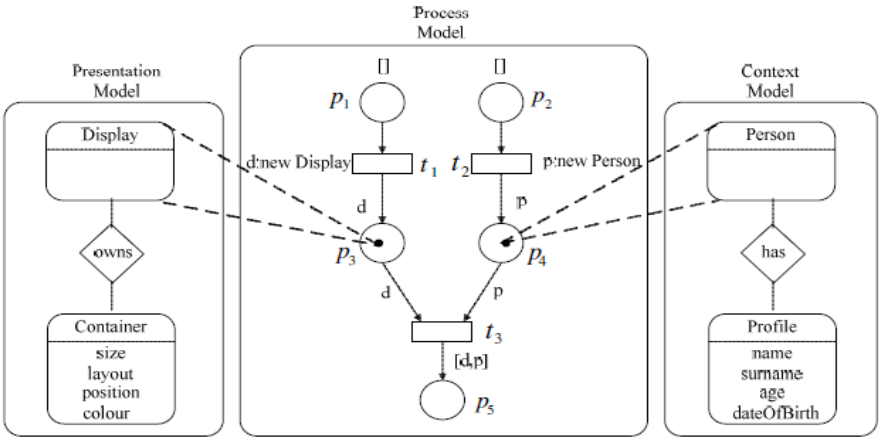


Figure 25 : Intégration des modèles d'un service sensible au contexte (Achilleos, 2009)

D'autre part, le framework baptisé ArchiMeDeS proposé par Lopez-Sanz et al. (López-Sanz, et al., 2012) vise la facilitation de la spécification des architectures orientées service. Les auteurs du framework adoptent l'architecture MDA. Ils ont initialement proposé un profil UML (López-Sanz, et al., 2007), au niveau PIM, faisant abstraction des considérations et contraintes techniques des plateformes. Par la suite, au niveau de leur travail (López-Sanz, et al., 2012) ils ont remplacé leur profil UML par un DSL, toujours pour spécifier le modèle PIM. La syntaxe abstraite de ce dernier est sous forme d'un modèle héritant du MOF (Figure 26) et la syntaxe concrète est une notation graphique organisée dans un profil UML (López-Sanz, et al., 2007).

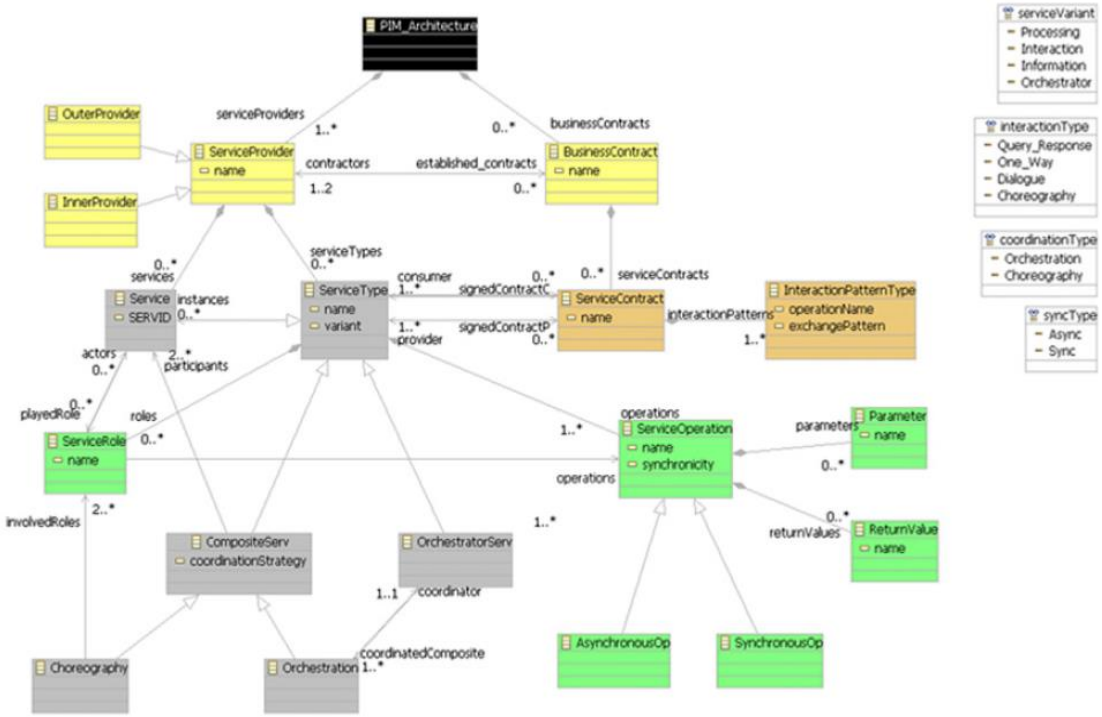


Figure 26 : Méta-modèle du DSL associé au PIM (López-Sanz, et al., 2012)

Le Tableau 3 Décrit les principaux éléments de la syntaxe concrète du DSL.

Tableau 3 : Principaux éléments de la syntaxe concrète du DSL du framework ArchiMeDeS (López-Sanz, et al., 2012)

Elément	Description
OuterProvider	Fournisseur des services externes au système
InnerProvider	Fournisseur des services internes au système
BusinessContract	Contrat entre fournisseurs de services
CoordinationType	Mode de composition
CompositeServ	Service résultant d'une composition de services
OrchestratorServ	Orchestrateur d'une composition de services
ServiceContract	Connecteur de services
ServiceOperation	Fonctionnalité atomique d'un service
ServiceRole	Rôle joué par un service à un moment donné

Quatre types de services sont proposés : i) *Information* : service d'accès aux données ; ii) *Interaction* : service communiquant avec l'extérieur du système ; iii) *Processing* : service effectuant des traitements et des calculs ; iv) *Orchestration* : service orchestrant d'autres services.

Par ailleurs, le concept de SOA dynamique (D-SOA) (Parigot, et al., 2008) se propose de fournir des moyens pour rendre l'architecture adaptable dynamiquement, en cours d'exécution, aux besoins des utilisateurs de l'application.

D-SOA est composée de deux étapes de génération basées sur un générateur de composants (Component Generator CG) et un gestionnaire de composants (Component Manager, CM).

En termes de développement dirigé par des modèles, la démarche s'appuie sur deux DSLs. Le premier DSL intitulé Component Description Meta Language (CDML), s'occupe de la description des services d'un composant. Le générateur de composants (CG) utilise un programme CDML pour produire le code non-fonctionnel des composants. Il indique les services requis (*output*) ou fournis (*input*) par un composant.

La syntaxe concrète du langage CDML est sous forme d'un langage de balisage. La racine d'un composant est représentée par la balise *component*. Cette dernière comporte les services requis (balise *input*) et les services offerts (balise *output*). La balise *facadeclass* permet de spécifier la classe façade générée.

Le deuxième DSL intitulé *World*, définit l'architecture de l'application en identifiant les connexions entre les instances de composants (liaison « *connectTo* »). La topologie d'une application est déterminée par les instances des composants utilisés ainsi que les liaisons les reliant. Au démarrage d'une application, le CM est lancé, ensuite il instancie la topologie de l'application en se basant sur un fichier *world*.
La syntaxe concrète du DSL *World* est aussi sous forme d'un langage de balisage. La racine d'un fichier world est la balise *world*. La balise *connectTo* permet de connecter deux composants en utilisant leurs identifiants. En termes d'implémentation, D-SOA s'exécute au-dessus de la plateforme OSGi[1].

En outre, Wada et al. *(Wada, et al., 2005a) (Wada, et al., 2005b) (Wada, et al., 2005c)* proposent un framework nommé *mTurnpike* qui permet aux développeurs de modéliser et programmer les concepts d'un domaine via un langage spécifique et de les transformer en code source final. *mTurnpike* fournit une abstraction permettant de représenter simultanément les concepts d'un domaine au niveau conceptuel (DSM : les modèles du domaine spécifique) et au niveau programmation (DSC : code spécifique au domaine en utilisant la programmation orientée attributs).

mTurnpike permet aux développeurs et aux concepteurs de travailler dans un niveau d'abstraction élevé. Les programmeurs implémentent les comportements dynamiques (code des méthodes) au niveau du DSC, avant que le transformateur DSC le transforme à un programme spécifique à une technologie cible. Ainsi, DSC (code annoté) est plus lisible et facile à maintenir en comparaison avec un code généré par une approche standard dirigée par les modèles (Wada, et al., 2005c).

Les auteurs de *mTurnpike* ont illustré leur framework par un DSL. Ce dernier nommé SOA DSL (Wada, et al., 2005c) (Wada, et al., 2005b), décrit les concepts spécifiques d'une architecture orientée services (Figure 27). Le DSL proposé se focalise essentiellement sur la modélisation de la connectivité entre les services. Cette dernière est représentée en utilisant l'élément *Connector*. Un *Connector* peut avoir des *Filters* pour personnaliser son comportement.

[1] *Open Services Gateway initiative* (OSGi) est un ensemble de spécifications qui ont pour but de définir une plateforme pouvant être gérée dynamiquement avec un modèle orienté composants.

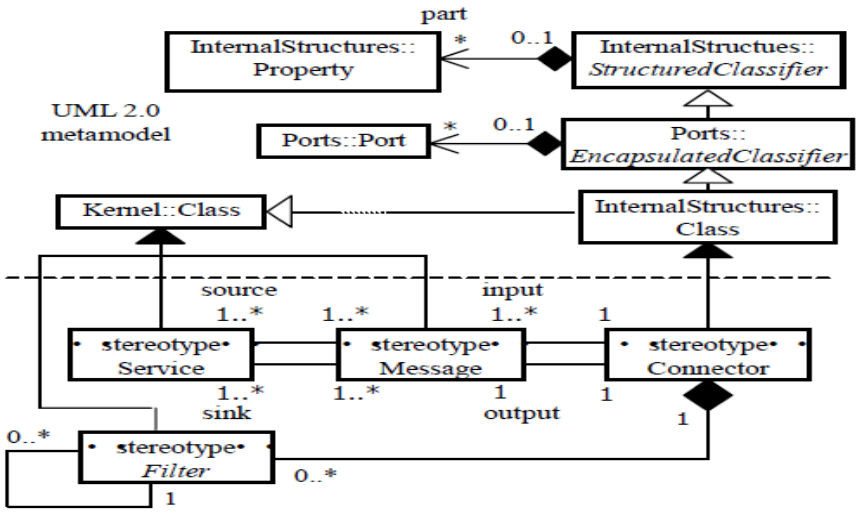


Figure 27 : Méta-modèle SOA DSL (Wada, et al., 2005c)

La syntaxe concrète de SOA-DSL est une syntaxe graphique composée des éléments : *Message, Service, Connector* avec une spécialisation de l'élément *Filter*.

Il est à signaler que malgré que l'étude porte sur les DSLs qui ne sont pas sous forme de profils UML, nous avons jugé intéressant de mettre l'accent sur ce framework puisque leurs auteurs projettent d'augmenter le niveau d'abstraction de SOA DSL au MOF.

II.3.2.2. Domaine : Sécurité

Saleem et al. (Saleem, et al., 2012) ont proposé un DSL pour modéliser les exigences de la sécurité au niveau du modèle des processus métiers. Le DSL a facilité la modélisation des exigences de la sécurité, par les experts métier, au niveau des diagrammes des processus métiers.

La syntaxe abstraite du DSL, représentée au niveau de la Figure 28, est définie en étendant le méta-modèle BPMN (OASIS, BPEL4WS, 2007).

La syntaxe concrète est définie en utilisant les stéréotypes ci-après, elle est sous forme d'une notation graphique :

- <<Confidentiality>> : permet de limiter l'accès à l'information qu'aux utilisateurs autorisés (nœuds concernés : *Pool*, *Lane*, *Activity* et *Group* du diagramme du processus métier) ;
- <<Integrity>> : permet de vérifier que l'information envoyée est bien celle reçue (nœud concerné : « *Message Flow* ») ;

- <<Availability>> : permet d'appliquer le principe de la non répudiation [1] (nœud concerné « *Message Flow* »).

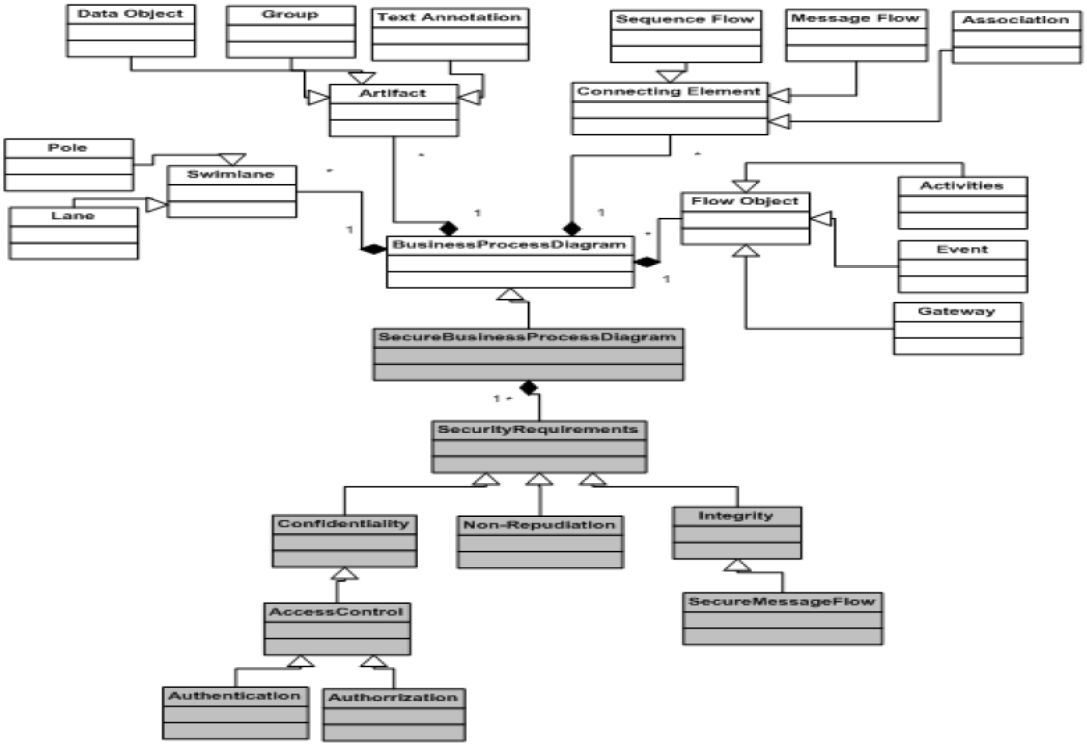


Figure 28 : Méta-modèle BPMN enrichi par des éléments de sécurité (Saleem, et al., 2012)

D'un autre côté, le DSL SOLJ (Secure Operations Language-Java) est un langage spécifique dirigé par les événements (Bharadwaj, et al., 2007). Il est sous forme d'une extension de java spécifique au développement des systèmes sécurisés, temps réel à base de services.

Les programmes SOLJ sont déployés, configurés et exécutés sur la plateforme SINS (Secure Infrastructure for Networked Systems) (Bharadwaj, et al., 2007).

La syntaxe concrète du langage proposé est basée sur celle du langage java. Tous les commentaires java qui commencent par « @ » seront interprétés par le compilateur SOLj, c'est-à-dire transformés en java. Ci-dessous une description des principales sections d'un programme SOLj :

- La section *type definitions* (*//@type definitions*) : comporte les types définis par l'utilisateur ainsi que les types énumérés, et ce en se basant sur les type prédéfinis de java ;

[1] La non-répudiation signifie la possibilité de vérifier que l'envoyeur et le destinataire sont bien les parties qui disent avoir respectivement envoyé ou reçu le message (wikipedia).

- La section *variable declarations* (*//@variable declarations*) : permet de déclarer les variables environnementales à surveiller (*//@ monitored variables*) ainsi que les variables internes (*//@internal variables et //@controlled variables*) ;
- La section *service declarations* (*//@Services*) : permet de déclarer les méthodes à invoquer ainsi que les services qui les fournissent. Cette section permet aussi de définir les *preconditions* à vérifier avant leurs invocations et les *postconditions* sur les valeurs de retour ;
- La section *assumptions* (*//@Assumptions*) : comprend les hypothèses sur lesquelles dépend le fonctionnement correct de l'application. L'exécution sera avortée si l'une de ces hypothèses est violée par l'environnement. Les propriétés de sécurité requises par l'application sont spécifiées dans la section *« guarantees »*.

II.3.2.3. Domaine : Qualité de service

Oberortner et al. (Oberortner, 2011) (Oberortner, et al., 2009) (Oberortner, et al., 2008) proposent une approche DSL dirigée par les modèles qui sépare les DSLs en sous langages par niveaux d'abstraction (Figure 29). Chaque sous langage est taillé pour un intervenant donné, par exemple ceux qui interviennent au niveau technique et ceux qui s'occupent de la conception et des contrats de service (SLA[1]). Les sous-langages sont reliés en utilisant le mécanisme d'héritage.

Cette approche a été validée par un DSL baptisé *Quality of Service Language* (QUALA). En se référant a un contrat de service, ce dernier vise la supervision du respect de la qualité de service et le déclenchement des actions à exécuter dans le cas d'une infraction. Le DSL est divisé en deux langages différents. Le premier est spécifique aux experts non techniques (haut niveau) et le deuxième est utilisé par les experts techniques (bas niveau).

Les experts non techniques spécifient, pour chaque service, les métriques de qualité de service à mesurer. En plus, ils s'occupent de la modélisation des SLAs et des actions à exécuter dans le cas d'une infraction. Le deuxième langage des experts techniques s'occupe de la façon avec laquelle les différentes métriques de qualité de service doivent être mesurées et comment l'action doit être exécutée sur une plateforme technologique particulière (Oberortner, et al., 2009). La Figure 29 illustre le modèle QoS de haut niveau :

[1] *Service Level Agreement (SLA)* dans le vocabulaire anglo-saxon.

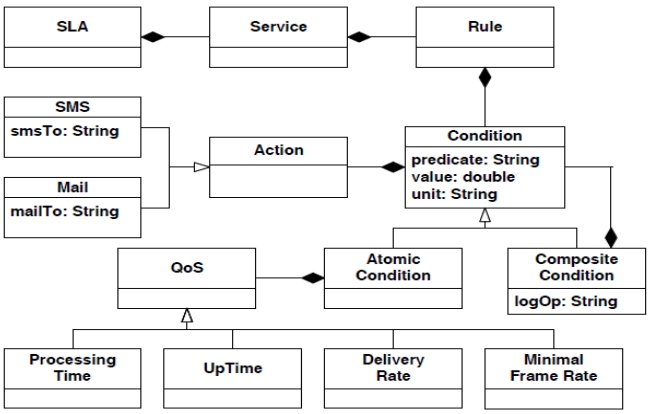


Figure 29 : Modèle de haut niveau du langage QUALA (Oberortner, 2011)

Une SLA est constitué de Services connectés à des règles (*Rule*). Une règle est composée de plusieurs conditions. L'élément *Action* représente les actions à exécuter dans le cas d'une violation d'une SLA. Les conditions atomiques sont reliées aux propriétés de qualité de service.

Les auteurs présentent le langage concret du modèle de haut niveau en utilisant EBNF[1] (Extended Backus-Naur Form) (Figure 30). La syntaxe concrète est une syntaxe textuelle. L'élément de base *sla-specification* est composé du nom du SLA suivi d'un bloc de services et leurs contraintes QoS.

```
sla-specification = sla-name '{' [service-name '{' [qos-constraint]* '}']* '}'
qos-constraint = rule '=>' action [',']?
rule = condition [logical-operator ['(']? condition [')']?]+
condition = qos-property operator predicate
qos-property = 'UpTime' | 'ProcessingTime' |
               'DeliveryRate' | 'MinimalFrameRate'
operator = '<' | '<=' | '>' | '>='
predicate = number unit
unit = '%' | 'd' | 'h' | 'm' | 's' | 'fps'
logical-operator = 'AND' | 'OR'
action = mail-action | sms-action
mail-action = 'mailto' '"' mail-address '"'
sms-action = 'smsto' '"' phone-number '"'
```

Figure 30 : Syntaxe concrète du langage de haut niveau QUALA (Oberortner, 2011)

Le modèle de bas niveau (Figure 31) dépend de la technologie sous-jacente hébergeant les services web. La technologie utilisée pour le modèle ci-dessous est Apache CXF (apache.org, 2016).

[1] Permet de faire une description formelle et mathématique d'un langage.

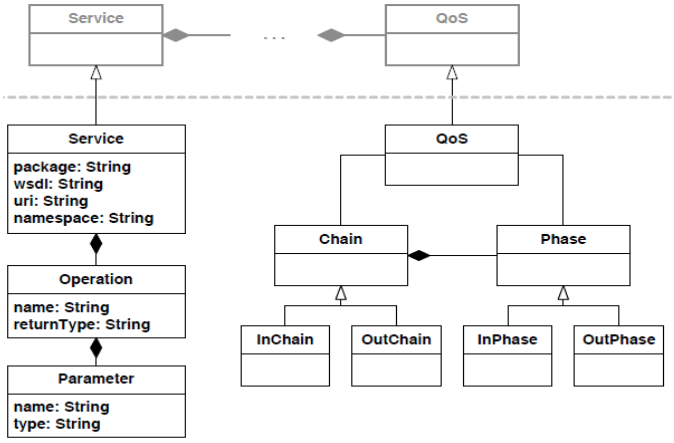


Figure 31 : Modèle de bas niveau du langage QUALA (Oberortner, 2011)

II.3.2.4. Domaine : orchestration de services

L'orchestration des services est un domaine très actif dans le monde service (He, et al., 2015) (Rosenberg, et al., 2009b) (Baidouri, et al., 2012) (Sun, et al., 2010a). Dans ce sens Swashup DSL (Maximilien, et al., 2007) se propose comme étant un modèle de programmation qui facilite et accélère la création et le déploiement des combinaisons de services hétérogènes (REST, SOAP, etc.). Selon Maximilien et al. (Maximilien, et al., 2007) une combinaison (*mashup*) est une application Web qui agrège plusieurs services pour atteindre un nouvel objectif. Sur le plan conceptuel, les combinaisons sont des applications utilisées pour réorienter les ressources web existantes.

Une combinaison (*mashup*) comprend trois principaux volets : i) La médiation de données : consiste à convertir, transformer et combiner une donnée d'un ou de plusieurs services pour fournir les inputs d'un autre service ; ii) La médiation de processus : est essentiellement une chorégraphie de services qui a comme point de mire la création de nouveau processus ; iii) La personnalisation de l'interface utilisateur : cible la récupération des informations des utilisateurs, ainsi que l'affichage des informations intermittentes et définitives du processus à utiliser.

Afin de fournir un modèle uniforme pour la construction et le partage des combinaisons de services, Maximilien et al. (Maximilien, et al., 2007) ont créé la plateforme Swashup. Cette dernière étend l'architecture RoR[1] (Ruby on Rails) avec un nouveau DSL, bibliothèques de support, ainsi que des modèles et services associés à la plateforme. La Figure 32 illustre les composants de haut niveau de l'architecture swashup.

[1] Est un framework web libre écrit en Ruby. Il suit le motif de conception modèle-vue-contrôleur aussi nommé MVC. En tant que framework, il propose une structure au programmeur qui lui permet de développer plus vite et plus intuitivement (wikipedia).

En utilisant l'outil d'interface utilisateur web Swashup (*Swashup web tool*), un utilisateur final crée, édite et déploie un projet Swashup qui contient les informations nécessaires pour décrire les services de la combinaison ainsi que la combinaison elle-même.

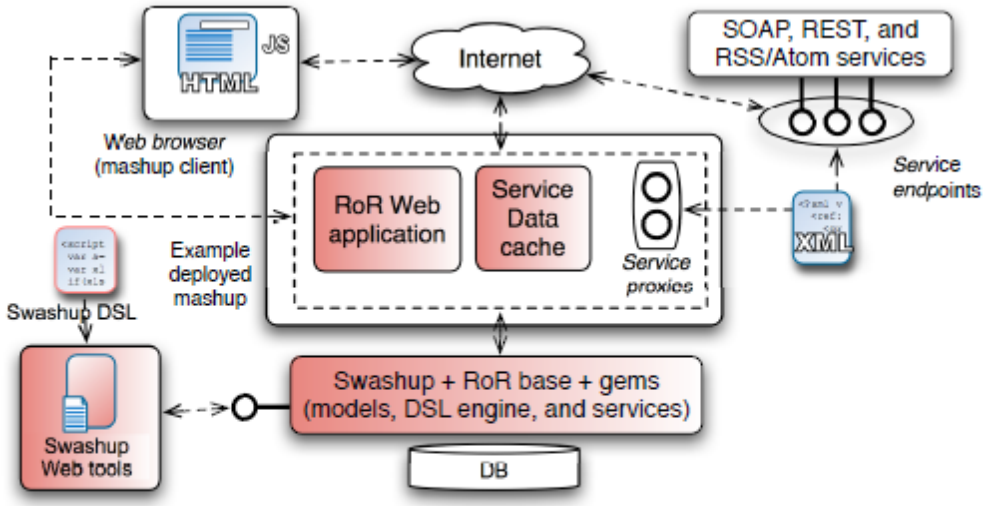


Figure 32 : Architecture de la plateforme Swashup (Maximilien, et al., 2007)

La syntaxe concrète du DSL proposé est sous forme d'une représentation textuelle, ci-dessous une description de ses principaux éléments :

- *Data* : décrit une information (*data element*) utilisée par un service (XML complexe type) ;
- *Api* : donne une description complète d'une interface de service (noms des opérations, paramètres, etc.)
- *Mediation* : décrit la transformation d'une ou plusieurs informations (*data element*) pour créer une nouvelle.
- *Service* : relie l'élément *api* avec le service concret (type de service (REST, SOAP, …), endpoint, etc.).

Dans le même sens, Rosenberg et al. *(Rosenberg, et al., 2009a) (Rosenberg, et al., 2009b)* ont défini le DSL VCL (Vienna Composition Language) permettant la spécification des compositions sensibles à la qualité de services (QOS) au sein de l'environnement VRESCO *(Rosenberg, et al., 2009a) (Rosenberg, et al., 2009b)*. En utilisant ce DSL, les contraintes QoS peuvent être spécifiées. Ces contraintes concernent les services individuels ainsi que les compositions de services.

L'utilisateur interagit avec le système en spécifiant une composition via le langage spécifique VCL. Ce dernier permet de spécifier des contraintes sur les entrées, les sorties, les pré-conditions, les post-conditions et la qualité de service (QoS).

Formellement, une hiérarchie de contraintes H est un ensemble de contraintes marquées (obligatoire; forte; moyenne; faible). L'utilisation d'hiérarchie de contraintes rend le processus de composition plus souple par rapport à un modèle qui n'utilise que les contraintes dures.

Lors de la réception d'une demande de composition, le service de composition génère un service composite en se basant sur la description initiale du VCL. Il sera déployé sur le moteur de composition sous-jacent et le point de connexion du service composite nouvellement créé sera retourné à l'appelant. Toutefois, si aucun service ne satisfait la spécification VCL, aucune composition valable ne sera générée.

La syntaxe concrète du langage VCL est une syntaxe textuelle. Une spécification d'un service composite (CS) est un tuple CS = <FD; FC; GC; BPS> (Figure 33), où :

- FD (*Feature Definition*) : représente les fonctionnalités (services) à composer ;

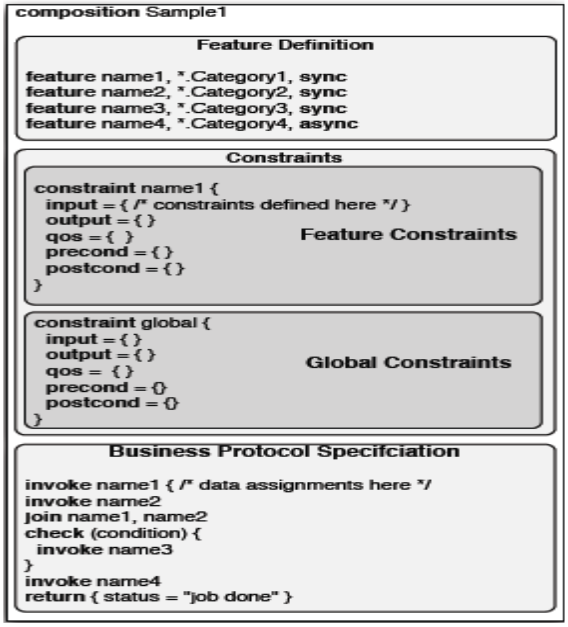


Figure 33 : Structure globale d'un programme VCL (Rosenberg, et al., 2009b)

- FC (*Feature Constraints*) : représente un ensemble optionnel de contraintes sur les fonctionnalités à composer (données input, output ; pré-conditions et post-conditions et contraintes QoS) ;
- GC (*Global Constraints*) : identiques aux « *Feauture Constraints* », mais ils concernent la composition ;
- BPS (*Business Protocol Specification*) : permet de définir les services à invoquer et permet également de spécifier des conditions et des boucles d'exécution.

II.3.2.5. Discussion

Le Tableau 4 présente le résultat de l'étude comparative des différents DSLs étudiés (Lethrech, et al., 2012). Nous nous sommes essentiellement focalisés sur leur prise en charge du concept d'adaptation. Les critères de comparaison adoptés sont comme suit :

- Variabilité de la logique métier ***(CV1)*** : variation de l'algorithme de la fonction atomique utilisée ;
- Variabilité d'attributs ***(CV2)*** : variation du nombre et du type d'attributs utilisés par une fonction atomique ;
- Variabilité du flux de travail ***(CV3)*** : variation de la séquence d'invocation des méthodes utilisées par une composition ;
- Variabilité de la présentation graphique ***(CV4)*** : l'interface graphique est adaptable selon le dispositif utilisé par l'utilisateur (PC, PDA,…) ;
- Variabilité de la localité ***(CV5)*** : machine qui héberge le service.

En plus de l'ensemble des types de variabilités cités ci-dessus, les critères suivants sont aussi considérés :

- Indépendance de la technologie cible ***(C6)*** : indépendance de la technologie d'implémentation ;
- Génération automatique du code source ***(C7)*** : automatisation des étapes de transformation ;
- Faible couplage ***(C8)*** : Séparation entre présentation, métier, adaptation, etc ;
- Composition de services ***(C9)*** : Possibilité de modéliser une orchestration de services.

Tableau 4 : Résultat de l'étude comparative des DSLs étudiés

	Critères de variabilité					**Autres critères**			
	CV1	***CV2***	***CV3***	***CV4***	***CV5***	***C6***	***C7***	***C8***	***C9***
DSLs d'Achilleas	O	O	X	X	O	X	X	X	O
ArchiMeDeS	O	O	O	O	O	X	X	--	--
D-SOA	O	O	X	O	X	O	X	X	X
mTurnpike	O	O	O	O	O	X	X	X	--
DSL de Saleem et al.	O	O	--	O	O	--	--	--	X
SOLj	--	--	--	--	--	O	O	--	X
Quala	O	O	O	O	O	X	X	X	--
Swashup DSL	O	X	X	O	--	--	--	X	X
VCL	O	O	X	O	O	O	--	X	X

X : critère pris en charge O : critère non pris en charge -- : non déterminé

Nous reprochons au travail d'Achilleos (Achilleos, 2009) le fait que malgré l'utilisation de l'approche DSM, l'intervention manuelle des développeurs est nécessaire pour compléter le code source. En outre, le travail d'Achilleos ne s'intéresse ni à la modélisation de la variabilité de service ni à la modélisation des règles d'adaptation. En sus, sa proposition ne traite pas l'interaction entre les services puisqu'elle s'occupe de la production d'un seul service à la fois. Leur méta-modèle de contexte modélise plusieurs préoccupations du contexte. Notamment, la catégorisation du contexte, la validité du contexte, la qualité du contexte, la confidentialité du contexte, etc. En revanche, la modélisation des relations entre les éléments du contexte n'est pas prise en charge. Finalement, nous soulignons que malgré le caractère spécifique du générateur du code source il utilise deux modèles au niveau M1, PIM et PSM (Figure 56).

En ce qui concerne le travail de Lopez-Sanz et al. (López-Sanz, et al., 2012), leur approche de modélisation est une approche hybride, ils ont essayé de profiter des avantages de l'approche DSM tout en bénéficiant de la popularité de la notation graphique d'UML. La notion d'adaptation n'est pas traitée par leur DSL.

Certes le framework *mTurnpike* (Wada, et al., 2005a) (Wada, et al., 2005b) (Wada, et al., 2005c) permet une séparation des préoccupations entre le concepteur, le développeur et l'ingénieur de la plateforme[1], en revanche, le processus de développement *mTurnpike* est complexe vu qu'il comporte beaucoup d'étapes. En outre, *mTurnpike* ne respecte pas la règle d'art de l'approche DSM (Kelly, et al., 2009b) qui consiste à transformer les modèles spécifiques directement au code source final (transformer un modèle spécifique en modèle UML est insensé pour Kelly (Kelly, et al., 2009b)).

L'utilisation de l'ingénierie dirigée par les modèles (IDM) et la séparation des préoccupations ont été respectées par presque l'ensemble des DSLs. En outre, la syntaxe concrète de l'ensemble des DSLs étudiés est soit textuelle soit graphique.

Nous retenons essentiellement de ce tableau comparatif que la plupart des critères d'adaptation utilisés ne sont pas pris en charge par les DSLs étudiés. Nous tenons aussi à souligner que l'utilisation de l'ingénierie des langages spécifiques au domaine par les SOSs reste très limitée et s'intéresse essentiellement à des préoccupations transverses à l'instar de la sécurité, l'orchestration, la qualité de service, etc.

Finalement, nous citons, ci-dessous, quelques bonnes pratiques concernant la conception d'un système orienté services à base de DSLs :

[1] S'occupe des règles de transformation.

- En général le code source généré automatiquement est difficile à lire et à comprendre (Wada, et al., 2005c). Par conséquent, il faut procéder à une génération totale du code sans l'intervention des développeurs, sinon l'intervention des développeurs doit être à un niveau d'abstraction élevé à l'instar de *mTurnpike* (Wada, et al., 2005b) (Wada, et al., 2005a) ;
- Quelques mauvaises pratiques de l'approche DSM sont à éviter : i) non maitrise du problème du domaine ; ii) extension d'un langage de modélisation universel (UML par exemple) ; iii) centrer le langage sur le détail technique du code ; iv) choisir un formalisme de représentation (textuelle ou graphique) erronée (Kelly, et al., 2009b) ;
- Il faut adopter une approche incrémentale lors de la mise en œuvre des DSLs (Oberortner, 2011) (Bierhoff, et al., 2006) ;
- Les noms utilisés au niveau du modèle du DSL doivent être reproduits au niveau de la syntaxe concrète du DSL (Oberortner, et al., 2008).

II.3.3. Synthèse

Comme déjà cité au niveau de la section II.2.2 un grand écart existe entre le problème d'un domaine donné et sa solution (généralement sous forme de code source). La Figure 34 présente quatre types de ponts entre le domaine et le produit final. Dans les deux premiers cas, les concepts du domaine doivent être manuellement transformés aux concepts des langages de développement. La troisième alternative consiste en l'utilisation de langages de modélisation universels, tel que UML. Sachant qu'un pourcentage relativement faible de code peut être généré automatiquement à partir de ces modèles, le développeur doit résoudre le problème à trois reprises : la première en utilisant les termes du domaine (souvent dans sa tête), la deuxième en utilisant UML et la troisième en utilisant le code source.

Par contre, dans le monde DSM, la solution du domaine est directement décrite, graphiquement ou textuellement, via un langage spécifique. Les modèles du domaine sont directement transformés au produit final.

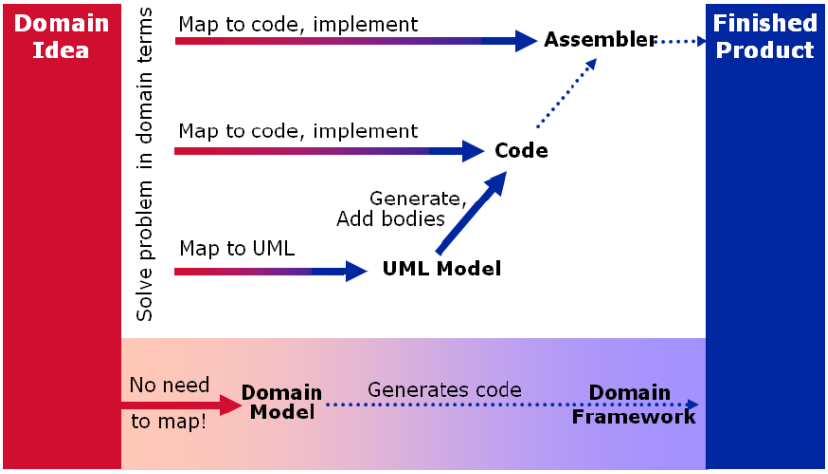


Figure 34 : Types de correspondance entre un problème d'un domaine spécifique et ses solutions (MetaCase, 2012)

D'une part, l'OMG a proposé un concept permettant de personnaliser UML pour un domaine spécifique. Ce concept appelé *Profil UML* est basé sur des mécanismes d'extension (*stereotypes, tagged values et constraints*) (Sun, et al., 2010b) (Kenzi, 2010). L'OGM a standardisé plusieurs profils UML à l'instar de *Systems Modeling Language* (SysML), *UML Testing Profile* (UTP), etc (OMG, UML, 2016).

Par contre, les profils UML n'ont pas eu beaucoup de succès au niveau du développement totalement dirigé par les modèles (Kelly, et al., 2008a). En effet, des études empiriques mesurant la productivité confirment une dégradation de la productivité en utilisant UML. Par exemple, une étude menée par la société Modelware (MODELWARE, 2006) au niveau de quatre sociétés différentes, affirme qu'aucune différence n'existe entre l'utilisation d'UML comme langage de modélisation et l'approche manuelle dirigée par le code source.

Il importe de signaler que depuis 2004 UML n'est plus une condition pour adopter MDA (Bonnet, 2005), un langage de modélisation spécifique conforme au MOF peut être utilisé. Encore plus, Grady Booch et al. affirment que MDA n'atteindra ses objectifs que si les concepts de modélisation correspondent directement aux concepts du domaine au lieu des concepts technologiques (Booch, et al., 2004). Deux mécanismes pour réaliser des DSLs standardisés sont proposés par l'OMG (OMG, MOF, 2016) (OMG; Soley, R. M., 2001) (Achilleos, 2009) : définition d'un langage qui se base sur le MOF (Meta-Object Facility) et les profils UML.

Beaucoup d'encre a été coulé ces dernières années concernant l'approche d'ingénierie des langages spécifiques au domaine à adopter : profil UML ou DSL. Beaucoup de critiques ont avancés qu'UML souffre d'un manque de richesse sémantique (Saleem, et al., 2012). En

outre, Oberortner et al. (Oberortner, et al., 2009) ont mis l'accent sur le fait que la maitrise d'UML est un prérequis pour l'utilisation d'un profil UML, ce qui nécessite le déploiement d'un effort supplémentaire pour les équipes fonctionnelles. En plus, Iseger (Iseger, 2010) considère que la génération totale du code source en utilisant les profils UML nécessite la maitrise d'un langage d'actions et l'utilisation d'OCL (d'autres langages à maitriser par le concepteur). En sus, un profil UML hérite d'UML même les caractéristiques indésirables, soit parce qu'ils ne sont pas pertinentes ou qu'ils doivent être limitées pour un profil donné (López-Sanz, et al., 2012). Ces caractéristiques sont généralement difficiles à enlever des outils de modélisation existants.

D'autre part, l'utilisation de langages spécifiques externe au monde UML suscite de plus en plus l'intérêt de la communauté de recherche spécialisée dans l'ingénierie dirigée par les modèles (Biehl, et al., 2014) (López-Sanz, et al., 2012) (Bonnet, 2005). Les avantages des DSLs proviennent de deux principes de base de la conception des langages : l'abstraction et la restriction ; Le choix des abstractions appropriées facilite les phases d'analyse des besoins, de conception, d'implémentation, et de maintenance. En outre, le fait de restreindre un langage à un domaine spécifique permet de rendre sa syntaxe concrète plus proche de la sémantique du domaine. Un autre point fort des DSLs consiste dans le fait que l'environnement de modélisation spécifique peut contraindre et valider un modèle par rapport à la sémantique du domaine, chose qui n'est pas possible pour les profils UML (Dalgarno, et al., 2008).

Retenons aussi que la volonté de Microsoft d'utiliser UML pour la couche logique (PIM) et les DSLs pour la couche physique (PSM) a été fortement critiquée (Cameron, 2008) (Kelly, 2008b). En réalité, UML est plus spécifique que les DSLs concernant le choix de l'implémentation (restriction à un langage orienté objets ; les éléments d'implémentation : classes, méthodes et les champs sont spécifiés directement au niveau des modèles). En d'autres termes, un DSL a un niveau d'abstraction plus élevé qu'UML. Par ailleurs, bien qu'IBM soutienne la philosophie MDA, elle insiste tout de même sur l'utilisation des langages spécifiques pour représenter et manipuler les concepts spécifiques d'un domaine (Bonnet, 2005).

Par contre, d'autres critiques de l'approche DSL existent, notamment le coût élevé de la première implémentation de la solution DSM, l'interopérabilité entre les DSLs est à étudier et à implémenter, des supports de formation à produire, etc.

Le Tableau 5 synthétise l'ensemble des points forts et points faibles des deux approches.

Tableau 5 : DSL vs Profil UML

Profil UML	
Points forts	**Points faibles**
- Existence d'outils de modélisation, transformation, etc. - Syntaxe visuelle populaire.	- Manque de richesse Sémantique. - Connaissance de base d'UML est nécessaire. - Génération totale du code nécessite l'utilisation de d'autres langages (exemple d'OCL). - Faible niveau d'abstraction (éléments d'implémentation au niveau des modèles, restriction aux langages OO).
DSL héritant du MOF	
Points forts	**Points faibles**
- Haut niveau d'abstraction avec un domaine fonctionnel restreint. - Plus de productivité (génération totale du code source pour une famille de produits). - Meilleure qualité du code généré (pas de bugs syntaxiques ou logiques). - Validation des modèles par rapport à la sémantique du domaine est possible en utilisant l'environnement de modélisation ➔ pas de conception qui ne respecte pas les règles architecturales.	- Coût élevé de la première implémentation (syntaxe abstraite et concrète, transformations, outils de modélisation, etc.). - Interopérabilité entre les DSLs est à étudier et à implémenter.

En conclusion, l'adoption du paradigme DSM est fortement conseillée si la maitrise du domaine en question est totale, et s'il y a une visibilité des futurs domaines de développement, c'est-à-dire que l'effort initial de la création de la boite à outils DSL est amortissable par la suite (Kelly, et al., 2008a) (Kelly, S.; MetaCase, 2009a) (Kelly, 2008b) (Liu, et al., 2010) (Pitkänen, et al., 2006) (Gray, et al., 2008). Il y a aussi lieu de signaler qu'avec l'apparition des frameworks de méta-modélisation facilitant la création des boites à outils DSL, le coût de

mise en place d'une solution DSM a était largement réduit (voir la section II.2.2). Enfin, nous tenons à souligner que l'utilisation des langages de modélisation spécifiques indépendants des langages de modélisation universels représente un choix prometteur pour l'ingénierie du logiciel dirigée par les modèles, en l'occurrence l'ingénierie des SOSAs.

II.4. Technologies d'implémentation

Cette section vise l'introduction des technologies utilisées dans l'implémentation des SOSAs et aussi la justification de nos choix technologiques. Elle est divisée en deux sous sections : implémentations des services et implémentations de l'adaptation.

II.4.1. Implémentation des services

Vers la fin des années 90, le concept d'architecture distribuée basé sur des solutions comme CORBA, DCOM, RMI, …, a essayé de répondre au besoin de communication entre les SIs. Par la suite, d'autres technologies ont pris la relève, nous en décrivant ci-dessous la pile des standards des services web et la technologie REST.

II.4.1.1. Pile des standards W3C / OASIS

Comme déjà précisé dans la section II.2.1.3 les services web représentent l'implémentation la plus prometteuse de l'architecture SOA. Rappelons qu'elle est la technologie la plus utilisée dans le monde industriel.

Nous allons présenter, ci-dessous, une pile de standards et langages liés à cette technologie (Figure 35).

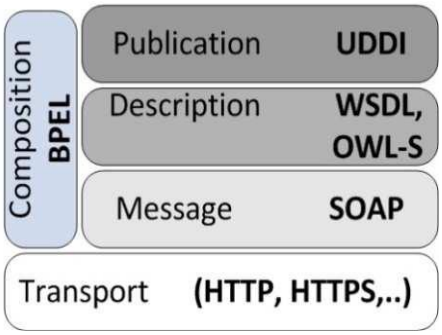


Figure 35 : Pile de standards et langages des services Web (Boukadi, 2009)

- SOAP (Simple Object Access Protocol) est un protocole de communication des services Web, standardisé par W3C. Les messages SOAP sont généralement transmis via Le protocole applicatif HTTP ;

- L'interface publiée d'un service Web est décrite en utilisant le *Web Service Definition Language* (WSDL). Ce dernier est un standard W3C, basé sur XML, initialement proposé par Microsoft et IBM ;
- OWL-S est une ontologie fondée sur le standard W3C OWL « Web Ontology Language ». Ce dernier s'occupe de la description sémantique des services ;
- Le standard OASIS UDDI « Universal Description, Discovery and Integration ». C'est un registre de services Web dont les descriptions y sont publiées sous forme de fichiers WSDL.
- BPEL « Business Process Execution Language For Web Services (BPEL4WS) » (OASIS, BPEL4WS, 2007) est un standard OASIS qui s'occupe de la composition de services Web en vue de produire des processus métiers.

II.4.1.2. Technologie REST

Une autre technologie qui a aussi apparue vers le début des années 2000[1] et qui suscite de plus en plus l'intérêt de la communauté de recherche est la technologie *REST* (REpresentational State Transfer) (Pautasso, et al., 2008). C'est un style architectural des systèmes distribués qui se base essentiellement sur les technologies web (standards HTTP et URI) appelé aussi architecture orientée web[2]. En effet, les services web REST exposent leurs fonctionnalités sous forme de ressources (URI) accessibles et identifiables en utilisant le standard HTTP. L'implémentation *REST* utilise le protocole HTTP au niveau de la couche applicative tandis que l'implémentation basée sur la pile de standards de la Figure 35 utilise le même protocole mais au niveau de la couche transport.

Certes, l'implémentation *REST* est plus facile à utiliser que la pile des standards illustrée dans la Figure 35, en revanche, plusieurs préoccupations transverses sont à développer manuellement. Nous citons par exemple la découverte de services, la gestion des transactions, la composition de services, etc. En outre, l'existence d'API[3] facilitant le développement des services web standards est un plus à considérer (Pautasso, et al., 2008). Dans le même sens, Pautasso et al. (Pautasso, et al., 2008) considèrent que l'implémentation fondée sur la pile de standards de la Figure 35 est la plus adaptée pour le monde professionnel.

[1] Créé par Roy Fielding en 2000 dans le chapitre 5 de sa thèse de doctorat (wikipedia).
[2] *Web-Oriented Architecture* dans le vocabulaire anglo-saxon.
[3] *Application Programing Interface* dans le vocabulaire anglo-saxon.

II.4.2. Implémentation de l'adaptation

II.4.2.1. Moteur de règles

Comme nous l'avons déjà expliqué au niveau de la section II.2.3.2, l'adaptation est soit : i) manuelle basée sur l'intervention humaine (concernant par exemple le déclenchement de l'adaptation, la stratégie, les mécanismes et les sujets de l'adaptation), soit ii) automatique assurée par un système expert[1] (Kabbaj, 2009). Un moteur de règles permet de rendre un système capable de faire un raisonnement logique et de produire des conclusions à partir d'une base de faits et d'une base de connaissances ou de règles.

L'utilisation d'un moteur de règles présente plusieurs avantages :

- Il permet de regrouper les règles métier de l'entreprise dans un seul endroit bien déterminé, constituant ainsi un référentiel métier de l'entreprise ;
- La définition des règles est proche du langage naturel ce qui permet même aux spécialistes du métier de spécifier et de mettre à jour les règles ;
- Facilite la maintenance ;
- Favorise le faible couplage, etc.

II.4.2.2. Programmation Orientée Context (COP)

Afin de faciliter le développement des applications sensibles au contexte, Hirschfeld et al. (Hirschfeld, et al., 2008) (Appeltauer, et al., 2011) ont proposé une nouvelle technique de programmation appelée *Context-Oriented Programming* (COP). COP qui est basé sur la POO, traite explicitement le contexte en prévoyant des mécanismes pour adapter dynamiquement le comportement du système aux changements du contexte.

Le principal concept du paradigme COP est le concept de couche (*Layer*). Les Couches agrègent les variations comportementales du système. Ils peuvent être activées et désactivées dynamiquement, même en cours d'exécution. L'application est composée de l'ensemble des couches activées à un moment donné en se basant sur l'état du contexte. La portée de l'activation/désactivation des couches peut être contrôlée de manière explicite. La même couche peut être simultanément active ou non pour différentes portées de la même application. L'implémentation la plus répondue du paradigme COP est l'extension du langage java *ContextJ* (Hirschfeld, 2011).

[1] Plus précisément, un système expert est un logiciel capable de répondre à des questions, en effectuant un raisonnement à partir de faits et de règles connues. Il peut servir notamment comme outil d'aide à la décision (wikipedia).

II.4.2.3. Programmation Orientée Aspect (AOP)

La programmation orientée aspect (AOP) permet de traiter, indépendamment, chaque préoccupation transversale d'un système. Elle s'occupe aussi de l'intégration des différentes préoccupations pour former le système final. Ce paradigme est utilisé par les systèmes adoptant la POO.

La programmation orientée aspect est largement utilisée dans l'implémentation des systèmes orientés service adaptables (Monfort, et al., 2009) (Hafiddi, 2012) (Boukadi, 2009) (Kazhamiakin, et al., 2010). D'une part, l'AOP considère l'adaptation comme une préoccupation transversale d'un système logiciel. D'autre part, l'AOP permet d'implémenter l'adaptation dynamique en utilisant le mécanisme de tissage d'aspects. Ce dernier permet de retarder l'intégration de l'adaptation au moment de l'exécution (Kazhamiakin, et al., 2010).

II.4.2.4. Programmation Orientée Caractéristique (FOP)

A l'instar d'AOP, le paradigme *Feature-Oriented Programming* (FOP) vise également l'externalisation des préoccupations transversales (Hirschfeld, et al., 2008). FOP est largement utilisée par les lignes de produit logiciel (SPL[1]). Une SPL est une famille de programmes définis par des combinaisons de caractéristiques[2]. Une caractéristique représente un incrément dans le développement d'un programme ou d'une fonctionnalité. Au niveau implémentation, une caractéristique du paradigme FOP est similaire à une couche (*Layer*) du paradigme COP. Par contre, la différence entre FOP et COP réside dans le fait que le premier se focalise sur la sélection et la combinaison des caractéristiques au moment de la compilation, tandis que le second outre le fait qu'il prend en charge l'activation et la désactivation dynamique de couches, il permet aussi de délimiter la portée de leurs activités (Cardozo, et al., 2011). L'une des implémentations du paradigme FOP est l'extension du langage Java *Jak*[3] (Batory, 2006). Le travail de Mohabbati et al. (Mohabbati, et al., 2013) fournit un aperçu de plusieurs travaux intégrant le paradigme FOP (modèle de caractéristiques) dans la mise en œuvre de solutions SOA.

[1] *Software Product Line* dans le vocabulaire anglo-saxon.
[2] *features* dans le vocabulaire anglo-saxon.
[3] Abréviation de *Jakarta.*

II.4.2.5. JCAF (Java Context Awareness Framework)

JCAF (JCAF, 2009) est un framework Java facilitant le développement d'applications sensibles au contexte. Il est composé de deux parties : i) l'infrastructure d'exécution (JCAF Runtime Infrastructure) ; ii) une API pour le développement des applications sensibles au contexte (JCAF Programming Model) à déployer sur l'infrastructure JCAF. L'API JCAF offre un ensemble de services pour l'exploitation de l'infrastructure d'exécution. Il est à noter que JCAF est le fruit d'un projet open source dont la communauté n'est plus active[1].

II.4.2.6. AWSDL et CWSDL

AWSDL (Lopez-Velasco, 2005) est une extension du standard WSDL permettant au fournisseur de services de décrire leurs différentes adaptations possibles. L'AWSDL se base sur le profil général de l'utilisateur (PGU). Ce dernier permet de présenter les besoins d'adaptation des utilisateurs en décrivant les caractéristiques personnelles de l'utilisateur ainsi que celles du contexte d'exécution de l'application. A l'instar d'AWSDL, le CWSDL (Mostefaoui, et al., 2004) vise l'intégration des informations du contexte au standard WSDL, et ce pour permettre l'adaptation des services aux contextes de leurs utilisateurs.

II.4.3. Synthèse

Nous avons essayé dans cette section d'illustrer les principaux choix d'implémentation de l'adaptation et de l'architecture orientée services. Il est à rappeler que la programmation orientée contexte est un concept nouveau qui est basé sur la programmation orientée aspect. La dernière implémentation JCOP [2] basée sur le langage Java a été produite dans le cadre des travaux de recherche du « Software Architecture Group » (Hirschfeld, 2011). Nous soulignons que JCOP et JCAF n'ont pas eu un grand succès dans le monde professionnel et leurs communautés ne sont plus très actives[3].

En plus, Les travaux AWSDL et CWSDL sont des travaux de recherche qui sont presque inexistant dans le monde professionnel.

Sur le plan implémentation de l'adaptation, nous avons opté pour l'utilisation d'un moteur de règles vu l'existence d'une multitude d'implémentations open source, sa présence très forte dans le monde professionnel et sa favorisation du faible couplage et la séparation des préoccupations.

[1] La dernière version JCAF date de 2011.
[2] JCOP est le successeur de ContextJ.
[3] La dernière version de JCOP date du mois 4 2012 et celle de JCAF date de 2011.

En ce qui concerne le choix technologique de l'implémentation des services, nous avons adopté une implémentation qui repose sur la pile de protocoles de la Figure 35. Ce choix est justifié par le fait que la technologie service web est bien outillée et qu'elle est largement utilisée.

II.5. Conclusion générale

Vu la complexité de la modélisation des SOSAs nous avons été amenés lors de cette étude bibliographique à étudier, séparément, les différentes facettes des approches de développement des SOSAs, à savoir, la modélisation des services, la modélisation du contexte et la modélisation de la logique d'adaptation. Nous avons divisé l'ensemble des travaux étudiés en deux catégories : approches à base d'UML et approches à base de DSLs. La majorité des approches de modélisation étudiées se basent sur le langage de modélisation générique UML, ou plus précisément sur des profils UML, pas très performants dans le monde IDM[1] (voir la section II.3.3). Aussi, notre étude sur les approches de modélisation des SOSs à base de DSLs nous a montré que l'adaptation est presque absente des différentes approches proposées (voir la section II.3.2.5).

On peut donc conclure que le problème de modélisation d'un système orienté services adaptable basée sur une approche générative est loin d'être résolu et que les travaux effectués dans ce domaine :

- Sont basés sur des langages de modélisation universels. Dans ce sens nous tenons à rappeler que les résultats de notre comparaison entre l'approche DSM (DSLs héritant du MOF) et UML nous ont permis de mettre en exergue l'avantage de la libération des langages spécifiques de l'héritage d'UML (voir la section II.3.3) ;
- N'intègrent pas l'ensemble des composants d'un système adaptable (service, variabilité de service, contexte et règles d'adaptation) ;
- Présentent un couplage fort entre les composants d'un SOSA ;
- Nécessitent tous l'intervention manuelle des développeurs.

Pour ces raisons, nous avons proposé une nouvelle approche de modélisation des systèmes orientés services adaptables qui distingue entre service, variabilité de service, contexte, règles d'adaptation et règles métier du domaine spécifique. Chaque composant est modélisé en utilisant un langage spécifique conforme au MOF. A l'instar d'une approche

[1] *Model Driven Engineering* dans le vocabulaire anglo-saxon.

DSM classique, des règles de transformations modèle-à-texte permettent de convertir l'ensemble des modèles à la solution finale. Nous avons aussi défini un processus de développement DSM pour le développement des SOSAs associé à notre approche. Cette dernière a été validée en utilisant une boite à outils spécifique au domaine de calcul et restitution d'impôts.

Le chapitre suivant décrit d'une façon approfondie notre vision de modélisation des systèmes orientés services adaptables basée sur l'ingénierie des langages spécifiques au domaine.

Chapitre III

CADSSOMA : Une approche de modélisation des SOSA basée sur l'ingénierie des langages spécifiques au domaine

III.1. Introduction

Il est bien connu que presque l'ensemble des cycles de développement logiciel accordent beaucoup d'importance à l'étape de conception. Cette dernière, est basée essentiellement sur la modélisation qui est, sans doute, une technique fondamentale permettant d'atténuer la complexité liée au développement des systèmes logiciels. A ce sujet, il importe de rappeler que les modèles apportent plusieurs avantages : i) la constitution d'un référentiel métier de l'entreprise, ii) la facilitation de la communication, iii) la séparation entre le monde métier et le monde technologique, iv) l'élévation du niveau d'abstraction à un niveau proche du monde réel, etc.

A présent, la modélisation a renforcé sa position au centre des nouveaux paradigmes de production de logiciels, notamment le développement dirigé par les modèles, et plus particulièrement l'approche DSM. A l'inverse des processus de développement traditionnels, ne prévoyant aucune liaison entre les modèles et le code source d'un système, l'approche DSM accorde aux modèles une place primordiale dans son processus de production. On parle, effectivement, de modèle source à la place de code source (Kelly, et al., 2008a), le code source final n'est que le résultat d'une transformation de modèles (Diaw, et al., 2008).

Dans cette visée, nous proposons dans ce chapitre, une approche de modélisation des systèmes orientés services adaptables. Cette approche intitulée CADSSOMA (***C****ontext-****A****ware, **D**omain **S**pecific and **S**ervice **O**riented **M**odeling **A**pproach*) est basée essentiellement sur l'approche DSM. Cette dernière consiste en l'élaboration de modèles spécifiques et leur transformation au code source final en utilisant un langage de transformation modèle-à-texte. Dans un souci de séparation des préoccupations et de maîtrise de la complexité de ce type de systèmes, cinq modèles sont à produire lors de la phase de modélisation : le modèle des services spécifiques au domaine, le modèle du contexte, le modèle de la variabilité de services, le modèle des règles d'adaptation et le modèle métier du domaine spécifique.

A l'instar d'une approche DSM classique, l'approche CADSSOMA est divisée en deux phases : la phase de modélisation et la phase de génération de la solution finale en utilisant l'outil support CADSSOTB. Cette dernière est détaillée au niveau du chapitre V.

Ce chapitre est organisé comme suit : la deuxième section décrit la syntaxe abstraite des différents DSLs utilisés dans la modélisation des services, de la variabilité de services, du contexte, des règles d'adaptation et des règles métier du domaine spécifique. La troisième section illustre notre approche en utilisant le domaine de calcul et restitution d'impôts. Elle est divisée en deux sous sections. La première définit le méta-modèle des services spécifiques au domaine et celui du métier du domaine spécifique (calcul et de restitution d'impôts). La deuxième prend en charge la spécification des cinq modèles de la phase de modélisation de notre approche CADSSOMA. Enfin, le chapitre est clôturé par une conclusion générale.

III.2. Syntaxes abstraites des DSLs

Un modèle doit obéir à des règles de modélisation définies par un langage spécifique. Ce dernier doit identifier la sémantique, la grammaire et la syntaxe permettant de réglementer la création des modèles, c'est-à-dire la création des entités et leurs relations. Un langage de modélisation spécifique au domaine est composé d'une syntaxe abstraite et d'une syntaxe concrète (Langlois, et al., 2007). Celle-ci représente la notation utilisée pour produire les modèles. Elle est souvent graphique ou textuelle. La syntaxe abstraite, appelée aussi méta-modèle, est une spécification formelle qui représente les éléments nécessaires pour produire des modèles. En d'autres termes, un méta-modèle est un modèle d'un langage de modèles et tout modèle doit être valide par rapport à son méta-modèle.

Comme nous l'avons explicité au niveau de la section II.3.3, et malgré le succès de leur langage de modélisation UML, l'OMG propose deux manières pour créer un DSL : les profils UML et les méta-modèles héritant du MOF. L'ensemble des DSLs proposés dans le cadre de notre approche sont des langages de modélisation spécifiques conforment au MOF. Ce choix a été discuté au niveau de la section II.3.3, justifié principalement par le fait qu'un DSL est sémantiquement très fort puisqu'il utilise les concepts d'un domaine fonctionnel restreint.

Le Meta Object Facility (MOF), standard de l'OMG, fournit un framework de gestion des métadonnées et des services associés. Il facilite le développement et l'interopérabilité des modèles et des systèmes dirigés par les métadonnées (OMG, MOF, 2015). En se basant sur le concept MDA, MOF définit des correspondances entre les modèles indépendants de la

plateforme (PIM) des métadonnées et des plateformes spécifiques (MOF-à-Texte, MOF-à-XML (spécification XMI), MOF-à-Java (JMI spécification), etc.) (OMG, MOF, 2015). À partir de sa deuxième version, le méta-méta-modèle MOF est composé de deux parties : Essential MOF (EMOF), pour la définition des méta-modèles sans associations, et Complete MOF (CMOF) pour la spécification des méta-modèles avec associations.

Il est à noter que depuis la version 2.4 le MOF a été aligné avec UML, et ce en partageant le même méta-modèle. En d'autres termes, le méta-modèle du MOF n'est qu'un sous ensemble du méta-modèle d'UML. Le MOF 2.5 réutilise un sous-ensemble des symboles de modélisation structurelle d'UML 2.5 utilisé dans la modélisation des classes. Retenons aussi que le MOF 2.5 ne définit aucun symbole de modélisation supplémentaire (OMG, UML, 2016).

Cette section est consacrée à la spécification de la syntaxe abstraite des langages spécifiques au domaine utilisés dans l'étape de modélisation de notre approche (Lethrech, et al., 2014) (Lethrech, et al., 2014), à savoir : le langage de modélisation des services, le langage de modélisation de la variabilité de services, le langage de modélisation du contexte, le langage de modélisation des règles d'adaptation et le langage de modélisation du métier du domaine spécifique.

III.2.1. Modélisation des services : méta-modèle générique

Le service représente l'élément élémentaire d'un système d'information orienté services. L'urbanisation de ce dernier passe inévitablement par une étape de modélisation de l'ensemble de ses composants primitifs. Pour ce faire, nous avons proposé un méta-modèle de service permettant d'organiser les éléments d'un système orienté services (Figure 36).

Les principales préoccupations prises en charge par notre méta-modèle sont comme suit :

- Généricité : à spécialiser par des éléments d'un domaine spécifique en utilisant le mécanisme d'héritage ;
- Extensibilité : le modèle doit permettre l'ajout d'un nouveau type de service d'une manière simple et intuitive ;
- Méta-modèle proche d'un standard : Intégration de l'ensemble des concepts communs des méta-modèles étudiés (Chapitre II), et ce afin de faciliter son adoption par les systèmes existants ;

- Méta-modèle conforme au MOF, indépendant de tout autre langage de modélisation.

Le fait de considérer que le méta-modèle représente la sémantique d'un langage est une fausse idée (Achilleos, 2009). En effet, le méta-modèle ne représente qu'une description de la syntaxe abstraite d'un langage. Une correspondance entre les concepts d'un méta-modèle et une description en utilisant un langage courant est nécessaire.

L'élément *Service* représente le cœur de notre méta-modèle générique de service. Les attributs *name, version* et *description* représentent l'identité du service. L'attribut *type* détermine le type de l'implémentation cible (*Rest* ou *SOAP*). L'élément service a été spécialisé en trois catégories :

- Service métier (*BusinessService*) : implémente une activité métier. Il est à spécialiser par des services métier du domaine spécifique cible (exemple du service *CTCalculationService*, voir la section III.3). De plus, un *BusinessServices* peut utiliser des *UtilityServices*. La relation « *uses* » du service A vers service B signifie que le service A fait appel au service B ;
- Service utilitaire (*UtilityService*) : représente une activité non fonctionnelle. C'est-à-dire qui n'a ni un objectif ni un sous objectif métier. Les services utilitaires sont utilisables par les services métier ;
- Service d'orchestration (*SchedulingService*) : en vue de produire un service d'une granularité plus importante, *SchedulingService* utilise les *BusinessServices* déjà implémentés via l'élément *ShedulingStrategies* (exemple du service *CTRestitutionProcess*, voir la section III.3). La classe *SchedulingService* est une classe abstraite, spécialisée par la classe *Process* qui représente un processus métier.

Un service possède :

- Une ou plusieurs *ServiceOperations*. Cette dernière possède plusieurs *ServiceParameters*. Ils sont spécialisés en *InServiceParameter* et *OutServiceParameter*, représentant respectivement les paramètres en entrée et les paramètres en sortie ;
- Au moins un contrat de service représenté par l'élément *SLA (Service Level Agreement)*. Il identifie les engagements de la qualité de service à respecter. Chaque *SLA* possède des conditions modélisées en utilisant l'élément *Condition,* et précisément via son attribut principal qui est sous forme d'une expression booléenne. Chaque *Condition* est connectée à des réactions (envoi d'email, SMS, etc.) qui sont une spécialisation de l'élément *UtilityService*. L'élément *Requester* représente l'utilisateur d'un service ;

- Au moins une interface de service représentée par l'élément *ServiceInterface*. Elle permet de modéliser les opérations fournies par un service. Un Service possède une seule *BaseServiceInterface*. Cette dernière regroupe l'ensemble des opérations immuables dont le comportement n'est pas influencé par le contexte d'exécution ;
- Un *ServicePackage*. Il regroupe plusieurs services permettant une meilleure organisation de l'application.

Deux éléments qui font partie du méta-modèle de la variabilité de service (voir la section III.2.2) sont représentés au niveau du méta-modèle générique de service (Figure 36). Le premier est l'élément *ShedulingStrategy*, il identifie les variantes de service ainsi que la stratégie d'orchestration (ordre d'exécution) utilisée par un *SchedulingService*. Le deuxième est *ServiceVariationPoint*, chaque service en possède au moins un. Il représente les variantes du service adaptable.

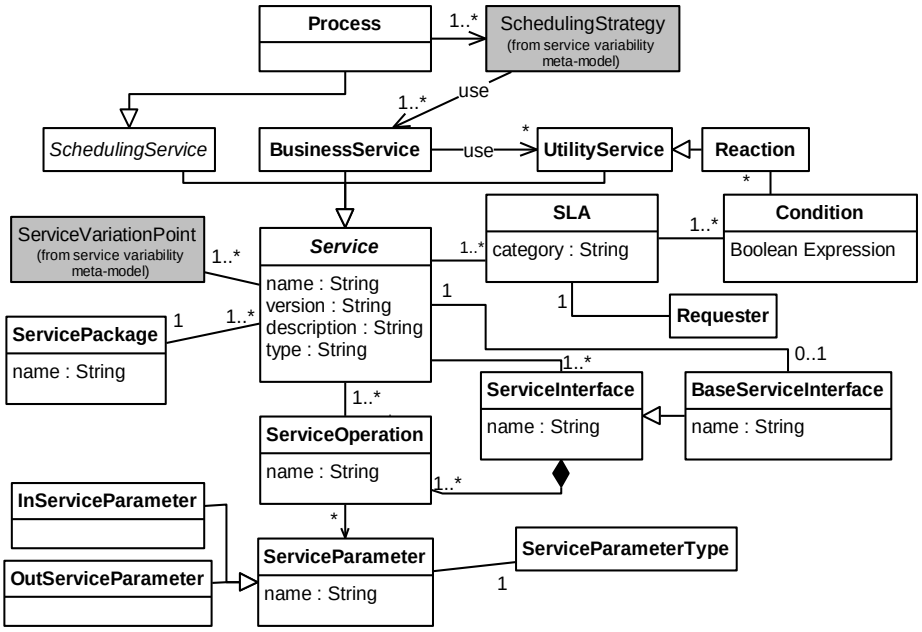


Figure 36 : Le méta-modèle générique de services

Le méta-modèle de service proposé (Figure 36) est un méta-modèle générique. Il doit être spécialisé par les éléments d'un domaine spécifique pour produire un méta-modèle des services spécifiques au domaine (Lethrech, et al., 2013). Nous tenons à souligner que le mécanisme de spécialisation d'un méta-modèle générique par des éléments d'un domaine spécifique est largement adopté par la communauté de recherche (Dhungana, et al., 2007) (Achilleos, 2009) (Berg, et al., 2011) (Beydoun, et al., 2009) (Clotet Martínez, et al., 2008) etc.

III.2.2. Modélisation de la variabilité de service

Comme déjà cité au niveau du chapitre II, l'adaptation traitée par notre approche est de type *designed* (Bucchiarone, et al., 2013) (ou statique (Canal, et al., 2006)), sensible au contexte (context-aware) (Kazhamiakin, et al., 2010) et manuelle (Canal, et al., 2006). Dans une adaptation de type *designed* les différentes variantes possibles d'un service doivent être connues au préalable, c'est-à-dire avant l'exécution du système.

Le mécanisme utilisé pour modéliser les variantes d'un service repose sur la modélisation explicite de la variabilité de service. Dans ce sens, et pour une meilleure prise en charge de cette dernière, elle doit être conçue et modélisée au niveau des premières étapes de notre approche de modélisation. Le choix de l'utilisation de la modélisation explicite de la variabilité est basé sur les résultats satisfaisants de l'utilisation de ce mécanisme par les lignes de produit logiciel et aussi par les systèmes orientés services (Clotet Martínez, et al., 2008). De plus, l'utilisation d'un modèle dédié de variabilité de service favorise la séparation des préoccupations.

La variabilité de service est une caractéristique (logique, représentation graphique, etc.) qui peut varier au sein d'un service selon le changement de son contexte d'utilisation. La variabilité de service est représentée par des points de variation (*variation point*). Un point de variation est une place dans un service où la différence se produit (Figure 37).

Les préoccupations prises en charge par le méta-modèle de variabilité proposé sont comme suit :

- Indépendance : afin de faciliter l'intégration de la notion de variabilité de service aux systèmes orientés service existants, la modélisation de la variabilité de service doit être faite en utilisant un modèle dédié ;
- Flexibilité : le modèle doit supporter l'addition de nouvelles variabilités de service de la manière la plus simple possible ;
- Extensibilité : le modèle doit supporter l'addition de nouveaux types de variation de service d'une manière simple et intuitive.

Chaque service possède plusieurs points de variation. Nous les avons classés en deux catégories :

- Variation fonctionnelle (*FunctionalVariation*) : traite la variation des règles métier. Elle est divisée en trois sous catégories :

- *Logic* : La logique interne d'un service peut varier d'un utilisateur à l'autre, ce type de point de variation permet de modéliser la variation de la logique métier d'un service ;
- *Interface* : pour la variation de la signature des opérations des services. Le même service peut avoir besoin de paramètres différents selon son contexte d'utilisation ;
- *SchedulingStrategy* : pour la variation de la séquence d'invocation et les variantes de services utilisés par un processus. Il est spécialisé en deux sous-catégories: *SequentialSchedulingStrategy* et *ParallelSchedulingStrategy*. La première utilise une stratégie d'exécution séquentielle et la deuxième est utilisée pour une exécution parallèle.

- Variation technique (*TechnicalVariation*) : représente les points de variations non fonctionnels, qui sont en relation avec des éléments techniques utilisés par le service. Elle est divisée en deux sous catégories :
  - *Locality* : pour la variabilité de l'emplacement du service, un service peut être hébergé par plusieurs serveurs avec différents URLs ;
  - *GraphicalPresentation* : l'interface de l'utilisateur est adaptable selon le dispositif utilisé par l'utilisateur et aussi selon les paramètres requis par le service utilisé ;

Le modèle de la variabilité relie le modèle de service et le modèle de contexte. En effet, l'élément service possède plusieurs *ServiceVariationPoints* et chaque *ServiceVariationPoints* est relié à une instance de l'élément *ServiceContextElement* du méta-modèle du contexte (voir la section III.2.3). Cette liaison et faite en utilisant le modèle des règles d'adaptation, explicité au niveau de la section III.2.4.

L'élément *ShedulingStrategy* est en relation avec l'élément *Process* qui fait partie du méta-modèle générique de service (voir la section III.2.1).

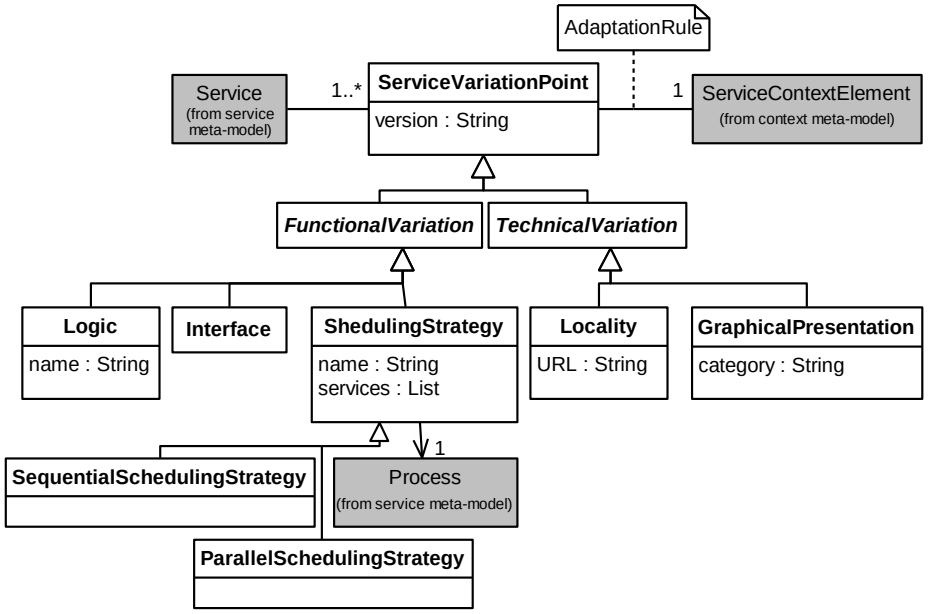


Figure 37 : Méta-modèle de la variabilité de service

III.2.3. Modélisation du contexte

Comme nous l'avons cité au niveau du chapitre II, le développement des systèmes adaptables repose essentiellement sur le paradigme CAC. Ce dernier permet essentiellement de modéliser et organiser l'information contextuelle pour faciliter son exploitation. Une multitude d'approches de modélisation du contexte existent dans la littérature scientifique, ci-dessous, nous en citons les principales catégories (Boukadi, 2009) (Chen, et al., 2000) (Bettini, et al., 2010) (Baldauf, et al., 2007) (Strang, et al., 2004) :

- Modèle pairs clef-valeur (*Key-value*) : est la structure de donnée la plus simple pour modéliser l'information contextuelle (Strang, et al., 2004) (Baldauf, et al., 2007). En particulier, les modèles clé-valeur sont faciles à gérer, cependant, ils ne permettent pas une description complète du contexte. En effet, ils présentent des faiblesses en ce qui concerne la structuration complexe des informations contextuelles, entre autres l'expression des relations qui peuvent exister entre ces derniers (Strang, et al., 2004) (Boukadi, 2009). D'autres travaux ajoutent la dimension degré de certitude au modèle clef-valeur (Schmidt, et al., 1999). On parle alors du triplet « clef-valeur-degré de certitude » ;
- Modèle de balisage (*markup models*) : les modèles de contexte basés sur le balisage utilisent une variété de langages de balisage, notamment XML (Bettini, et al., 2010). Tous les modèles à base de balisage utilisent une structure de données hiérarchique sous forme de balises avec des attributs et du contenu (Baldauf, et al., 2007). Le travail

de Bucchiarone et al. (Bucchiarone, et al., 2010a) et le standard W3C *Composite Capabilities/Preference Profile*[1] (CC/PP) sont des exemples illustrant cette catégorie ;

- Modèle graphique : cette approche utilise des modèles formels afin de modéliser les informations contextuelles. Nous citons par exemple le langage de modélisation UML qui possède une forte composante graphique[2]. Raik et al. qui ont utilisé le diagramme état-transition (Raik, et al., 2012) (voir la section II.3.1) et le travail de Sheng et al. (Sheng, et al., 2005) qui ont proposé le profil UML *ContextUML* (voir la section II.3.1.1) en sont la parfaite illustration. Il est aussi à noter que l'utilisation d'un langage spécifique (DSL) est aussi envisageable (Achilleos, 2009) ;
- Modèle orienté objet : une modélisation orientée objet du contexte est largement utilisée (Bettini, et al., 2010) (Baldauf, et al., 2007) (Strang, et al., 2004) (Chen, et al., 2000). Cette approche essaye de tirer profit des principaux avantages de l'approche orientée objet, à savoir l'encapsulation et la réutilisation (Strang, et al., 2004). En réalité, l'information contextuelle est représentée via les attributs (état de l'objet) et elle est mise à jour via les méthodes (comportement de l'objet) (Chen, et al., 2000). Le travail de Hofer et al. (Hofer, et al., 2003) en est une parfaite illustration ;
- Modèle basé sur la logique (*Logic-based model*) : Dans un modèle de contexte basé sur la logique, le contexte est défini comme des faits, des expressions et des règles. Habituellement l'information contextuelle est ajoutée, mise à jour et supprimée à partir d'un système à base de règles en termes de faits ou déduite des règles du système en utilisant le processus d'inférence. Le travail de Yahyaoui et al. (Yahyaoui, et al., 2011) illustre ce type de modèle. D'autres exemples existent au niveau de l'étude de Strang et al. (Strang, et al., 2004) ;
- Modèles basés sur des ontologies : en général les ontologies permettent de faire une description des concepts et des relations. Elles représentent un instrument très prometteur pour la modélisation de l'information contextuelle. Cela est justifié essentiellement par leur expressivité forte et formelle. En outre, cette approche permet non seulement de modéliser le contexte, mais aussi de raisonner sur les données décrites (Baldauf, et al., 2007) (Boukadi, 2009).

Malgré la multitude de modèles représentant le contexte (0) aucun formalisme standard permettant la modélisation des informations contextuelles n'est adopté (Vale, et al.,

[1] Utilisé dans la description des dispositifs mobiles.
[2] Quatorze types de diagrammes pour UML 2.3.

2009). De ce fait, nous avons proposé notre propre méta-modèle du contexte tout en tirant profit des bonnes pratiques de l'ensemble des modèles étudiés et aussi en essayant de satisfaire les exigences suivantes :

- Simplicité : les expressions et les relations utilisées doivent être assez simples pour simplifier le travail des développeurs ;
- Flexibilité et extensibilité : le modèle doit supporter, de la manière la plus simple possible, l'addition de nouveaux éléments contextuels ;
- Généricité : le modèle ne doit pas être limité à un domaine spécifique ou à une plateforme donnée, mais il doit supporter la modélisation de différents types de contextes indépendamment des domaines d'application et des plateformes.

Notre définition du contexte, qui est une jointure entre la définition de Dey (Dey, 2001) et celle de (Winograd, 2001), considère qu'un contexte est l'ensemble d'informations structurées, qui évolue, sert l'adaptation et qui caractérise la situation d'une entité. Une entité est une personne, un lieu ou un objet qui est considéré comme pertinent pour l'adaptation d'une application.

En se basant sur la définition précédente un contexte est composé de plusieurs *ContextElements* et un *ContextElements* est composé de plusieurs *ContextParameters*. Nous avons aussi défini l'entité *ServiceContextElement* qui est un *ContextElement* spécifique à un service. Un *ServiceContextElement* est composé de *ContextParameters*, appartenant à différents *ContextElements*. Ces *ContextParameters* représentent l'ensemble des éléments contextuels, responsables de l'adaptation d'un seul service. Chaque *ServiceVariationPoint* (voir la section II.2.3.4.1) est relié à une instance de *ServiceContextElement*.

Un contexte peut être divisé en plusieurs sous-contextes. Cette organisation peut par exemple être basée sur la source d'informations du contexte : environnement (température, position, niveau de la batterie, etc.), utilisateur (profil), capteur, etc. Notre méta-modèle du contexte est représenté dans la Figure 38.

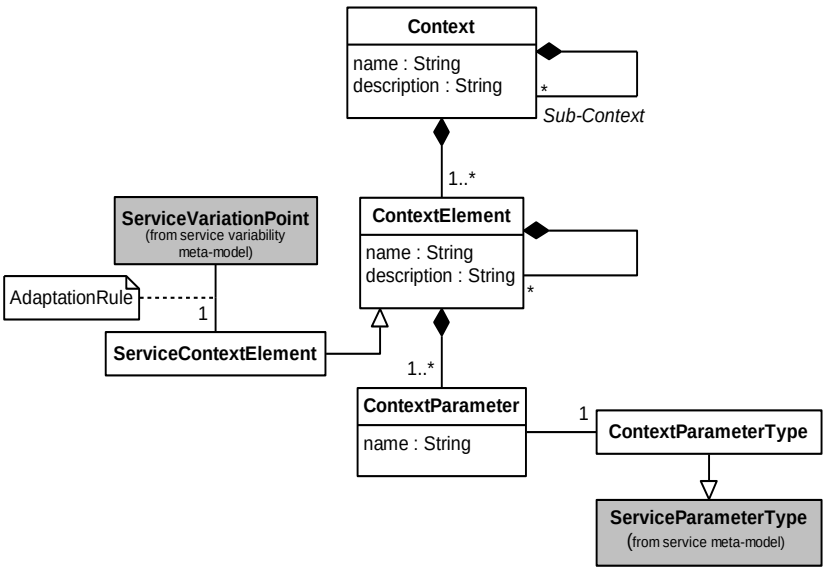


Figure 38 : Méta-modèle du contexte

III.2.4. Modélisation des règles d'adaptation

L'adaptation traitée par notre approche est une adaptation manuelle (Canal, et al., 2006), c'est-à-dire que la logique d'adaptation doit être spécifiée et mise en œuvre manuellement par des développeurs.

La modélisation de l'adaptation est assurée par le modèle des règles d'adaptation. Ce dernier doit satisfaire les exigences suivantes :

- Indépendance : afin de favoriser le faible couplage et la forte cohésion, le modèle de la logique d'adaptation doit être indépendant des autres modèles. Et ce pour faciliter la substitution d'une logique d'adaptation d'un service par une autre ;
- Exhaustivité : il doit permettre la modélisation de l'exhaustivité des règles d'adaptation ;
- Facilité d'utilisation : la mise en place du modèle doit être simple et intuitif.

Le modèle des règles d'adaptation est modélisé en utilisant des tables de décision (Pooch, 1974). Le concept de table de décision provient de l'approche classique basée sur le paradigme de séparation des données et des traitements. Les tables de décision ont été largement utilisées pour produire des spécifications des systèmes complexes (Boffoli, et al., 2009) (Vanthienen, et al., 1992), elles offrent les avantages suivants (Pooch, 1974) :

- Une grande facilité de lecture, même en présence de nombreuses conditions, actions et règles ;

- Une représentation précise de la logique interne d'une fonction (opération d'un service) ;
- Grâce au mécanisme de complétude, il est possible de garantir l'exhaustivité des règles applicables aux différentes données reçues en entrée.

Une table de décision est composée de quatre quadrants (Vanthienen, et al., 1992) (Pooch, 1974) (Figure 39). Le quadrant en haut à gauche appelé « *condition stub* », contient l'ensemble des éléments qui doivent être examinées pour un problème particulier. Les conditions à tester (string, ensemble de valeurs, expression, etc.) sont exprimées au niveau de la section « *Condition entry* ». Le quadrant en bas à gauche nommé « *action stub* » contient la liste des actions qui peuvent être déclenchées, résultant des conditions énumérées au niveau des quadrants précédents. Enfin, les actions appropriées résultant des différentes combinaisons de réponses aux conditions seront indiquées au niveau du dernier quadrant « *action entry* ».

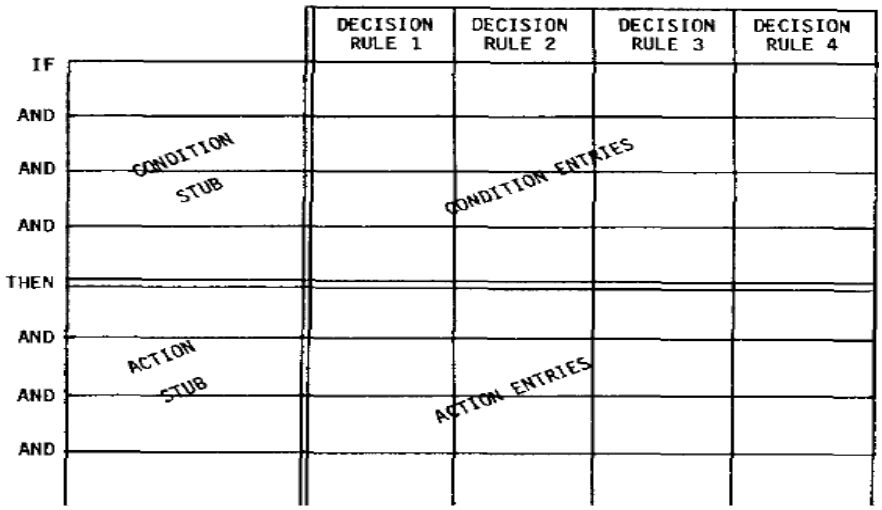


Figure 39 : Eléments de base d'une table de décision (Pooch, 1974)

Une table de décision exprime exhaustivement les relations entre la logique interne d'une opération ou d'un processus, les données fournies en entrée et les données restituées en sortie.

- Les données fournies en entrée deviennent les conditions de la table de décision : dans notre cas elles sont représentées par les paramètres des *ServiceContextElements* ;
- Les données restituées en sortie proviennent des actions déclenchées par la table de décision : dans notre cas elles sont sous forme de *ServiceVariationPoints* ;

- Les couples (paramètres des *ServiceContextElements*, *ServiceVariationPoints*) représentent les règles de la table de décision. Ces règles seront exploitées en utilisant un moteur de règle.

Les tables de décision utilisées sont des tables de décision complètes à entrées mixtes :

- Entrée mixte : permet plus d'ouverture et de flexibilité. Les valeurs de la section « *Condition entry* » sont soit sous forme d'expressions booléennes, soit des valeurs booléennes Vrai ou Faux. En outre, en plus de l'indication des actions à exécuter, la section « *action entry* » peut contenir les résultats de leurs exécutions ;
- Complète : la logique d'adaptation doit être représentée d'une manière complète. Par conséquent, toutes les combinaisons des conditions doivent être représentées. En d'autres termes, aucune règle ne doit manquer.

La logique d'adaptation pour l'approche CADSSOSMA représente la relation entre le modèle de la variabilité de service et le modèle du contexte. En d'autres termes, elle identifie la situation contextuelle nécessaire pour l'utilisation d'un variant de service.

Chaque service possède ses règles de sélection de variabilité de service. Elles sont représentées en utilisant une table de décision. Les paramètres du *ServiceContextElement* du service en question représentent les conditions de la table de décision, et les variantes du service représentent les actions. Pour un processus, les stratégies d'orchestration (*SchedulingStrategies*) représentent les actions de la table de décision.

La Figure 40 illustre notre méta-modèle de tables de décision. Nous avons classé les colonnes d'une table de décision en deux catégories :

- *ConditionActionColumn* : représente la colonne des conditions et des actions d'une table de décision. Une table de décision possède une seule colonne du type *ConditionActionColumn*. Elle est composée de :
    - *ConditionCells* : représentent les cellules de conditions. Ces derniers sont reliés aux *ContextParameters* constituant le *ServiceContextElement* du service en question.
    - *ActionCells* : représentent les cellules des actions. Ils sont directement reliés aux différentes variantes (*ServiceVariationPoint*) du service en question.
- *RuleColumn* : représente les règles d'adaptation d'une table de décision. Une table de décision possède plusieurs colonnes du type *RuleColumn*. Elle est composée

de plusieurs *ConditionActionValueCells*. Ils sont utilisés pour représenter les conditions (booleanExpression) sur les *ContextParameters*. En d'autres, termes ils représentent les règles d'adaptation du service. Les *ConditionActionValueCells* sont aussi utilisés pour déterminer les actions à exécuter.

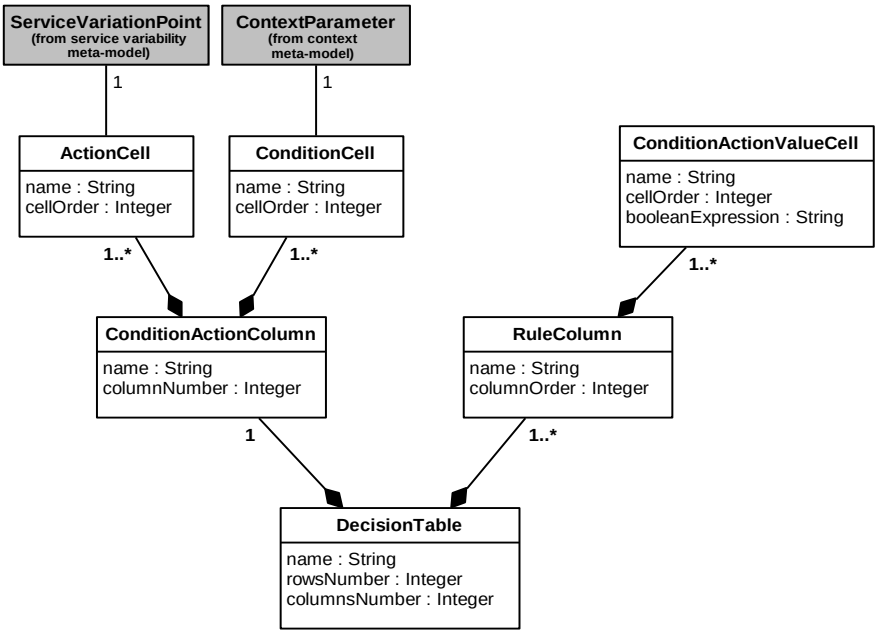


Figure 40 : Méta-modèle des règles d'adaptation (tables de décision)

Les tables de décision représentent un outil important pour la modélisation de la logique interne des applications. Bien qu'elles excellent dans le traitement unitaire des règles, elles ne permettent pas de garantir l'exhaustivité des traitements d'une fonction. D'où l'intérêt de la modélisation des règles métier du domaine en question.

III.2.5. Modélisation du métier du domaine spécifique

Le méta-modèle du langage dépend du domaine spécifique traité. Le développeur du domaine doit produire un DSL permettant la modélisation de la logique métier du domaine en adoptant l'approche DSM. Ce DSL est utilisé pour modéliser le cinquième modèle de notre approche intitulé modèle du métier du domaine spécifique. Il est à souligner que le modèle métier joue un rôle primordial dans la génération totale de la solution finale, et ce essentiellement grâce sa force sémantique. Cela permet de modéliser le comportement interne des opérations des services. Dans cette visée, nous tenons à rappeler que parmi les principaux objectifs de l'approche DSM est la génération totale des systèmes à partir des modèles, c'est pour cela qu'elle est aussi appelée « Programmation générative » (Bonnet, 2005). L'espace problème représente un ensemble d'abstractions en utilisant une notation spécifique à un domaine, et l'espace solution décrit un ensemble d'abstractions technologiques, représentant

l'implémentation du système spécifié et basé sur les artefacts de l'espace problème. Le mécanisme de transformation représente un trait d'union entre ces deux espaces.

Les exigences à prendre en considération lors de la mise en place du langage du métier du domaine spécifique sont ceux de l'approche DSM :

- Le domaine cible doit être un domaine restreint pour permettre l'élévation du niveau d'abstraction du langage, facilitant ainsi la génération totale de la solution finale ;
- Le domaine cible doit présenter une dynamique fonctionnelle. En d'autres termes, un seul méta-modèle doit être utilisé pour modéliser plusieurs produits. Cela permet d'amortir l'investissement initial de la solution DSM ;
- Une parfaite maitrise du métier du domaine cible est nécessaire.

Le modèle du métier du domaine spécifique représente un référentiel métier de l'entreprise, qui est considéré à lui seul comme précieux, même en l'absence d'un générateur de code (Kelly, et al., 2008a). La section III.3.2.2 présente un exemple de méta-modèle du métier de calcul et restitution d'impôts.

III.3. Etude de cas

III.3.1. Description de l'étude de cas : calcul et restitution d'impôts

Nous allons illustrer notre approche par un domaine spécifique qui présente une grande sensibilité au contexte. Notre choix s'est fixé sur le domaine de gestion d'impôts. Il est à noter qu'un système d'information de gestion d'impôts est supposé être mis à jour après chaque loi de finances. Par conséquent, il doit être flexible, évolutive et personnalisable autant que possible. Cette instabilité fonctionnelle représente, selon Kelly (Kelly, et al., 2008a), l'un des points essentiels justifiant l'adoption de l'approche DSM. Pour un souci de simplicité nous nous somme focalisés sur le domaine de calcul et restitution de l'impôt sur les sociétés (IS).

Une petite comparaison entre la loi de finances 2016[1] et celle de 2015[2], en ce qui concerne les mesures spécifiques à l'impôt sur les sociétés, montre l'introduction de plusieurs mises à jour, entre autres nous citons :

[1] Bulletin officiel N° 6423 bis du 21 décembre 2015.
[2] Bulletin officiel N° 6320 bis du 25 décembre 2014.

- Taux de calcul de l'IS :

En 2015

- 10% pour les sociétés réalisant un bénéfice fiscal inférieur ou égal à trois cent mille (300.000) dirhams ;
- 30 % pour les sociétés réalisant un bénéfice fiscal supérieur à trois cent mille (300.000) dirhams ;
- 37 % pour les établissements de crédit et organismes assimilés, Bank Al Maghrib, la Caisse de dépôt et de gestion, les sociétés d'assurances et de réassurances.

En 2016

- 10% pour les sociétés réalisant un bénéfice net fiscal inférieur ou égal à trois cent mille (300.000) dirhams ;
- 20% pour les sociétés réalisant un bénéfice net fiscal supérieur à trois cent mille (300.000) dirhams et inférieur ou égal à un million (1.000.000) dirhams ;
- 30% pour les sociétés réalisant un bénéfice net fiscal supérieur à un million (1.000.000) dirhams et inférieur ou égal à cinq millions (5.000.000) dirhams ;
- 31% pour les sociétés réalisant un bénéfice net fiscal supérieur à cinq millions (5.000.000) dirhams.
- 37 % pour les établissements de crédit et organismes assimilés, Bank Al Maghrib, la Caisse de Dépôt et de Gestion, les sociétés d'assurances et de réassurances.

- Suppression de l'imputation de la cotisation minimale en matière d'IS en 2016.

Les principaux services de notre cas d'utilisation sont *TaxCalculation* et *TaxRestitutionProcess*. Le premier est responsable du calcul d'impôts et le deuxième s'occupe de la restitution d'impôts. Ce sont des services génériques à spécialiser selon l'impôt cible. Par exemple *CTCalculationService* (*corporation tax calculation service*) est responsable du calcul de l'impôt sur les sociétés. Le service *CTRestitutionProcess* est un processus composé de plusieurs services visant la restitution de l'impôt sur les sociétés.

La logique métier de ces services dépend de plusieurs paramètres qui changent d'une loi de finances à une autre, comme : les exonérations du contribuable, la catégorie du contribuable, le régime du contribuable, etc. Par exemple, la logique métier de calcul de l'impôt sur les sociétés (service *CTCalculationService*) peut être catégorisée comme suit :

- Le calcul de l'impôt d'un contribuable totalement ou partiellement exonéré[1] : nous citons l'exemple du taux réduit pour le chiffre d'affaires à l'exportation.

  En effet, la partie du bénéfice proportionnelle au chiffre d'affaires à l'exportation est exonérée de l'impôt sur les sociétés pendant une période de cinq ans consécutifs, qui court à compter de l'exercice au cours duquel la première opération d'exportation a été réalisée. Ensuite le chiffre d'affaires à l'exportation est imposé au taux réduit de 17.5%.

  Le reste du bénéfice relatif au chiffre d'affaires local est imposé au taux normal ;
- Le calcul de l'impôt pour les contribuables non-résidents : pour les sociétés non résidentes adjudicataires de marchés de travaux, de construction ou de montage ayant opté pour l'imposition forfaitaire, la base de calcul de l'impôt est le montant du marché, et non pas le bénéfice (8% du montant du marché) ;
- Taxation d'office : si le contribuable ne satisfait pas l'obligation de dépôts de déclaration du résultat fiscal et du chiffre d'affaires, l'inspecteur sera dans l'obligation d'émettre un titre de recette dont le principal des droits est calculé en se basant sur les quatre dernières années ;
- Le mode de calcul normale : la base de calcul de l'impôt est le bénéfice de l'exercice en question, le taux à appliquer est le taux normal de l'IS.

En outre, et en vue de fournir une qualité de service adaptée au comportement du contribuable, l'administration fiscale a introduit le concept de catégorisation des entreprises. Après un audite effectué par les inspecteurs de l'administration fiscale le contribuable sera catégorisé. Trois catégories sont proposées :

- Catégorie A : permet d'accorder une avance de restitution de 80% sur les demande de restitution d'impôts ;
- Catégorie B : permet d'accorder une avance de restitution de 50% sur les demande de restitution d'impôts ;
- Catégorie C : pas d'avance de restitution d'impôts.

En plus de l'adaptation de la logique métier, il est possible d'imaginer d'autres types d'adaptations, par exemple l'adaptation en relation avec le dispositif utilisé par le contribuable, la position du contribuable, etc.

La phase de modélisation de notre approche est divisée en deux étapes. La première traite la définition des méta-modèles (syntaxe abstraite). Deux méta-modèles sont concernés

[1] Article 19 et article 6 du code général des impôts 2016.

(*Step1*, en vert dans la Figure 41) : i) Méta-modèle des services spécifiques au domaine, résultant de la spécialisation de notre méta-modèle générique des services par les éléments et les services du domaine spécifique (voir la section III.2.1) ; ii) Méta-modèle du métier du domaine spécifique. La deuxième étape porte sur la spécification des modèles (modélisation des cinq modèles, *Step 2* de la Figure 41).

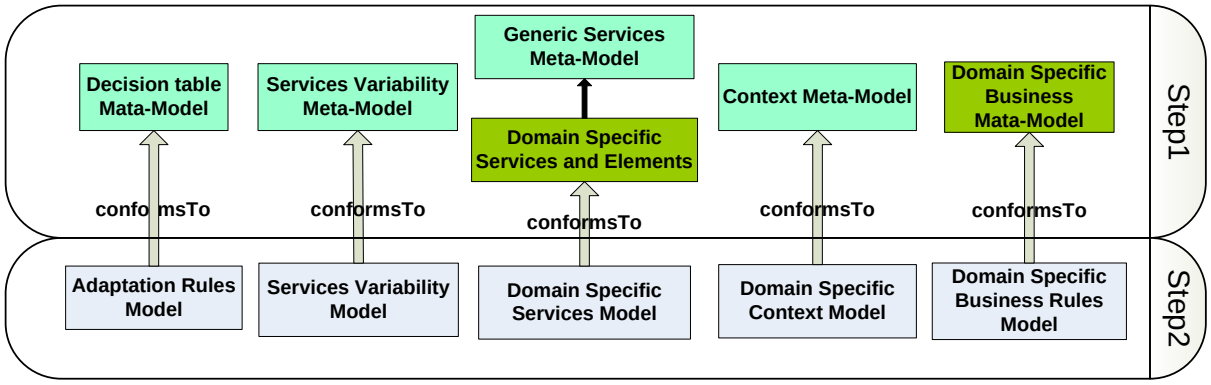


Figure 41 : Phase de modélisation de l'approche CADSSOMA

III.3.2. Définition des méta-modèles

III.3.2.1. Méta-modèle des services spécifiques au domaine : services de calcul et de restitution d'impôts

Le développeur du langage doit produire son méta-modèle des services spécifiques au domaine en héritant de notre méta-modèle générique de services (Figure 36). Dans cette optique, nous allons ajouter les éléments de notre domaine cible à notre méta-modèle générique des services pour produire notre méta-modèle des services de calcul et de restitution d'impôts. En plus des éléments génériques, ce dernier est divisé en méta-services spécifiques au domaine et méta-éléments spécifiques au domaine (Figure 42).

Pour la partie de calcul d'impôts, le principal service est le service *TaxCalculation*. Il permet de calculer la valeur d'un impôt. Ses services utilitaires sont : *GetTaxRate*, *GetRatePayerInfo* et *GetRatePayerExemption*. Ils permettent respectivement de récupérer le taux à appliquer, les informations du contribuable et les exonérations du contribuable.

Pour la partie restitution d'impôts nous utilisons les *BusinessServices* suivants :

- *DemandDeposit* : permet de déposer la demande de restitution d'impôts ;
- *AdvanceRestitution* : traite la restitution automatique qui est basée sur la catégorie du contribuable (A, B ou C) ;
- *RestitutionRecordVerification* : effectue une vérification du dossier de restitution (factures, charges, etc.) ;

- *TaxRestitution* : effectue la restitution du reliquat restant.

Des services utilitaires sont utilisés par les services métier cités ci-dessus :

- *GetDeficit* : permet de récupérer le déficit du contribuable de l'exercice précédent ;
- *GetExcess* : permet de récupérer l'excédent du contribuable de l'exercice précédent ;
- *GetMinimumContributionCredit* : permet de récupérer le crédit de la contribution minimale du contribuable de l'exercice précédent.

Les méta-éléments spécifiques au domaine de calcul et restitution d'impôts sont comme suit :

- *Tax :* représente un impôt donné ;
- *RatePayer :* représente un contribuable, le principal utilisateur de notre système ;
- *TaxDeclaration :* représente une déclaration d'un impôt ;
- *Exemption :* représente une exonération du contribuable ;
- *TaxCalculationSLA :* représente le contât de qualité de service du service *TaxCalculation* ;
- *TaxRestitutionSLA :* représente le contât de qualité de service du service (processus) *TaxRestitution*.

Un impôt (*Tax*) peut avoir plusieurs déclarations (*TaxDeclarations*). Un contribuable (*RatePayer*) peut déposer plusieurs *TaxDeclarations*, comme il peut bénéficier de plusieurs exonérations (*Exemptions*).

Les deux éléments *RatePayer* et *TaxDeclaration* héritent de l'élément *ServiceParameter* appartenant au méta-modèle générique de service. En d'autres termes ces deux éléments sont des paramètres d'entrées des services de calcul et de restitution d'impôts. *TaxCalculationSLA* et *TaxRestitutionSLA* sont des spécialisations de l'élément générique *SLA*, représentant les contrats de qualité de service.

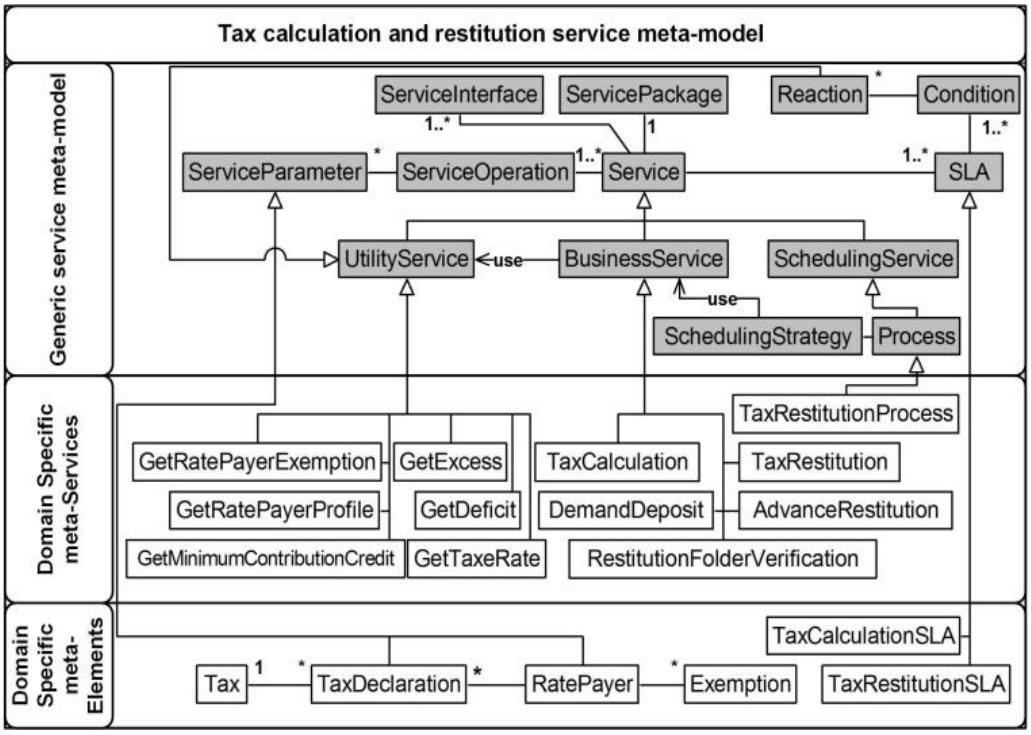


Figure 42 : Méta-modèle des services de calcul et de restitution d'impôts

En se basant sur notre méta-modèle de calcul et de restitution d'impôts, nous pouvons créer des modèles pour chaque impôt (impôt sur le revenu, impôt sur les sociétés, taxe sur la valeur ajoutée, etc.).

III.3.2.2. Méta-modèle du métier du domaine spécifique : domaine de calcul et restitution d'impôts

La logique de calcul de l'impôt est représentée sous forme de formules mathématiques. La Figure 43 représente notre méta-modèle de base utilisé dans la modélisation des formules de calcul d'impôts. Ces dernières sont représentées en utilisant l'élément *TaxCalculationFormulaDiagram*. Il est composé d'opérateurs, d'entrées et de résultats.

- L'élément *Operator* représente les opérateurs de la formule de calcul. En plus de l'opérateur *Max* qui permet de déduire la valeur la plus grande entre plusieurs valeurs en entrée, nous avons représenté les opérateurs arithmétiques *PlusOperator*, *MinusOperator*, *MultiplicationOperator* et *DivisionOperator*. D'autres opérateurs peuvent être ajoutés par la suite ;
- L'élément *Entry* représente les opérandes de la formule de calcul. Il hérite de l'élément *Number* qui est en relation directe avec l'élément *OperatorInput* ;
- L'élément *Result* représente le résultat de la formule de calcul. Il hérite de l'élément *Number*. Ce dernier possède un nom et une valeur.

La représentation graphique de la logique de calcul d'impôts permet de faciliter sa spécification par les spécialistes fonctionnels. Cette spécification représente, bien entendu, un référentiel fonctionnel de l'administration fiscale.

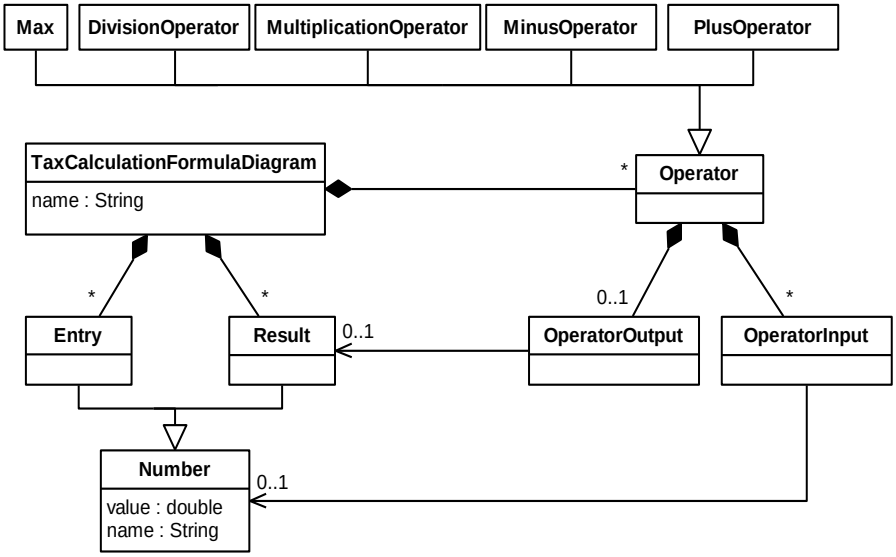


Figure 43 : Méta-modèle des formules de calcul d'impôts

III.3.3. Spécification des modèles

III.3.3.1. Modèle des services de calcul et de restitution de l'impôt sur les sociétés

Le modèle des services de calcul et de restitution de l'impôt sur les sociétés est représenté dans la Figure 44. Le service *CTCalculationService* permet de calculer la valeur de l'impôt sur les sociétés. Il utilise les services utilitaires suivants : *GetCTRate*, *GetRatePayerProfile*, *GetRatePayerExemption*, *GetCTDeficit*, *GetCTExcess* et *GetCTMinimumContributionCredit*. Ils permettent, respectivement, de récupérer le taux de l'impôt sur les sociétés, le profil du contribuable, les exonérations du contribuable, le déficit de l'impôt sur les sociétés, le crédit de l'excédent et de la cotisation minimale. Le service *CTCalculationService* utilise *CTDeclaration* comme paramètre et utilise le contrat de service *CTCalculationSLA*. Ce dernier est spécialisé en deux sous contrats, le premier pour les administrateurs et le deuxième pour les contribuables. Le temps de réponse pour un administrateur doit être inférieur à trois secondes, sinon un message et un SMS seront envoyés au responsable du service d'exploitation.

Le service de restitution de l'impôt sur les sociétés est un *SchedulingService* nommé *CTRestitutionProcess*. Il utilise quatre services métier : *CTRestitutionDemand*,

CTAdvanceRestitution, *CTRestitutionFolderVerification* et *CTRestitution*. Ils ont le même rôle que les services décrits au niveau du méta-modèle mais spécifiques à l'impôt sur les sociétés. *CTRestitutionProcess* a son propre contrat de service nommé *CTRestitutionSLA*.

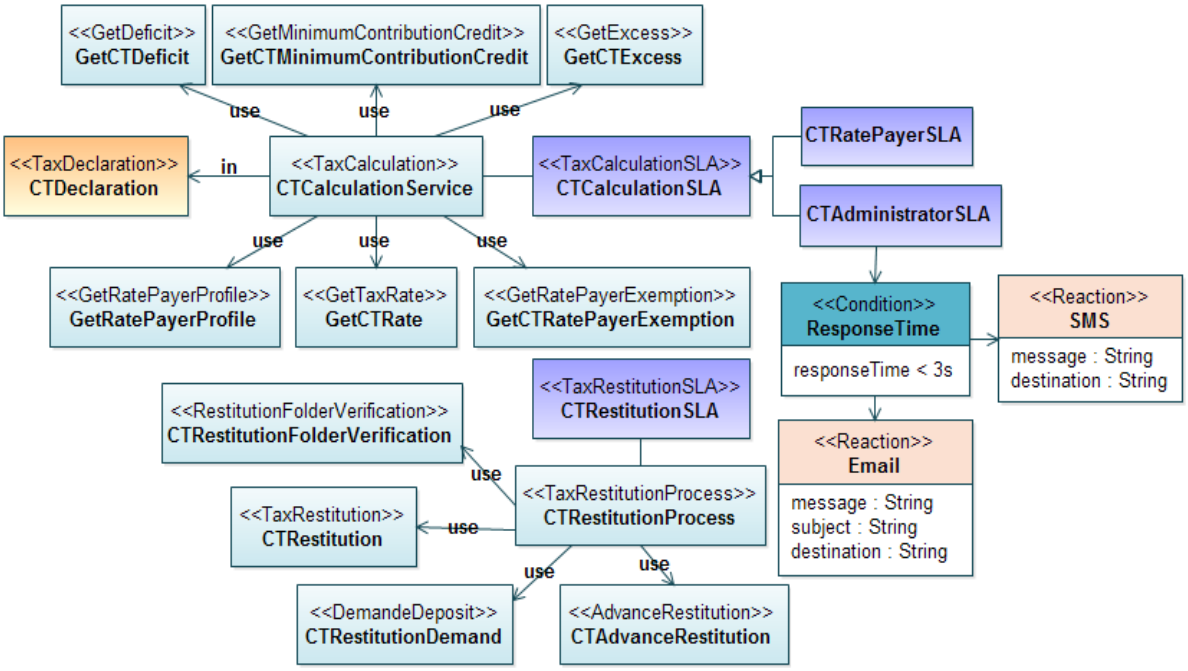


Figure 44 : Modèle des services de calcul et de restitution de l'impôt sur les sociétés

III.3.3.2. Modèle de la variabilité de services

Dans cette section, nous allons identifier les points de variation de chaque service. Chaque service possède au moins un point de variation. Pour une requête d'un service donné, un seul point de variation est utilisé.

CTCalculationService possède quatre points de variations logiques :

- *CTCalculationWithExemption* : permet de calculer l'impôt sur les sociétés en prenant en compte les exonérations du contribuable ;
- *CTCalculationWithoutExemption* : représente la logique standard de calcul de l'impôt sur les sociétés ;
- *CTCalculationNotResident* : calcule l'impôt des sociétés pour un non résident ;
- *CTAutomaticTaxation* : la logique de calcul est basée sur l'impôt sur les sociétés payé durant les quatre années précédentes.

GetCTRate possède trois points de variations logiques :

- *GetCTRateNormal* : est utilisé pour récupérer le taux de l'impôt sur les sociétés pour les contribuables résidents qui ont un secteur d'activité autre que le secteur financier (banque ou assurance) ;

- *GetCTRateFinancial* : est utilisé pour récupérer le taux de l'impôt sur les sociétés pour les contribuables résidents spécialisés dans le secteur financier ;
- *GetCTRateNotResident* : est utilisé pour les contribuables non-résidents.

CTAdvanceRestitution possède deux points de variations logiques :

- *CTAdvanceRestitutionCatA* : est utilisé pour les contribuables de catégorie A. Il permet une avance de restitution de 80% ;
- *CTAdvanceRestitutionCatB* : est utilisé pour les contribuables de catégorie B. Il permet une avance de restitution de 50%.

CTRestitutionProcess possède trois stratégies d'orchestration séquentielles : Elles se distinguent par l'utilisation du service *CTAdvanceRestitution* :

- *CategoryAStrategy* : utilise le point de variation *CTAdvanceRestitutionCatA* du service *CTAdvanceRestitution* ;
- *CategoryBStrategy* : utilise le point de variation *CTAdvanceRestitutionCatB* du service *CTAdvanceRestitution* ;
- *CategoryCStrategy* : ne permet aucune avance de restitution.

La Figure 45 rassemble l'ensemble des points de variation de l'impôt sur les sociétés.

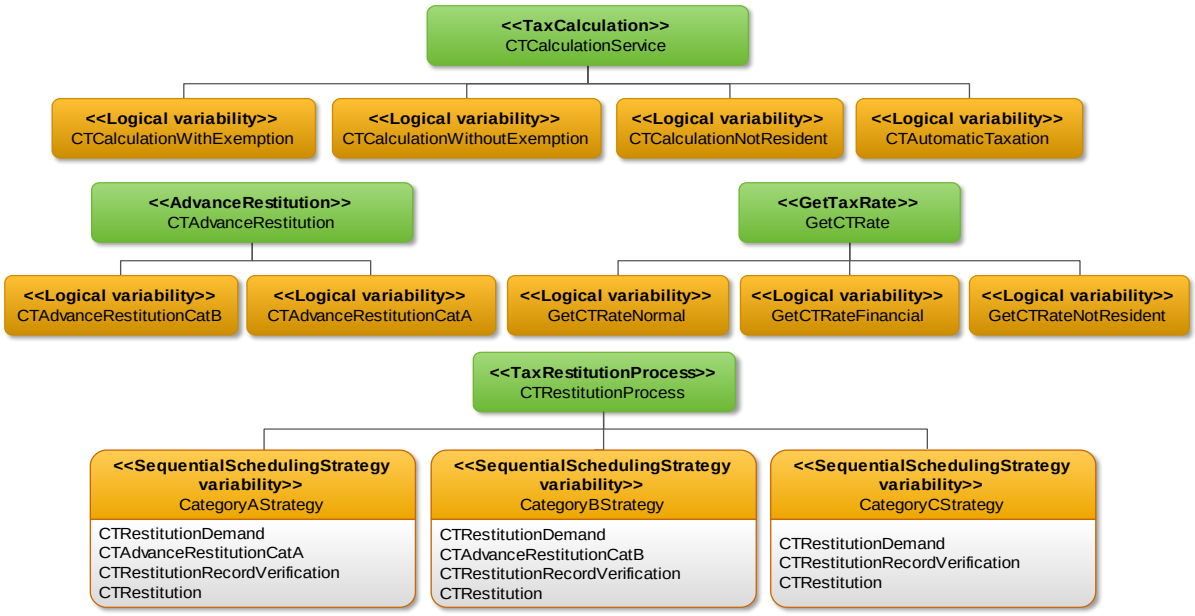


Figure 45 : Modèle de variabilité de service de l'impôt sur les sociétés

III.3.3.3. Modèle de contexte de l'impôt sur les sociétés

Notre modèle de contexte est composé d'un seul sous-contexte spécifique à l'impôt sur les sociétés (*CTContext*). Evidemment, il est possible d'ajouter d'autres sous-contextes pour d'autres impôts.

CTContext est composé de deux *ContextElements*, *RatePayer* et *CTDeclaration*. Le premier est composé de quatre *ContextParameters* :

- *Resident* : prendra la valeur *true* si le domicile fiscal du contribuable est au Maroc ;
- *Exemption* : prendra la valeur *true* si le contribuable bénéficie d'une exonération ;
- *TaxRegim* : prend la valeur *Bank* pour les organismes financiers et *Other* pour le reste ;
- *Category* : représente la catégorie du contribuable, trois valeurs sont possibles : A, B ou C.

Le dernier est composé d'un seul *ContextParameter* :

- *WithAnnex* : prendra la valeur *true* si le contribuable dépose la liasse fiscale[1].

Le modèle du contexte de l'impôt sur les sociétés est représenté au niveau de la Figure 46.

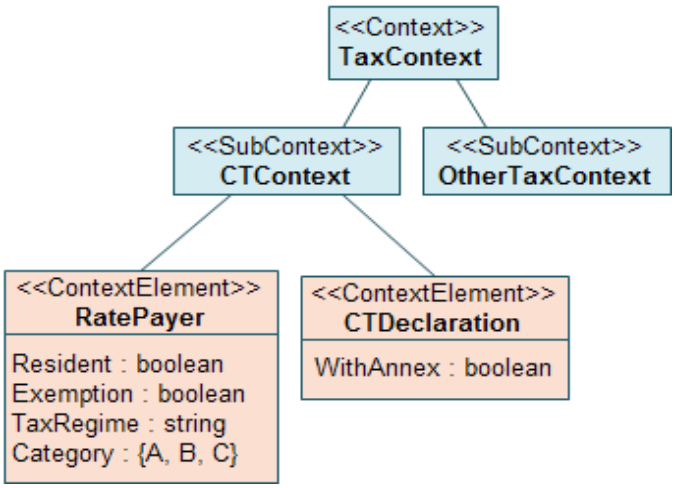


Figure 46 : Modèle du contexte de l'impôt sur les sociétés

À partir du modèle du contexte de l'impôt sur les sociétés nous avons identifié un *ServiceContextElement* pour chaque service adaptable. Il est à noter qu'un *ServiceContextElement* est composé de *ContextParameters*, de différents *ContextElements*, influençant l'adaptation du service en question. Les *ServiceContextElements* identifiés sont

[1] Liasse fiscale est un ensemble de documents fiscaux produits à la fin d'un exercice comptable. C'est une annexe de la déclaration fiscale.

(Figure 47) : *CTCalculationServiceSC*, *GetCTRateSC*, *CTAdvanceRestitutionSC* et *CTRestitutionProcessSC*. Ils sont responsables, respectivement, de la variation des services *CTCalculationService*, *GetCTRate*, *CTAdvanceRestitution* et *CTRestitutionProcess*.

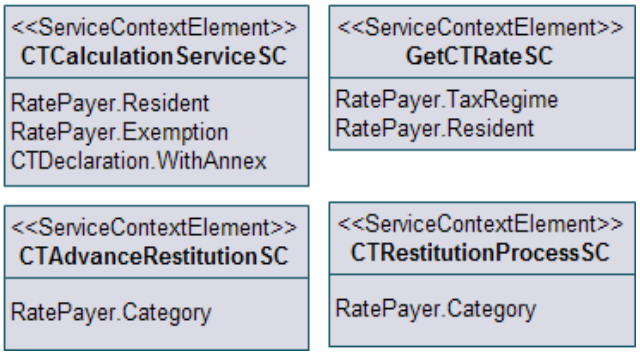


Figure 47 : Les *ServiceContextElements* des services de calcul et de restitution de l'impôt sur les sociétés

III.3.3.4. Modèle des règles d'adaptation

Au niveau de cette section nous allons modéliser les tables de décision de l'ensemble des services adaptables. Ces tables de décision représentent les règles de sélection de variabilité de service.

Les règles d'adaptation du service "CTCalculationService"

L'adaptation de ce service dépend de deux *ContextElements* : *RatePayer* et *CTDeclaration*. Ce service possède quatre points de variation : avec exonération (*with exemption*), sans exonération (*without exemption*), pour les non résident (*for non-resident*) et taxation d'office (*automatic taxation*). Les valeurs possibles des conditions sont (*Yes*, *No*). Une valeur indifférente est représenté par (-) (Tableau 6).

Tableau 6 : Règles d'adaptation du service "CTCalculationService"

CTCalculationService		R1	R2	R3	R4	ELSE
C1	An Exempt Ratepayer	Y	N	N	-	
C2	A Resident Ratepayer	-	Y	N	-	
C3	Is Declaration Deposited	Y	Y	Y	N	
C4	Is Annex deposited	N	-	-	-	
A1	CTCalculationWithExemption	X				
A2	CTCalculationWithoutExemption		X			
A3	CTCalculationNotResident			X		
A4	CTAutomaticTaxation				X	
A5	ERROR					X

Les règles d'adaptation du service "GetCTRate"

Si le contribuable est résident et le *ContextParameter TaxRegime* du ContextElement *RatePayer* est égale à *Bank* le point de variation *GetCTRateFinancial* sera utilisé. Si le contribuable n'est pas résident, le point de variation *GetCTRateNotResident* sera utilisé (Tableau 7).

Tableau 7 : Règles d'adaptation du service "GetCTRate"

GetCTRate		R1	R2	R3
C1	Ratepayer's Regime	Other	Bank	-
C2	A Resident Ratepayer	Y	Y	N
A1	GetCTRateNormal	X		
A2	GetCTRateFinancial		X	
A3	GetCTRateNotResident			X

Les règles d'adaptation du service "CTAdvanceRestitution"

Pour la restitution de l'impôt sur les sociétés, nous avons deux traitements différents, le premier pour les contribuables de catégorie « A » et le deuxième pour les contribuables de catégorie « B » (Tableau 8).

Tableau 8 : Règles d'adaptation du service "CTAdvanceRestitution"

CTAdvanceRestitution		R1	R2	ELSE
C1	Rate payer category	A	B	
A1	CTAdvanceRestitutionCatA	X		
A2	CTAdvanceRestitutionCatB		X	
A3	NoRestitution			X

Les règles d'adaptation du service "CTRestitutionProcess"

Le processus de restitution d'impôts possède trois *SchedulingStrategies*, un pour chaque catégorie de contribuable (A, B ou C) (Tableau 9).

Tableau 9 : Règles d'adaptation du service "CTRestitutionProcess"

CTRestitutionProcess		R1	R2	R3
C1	Rate payer category	A	B	C
A1	Category A strategy	X		
A2	Category B strategy		X	
A3	Category C strategy			X

III.3.3.5. Modèle des règles de calcul et de restitution de l'impôt sur les sociétés

Pour illustrer notre méta-modèle des formules de calcul d'impôts, nous allons modéliser les formules de calcul de l'impôt sur les sociétés.

Le Tableau 10 décrit les opérandes des formules de calcul de l'impôt sur les sociétés.

Tableau 10 : Description des opérandes des formules de calcul de l'impôt sur les sociétés.

Opérande	Description
E	Bénéfice de l'exercice concerné
E'	Bénéfice exonéré, proportionnel au chiffre d'affaires à l'exportation de l'exercice concerné
N	Année d'imposition
CTR	Taux de l'impôt sur les sociétés à appliquer
CTR'	Taux de l'impôt sur les sociétés à appliquer sur le bénéfice proportionnel au chiffre d'affaires à l'exportation
MCC	Crédit de la cotisation minimale des quatre dernières années de l'exercice concerné
MCR	Taux de la cotisation minimale (selon l'activité exercée, 0,5% ou 0,25% du chiffre d'affaires)
TE	Chiffre d'affaires exonéré de l'exercice concerné, proportionnel au chiffre d'affaires à l'exportation
D	Déficit des quatre dernières années de l'exercice concerné
S	Excédent de l'exercice précédent à l'exercice concerné
TT	Chiffre d'affaires total de l'exercice concerné

Formules mathématiques des points de variation du service CTcalculation:

- *CTCalculation* sans exonérations (formule classique):

$$\mathrm{CT} = \mathrm{MAX}\begin{pmatrix}(\mathrm{E} \times \mathrm{CTR}) - (\mathrm{MCC} + \mathrm{S} + \mathrm{D}),\\ \mathrm{TT} \times \mathrm{MCR}\end{pmatrix} \qquad (1)$$

- *CTCalculation* avec exonérations : Cas d'existence d'un chiffre d'affaires à l'exportation:

L'impôt sur les bénéfices proportionnels au chiffre d'affaires local, plus l'impôt sur les bénéfices proportionnels au chiffre d'affaires à l'exportation, moins la cotisation minimale, excédent et déficit des quatre exercices précédents de l'exercice concerné.

$$CT = MAX\left(\begin{matrix}\left((E - E') \times CTR\right) + (E' \times \ CTR') - (MCC + S + D), \\ TT \times MCR\end{matrix}\right) \quad (2)$$

$$E' = \frac{TE}{TT} \times E \quad (3)$$

Pour les cinq premières années où le bénéfice proportionnel au chiffre d'affaires à l'exportation est exonéré, le taux réduit de l'impôt sur les sociétés à appliquer (CTR') est égale à zéro.

- *CTCalculation* taxation d'office :

Egale à la moyenne des bénéfices des quatre dernières années de l'exercice concerné multipliée par le taux de l'impôt sur les sociétés à appliquer.

$$CT = \frac{\sum_{i=N-1}^{i=N-4} Ei}{4} \times CTR \quad (4)$$

- *CTCalculation* pour les non résident:

C'est le cas de l'imposition forfaitaire. Le chiffre d'affaires représente la base d'imposition et non pas le bénéfice (Chiffre d'affaires fois un taux libératoire de l'impôt sur les sociétés (8%)).

$$CT = TT \times CTR \quad (5)$$

III.4. Conclusion

Notre approche de développement des systèmes orientés services adaptables intitulée CADSSOMA, est divisée en deux phases : la phase de modélisation et la phase de génération du code source en utilisant l'outil support CADSSOTB. Ce chapitre s'occupe de la description de la première phase de notre approche.

La phase de modélisation est divisée en deux principales étapes :

- Etape de spécification du méta-modèle des services spécifiques au domaine (en héritant de notre méta-modèle générique des services) et du méta-modèle du métier du domaine spécifique, en l'occurrence, le domaine de calcul et restitution d'impôts.
- Etape de modélisation s'occupant de l'élaboration des cinq modèles de base de notre approche, à savoir le modèle des services spécifiques au domaine, le modèle de la variabilité des services, le modèle du contexte, le modèle des règles d'adaptation et le modèle du métier du domaine spécifique.

Afin de maitriser la complexité de la modélisation des systèmes orientés services adaptables, nous avons opté pour l'utilisation d'une multitude de modèles favorisant le faible couplage du système cible. En effet, nous avons utilisé un modèle dédié à la modélisation des règles d'adaptation qui relie le modèle de la variabilité de service et le modèle de contexte. Le premier permet de modéliser les variantes de service et le deuxième est utilisé dans la modélisation du contexte d'exécution des services adaptables. Notre méta-modèle de contexte est caractérisé par l'intégration de l'élément *ServiceContextElement*. Il regroupe l'ensemble des paramètres qui influencent l'adaptation d'un service. Une instance du *ServiceContextElement* est directement reliée à une variante de service. Le modèle des règles d'adaptation permet d'identifier la situation contextuelle nécessaire pour utiliser une variante de service.

La modélisation du métier du domaine spécifique joue un rôle important dans l'automatisation de la génération totale de la solution finale. Elle est réalisée en utilisant l'approche DSM.

La Flexibilité, la simplicité et l'extensibilité représentent les principales exigences prises en charge par les méta-modèles utilisés dans la phase de modélisation. La Figure 48 regroupe l'ensemble des méta-modèles de l'approche CADSSOMA.

Trois processus de développement sont illustrés dans le chapitre suivant : le premier concerne les systèmes orientés services, le deuxième représente notre proposition de formalisme du processus de développement des systèmes spécifiques au domaine, et le troisième est une proposition d'un processus DSM pour le développement des SOSA associé à l'approche CADSSOMA.

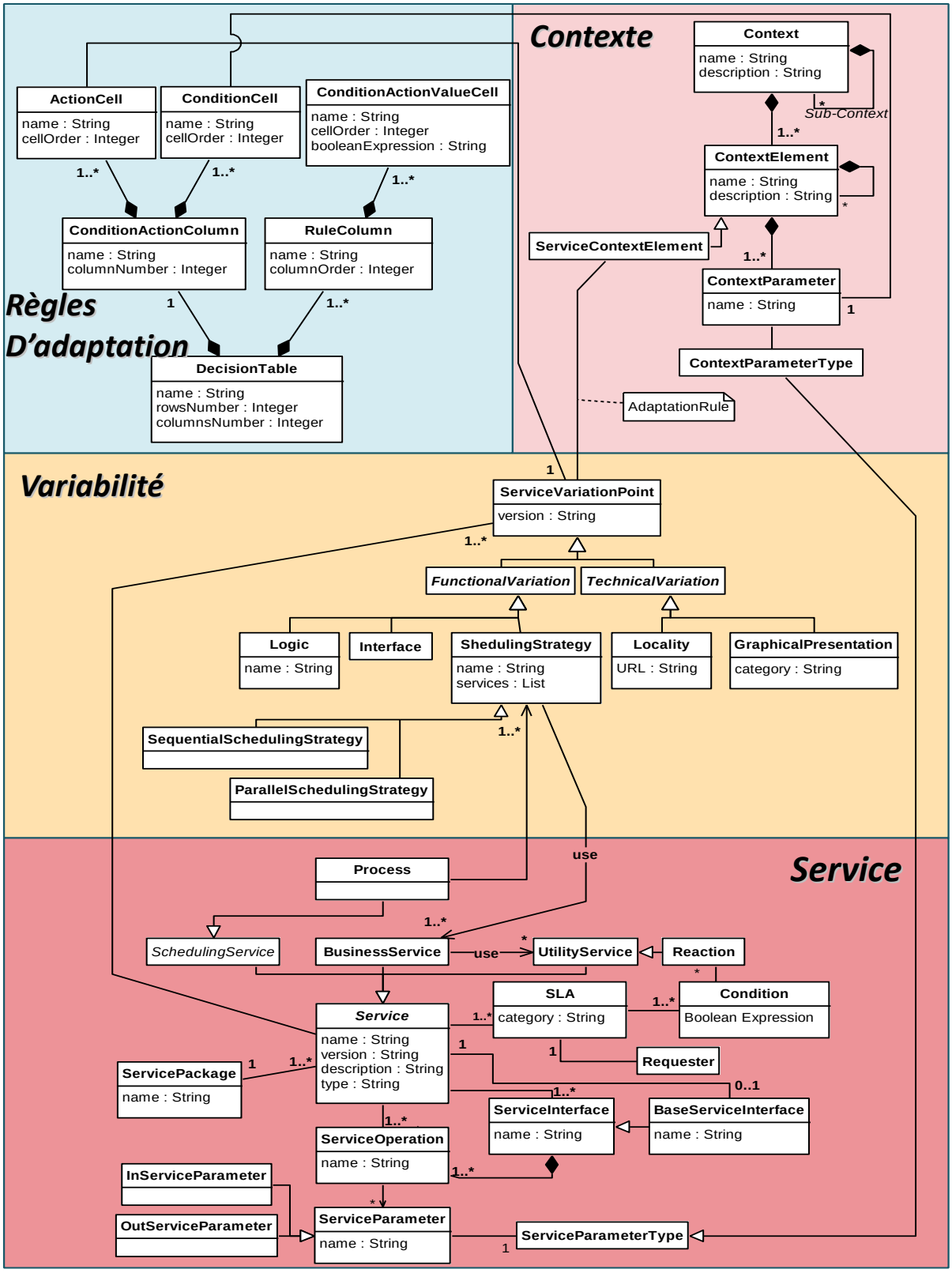


Figure 48 : Ensemble des méta-modèles de l'approche CADSSOMA

Chapitre IV

Processus DSM pour le Développement des SOSAs

IV.1. Introduction

Comme nous l'avons présenté dans le chapitre précédent, l'approche CADSSOMA permet la modélisation et la spécification des systèmes orientés service adaptables spécifiques au domaine. La modélisation de l'adaptation est basée sur le concept de variabilité de service. En effet, un modèle dédié à la variabilité de service est à produire au niveau de l'étape de conception. D'autre part, afin de profiter de l'information contextuelle, le contexte d'utilisation a également été modélisé en utilisant un modèle spécifique. De plus, et pour une meilleure séparation des préoccupations, les règles d'adaptation ont été modélisées séparément via les tables de décision. Le dernier modèle est celui des règles métier du domaine spécifique, il permet de décrire le fonctionnement interne des opérations des services.

Certes l'approche CADSSOMA (Lethrech, et al., 2015) joue un rôle important dans le développement des systèmes orientés services adaptables spécifiques au domaine, cependant il est nécessaire de définir un processus de développement pour rationaliser leur construction. Le principal but de ce chapitre est de présenter notre processus DSM pour le développement des systèmes orientés services adaptables. Un tel processus associé à l'approche CADSSOMA a pour objectif principal l'identification et la spécification des services adaptables spécifiques au domaine ainsi que leur implémentation. Pour ce faire, ce processus définit les phases, les artefacts et les activités nécessaires pour transformer la sémantique et les règles métier du domaine spécifique à des services adaptables, et ce, conformément à l'approche DSM.

Le processus de développement des applications spécifiques au domaine est très présent dans la littérature scientifique (Achilleos, 2009) (Kelly, et al., 2008a). En revanche, aucun formalisme regroupant l'ensemble des phases et artefacts de ce processus n'a été proposé. C'est pour cette raison que nous avons jugé intéressant de produire un formalisme du processus DSM permettant d'initier les développeurs à l'ingénierie des langages spécifiques au domaine.

Ce chapitre est structuré comme suit. La deuxième section présente globalement les phases, les activités et les artefacts du processus de développement d'un système orienté services. La troisième section a pour objectif d'illustrer notre formalisme du processus de développement des systèmes spécifiques au domaine. La quatrième section présente notre processus DSM pour le développement des SOSAs.

IV.2. Processus de développement des systèmes orientés services : phases, activités et artéfacts

« SOC se base principalement sur le service comme un élément fondamental pour le développement des systèmes d'information d'entreprise distribués et à large échelle. L'élément service possède des caractéristiques héritées de ses prédécesseurs tels que l'objet ou le composant, mais il ajoute d'autres caractéristiques telles que sa grosse granularité, son indépendance, son autonomie et son couplage faible. Ainsi, les approches traditionnelles pour le développement des systèmes logiciels orientés objet ou à base de composant sont inadéquates pour le développement des systèmes orientés services » (Kenzi, 2010). Pour faire face à ce problème, plusieurs processus de développements ont été proposés (Papazoglou, et al., 2006) (Arsanjani, et al., 2008) (Chang, et al., 2007a) (Kim, et al., 2005). « Globalement, ces processus définissent un ensemble de directives et de principes méthodologiques pour spécifier, construire et implémenter des systèmes orientés services en se basant sur l'élément service et sur l'architecture SOA tout en optimisant les coûts et le temps de leur développement » (Kenzi, 2010).

Le processus de développement SOA est un processus itératif et incrémental organisé en six phases (voir Figure 49) :

Phase I : Planification

L'étude de faisabilité d'une solution SOA est prise en charge par la phase de planification. Cette dernière vise essentiellement la compréhension de l'environnement métier de l'entreprise, le choix de la manière qui permettra de transformer ces exigences métiers en implémentation sous forme de services et finalement l'identification des technologies d'implémentation. Dans cette visée, il est à noter que l'intégration d'un nouveau paradigme au sein d'un système d'information existant soulève beaucoup de défis. L'un des plus importants est la réduction du coût de la transition. Quatre schémas sont possibles : i) des travaux ont proposé de créer des adaptateurs (wrappers) et de garder les composants sous-jacents intacts, et ce pour

favoriser la réutilisation des composants existants (Papazoglou, et al., 2008). Cette solution donne naissance à des services légers qui s'occupent tout simplement de faire l'orchestration des fonctionnalités déjà développées ; ii) Une autre vision propose de transformer les composants en service sachant que les systèmes orientés service utilisent le concept d'interface qui est emprunté du paradigme CBD (Kazhamiakin, et al., 2010) (Papazoglou, et al., 2008) ; iii) la troisième possibilité s'intéresse à un nouveau développement implémentations de service ; iv) le quatrième scénario concerne l'utilisation des services de d'autres fournisseurs.

Des critiques ont été adressées au premier et deuxième scénario. En effet, dans la majorité des cas, même l'exploitation des composants existants nécessite un effort de conception et de développement non négligeable. Encore, la spécification et la construction de processus métiers flexibles, adaptables en utilisant des composants monolithique d'une granularité importante n'est plus une tâche évidente.

« Durant cette phase, il faut aussi effectuer une analyse financière incluant les coûts et les bénéfices, ainsi qu'un plan de développement logiciel déterminant les tâches, les livrables ainsi qu'un programme d'action » (Kenzi, 2010).

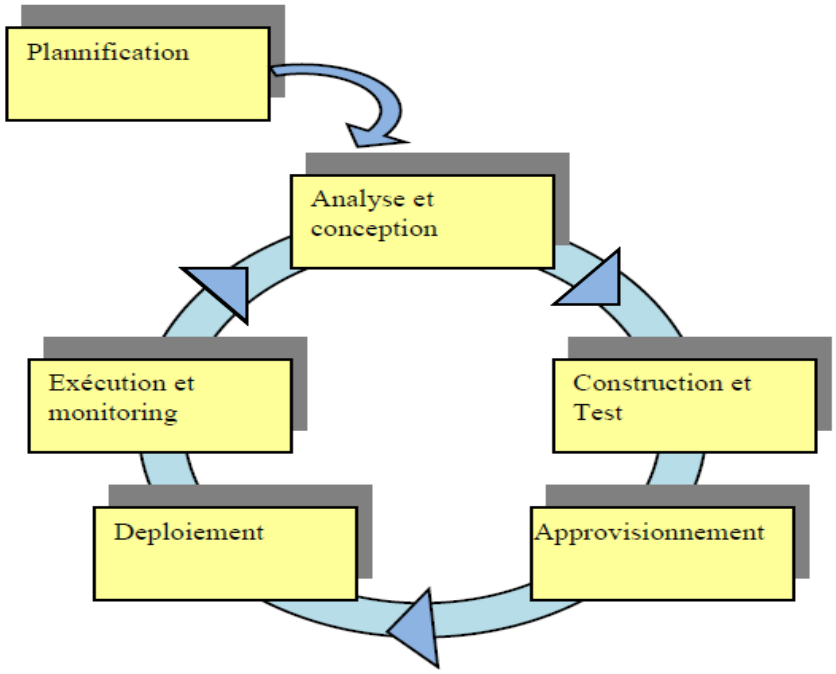


Figure 49 : Phase du processus de développement des SOS (Papazoglou, et al., 2006)

Phase II : Analyse et conception des services et des processus

Cette phase vise essentiellement l'identification et la spécification des services et des processus métiers, et ce indépendamment des implémentations technologique. Elle est organisée en deux étapes. La première s'intéresse à l'identification et la conception des services. Trois démarches classiques sont utilisées, à savoir la démarche descendante (*Top-Down*) basée sur les cas d'utilisation et les processus métiers de l'entreprise, la démarche ascendante (*Bottom-Up*) pour la réutilisation du patrimoine IT existant et la démarche hybride qui combine à la fois l'analyse des processus métiers afin de définir les services nécessaires à leurs réalisations, et l'analyse de l'existant pour déterminer les fonctions, modélisations, infrastructures existantes du système d'information (Zimmermann, et al., 2004) (Arsanjani, et al., 2008) (Arsanjani, 2004).

« La deuxième étape consiste à identifier et spécifier les processus. L'identification du processus métier consiste en l'identification des services à faire composer. La spécification du processus se base sur trois activités (i) la définition de l'objectif et de la structure du processus qui spécifie l'ordre d'exécution des services, les participants impliqués ainsi que les informations échangées entre eux, (ii) la définition des rôles que jouent les participants dans une conversation et (iii) la définition des besoins non-fonctionnels comme la sécurité des processus » (Kenzi, 2010).

Phase III : Construction et test

Cette phase est généralement composée des activités ci-dessous :

- Choix du type d'implémentation des services : RPC (Remote Procedure Call), orienté message/document, SOAP, REST, etc ;
- Définition des interfaces des services ;
- Implémentation des services (logique interne des méthodes) ;
- Publication des services dans un annuaire ;
- Développement des orchestrations de services pour former les processus métiers.

Phase IV : Approvisionnement

Cette phase, est composée de plusieurs activités qui s'intéressent au contrôle actif du comportement des services (facturation, certification des services, gouvernance, etc.).

Phase V : Déploiement

« Dans le cadre de cette phase, trois activités ont été définies :

- La publication des interfaces de services dans un annuaire de services ;
- Le déploiement des services et des processus métiers ;
- La publication des détails des implémentations des services en identifiant les informations nécessaires à leurs consommations (protocoles utilisé, point d'entrée du service, etc.) » (Kenzi, 2010).

Phase VI : Exécution

« Dans le cadre de cette phase, les services ainsi que les processus métiers sont déployés et opérationnels. Un client de service peut alors invoquer les opérations du service en se référant à la définition de son interface » (Kenzi, 2010).

IV.3. Développement des systèmes spécifiques au domaine : Equipe, Processus et cycle de vie

Au niveau de cette section, nous allons donner un aperçu détaillé du processus de développement des systèmes spécifiques au domaine en explicitant les phases, les activités, les artefacts et les rôles du processus. Au cours de notre travail de recherche nous avons constaté un manque de formalisme regroupant l'ensemble des étapes d'un processus de développement d'une solution DSM. Ce constat, confirmé aussi par Mernik (Mernik, et al., 2005), nous a motivé à proposer un modèle du processus de développement des applications spécifiques au domaine (Figure 50), qui est essentiellement basé sur les cas pratiques étudiés (Kelly, et al., 2008a) (Safa, 2006) (Rahien, 2010) (Achilleos, 2009) (Gronback, 2009) (Mernik, et al., 2005) et aussi sur notre expérience dans le cadre de la mise en place de l'outil support CADSSOTB. Cette section est structurée comme suit. La première sous-section met en exergue les spécificités de l'équipe DSM par rapport à une équipe de développement traditionnelle. La sous-section suivante illustre notre formalisme du processus de

développement des systèmes spécifiques au domaine. Et enfin, la dernière sous-section met en lumière le cycle de vie d'une solution DSM.

IV.3.1. L'Equipe DSM

Avant de commencer le développement d'une solution DSM, il faut au préalable constituer l'équipe DSM. Dans ce sens, l'adoption de l'approche DSM requiert une organisation de l'équipe de développement qui est totalement différente de l'organisation d'une équipe classique. L'approche DSM nécessite des développeurs expérimentés pour créer les trois éléments de base d'une solution DSM, à savoir le langage de modélisation spécifique au domaine, le générateur de code spécifique au domaine et un framework du domaine. Ces derniers forment ensemble un environnement de développement pour le reste de l'équipe. Le travail de tous les développeurs sera encadré et guidé par l'environnement de développement réalisé par l'équipe DSM. En effet, ils ne font que créer des modèles en utilisant le langage spécifique, ensuite l'application sera automatiquement et totalement générée. Le code généré ne devrait pas être modifié et ne nécessiterait même pas d'être visualisé.

L'intégration des utilisateurs finaux (équipe pilote ou experts métier) dès les premières étapes de la réalisation de la solution DSM est un choix stratégique pour sa réussite. L'intervention de l'équipe pilote réside dans la spécification des concepts du domaine, le test du prototype de la solution DSM, la spécification des règles du domaine et la validation du langage de modélisation en s'assurant que c'est un bon moyen pour construire des applications du domaine. Il est certain que l'équipe de la solution DSM (*DSM Team*) doit encadrer l'équipe pilote (*Pilote team*) dans l'utilisation de la solution DSM, aussi, elle doit montrer une grande réactivité dans la correction et à l'amélioration des diverses parties de la solution pour éviter toute interruption du travail de l'équipe pilote.

IV.3.2. Processus de développement des systèmes spécifiques au domaine

Le processus de développement d'une solution DSM peut être décomposé en six phases (Figure 50).

Phase I : Analyse du domaine spécifique cible

Cette phase est divisée en deux étapes :

1 – La validation de l'applicabilité de l'approche DSM au niveau du domaine cible :

L'approche DSM n'est pas une solution générique à tous les problèmes techniques de l'ingénierie du logiciel (Mernik, et al., 2005). Pour cela, une étape de validation de l'applicabilité de l'approche DSM pour le domaine spécifique cible est nécessaire. Pour ce faire, nous proposons l'utilisation du questionnaire proposé par Laurent Safa et Kelly (Kelly, et al., 2008a) (Annexe D). Il est constitué d'un ensemble de questions permettant de faciliter la prise de décision. La notation des réponses du questionnaire permet de faciliter la prise de décision concernant l'adoption de l'approche DSM.

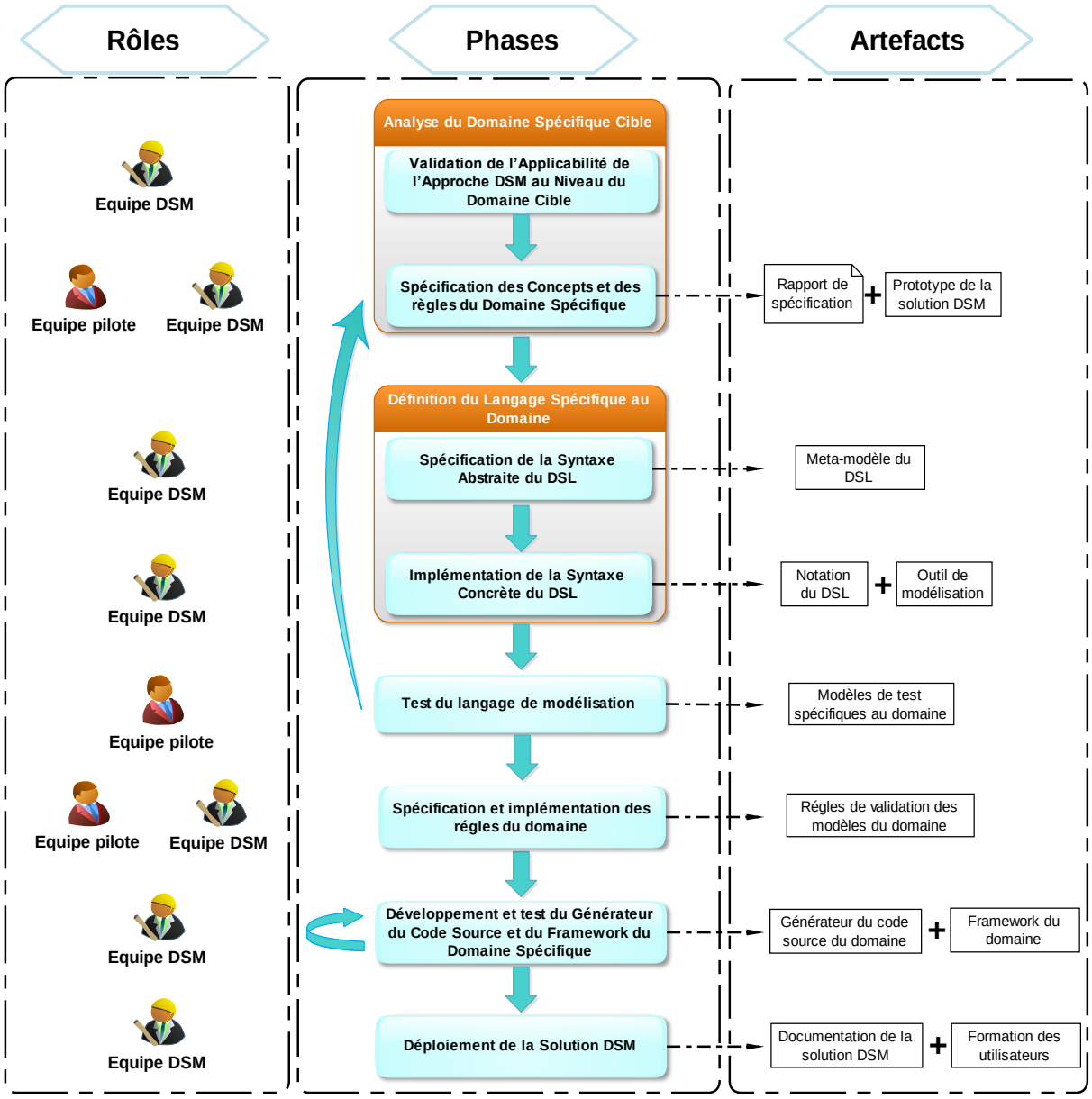


Figure 50 : Processus de développement des applications spécifiques au domaine

Deux cas de figure se présentent :

a. L'organisme visant l'utilisation de l'approche DSM est spécialisé dans un seul métier. Dans ce cas le proverbe American *"Do, or do not. There is no try*" sera appliqué.
b. L'organisme visant l'utilisation de l'approche DSM s'intéresse à plusieurs domaines. Dans ce cas il est préférable, en premier lieu, de tester l'approche DSM dans un domaine restreint, permettant ainsi de valider l'utilisation de l'approche DSM et aussi d'aborder les autres domaines avec plus d'expérience.

2 – Spécification des concepts du domaine spécifique :

L'objectif principal de cette étape est la détermination du périmètre fonctionnel, la décomposition des informations, ainsi que la collecte des éléments clés du domaine. Ces derniers représenteront le noyau du langage de modélisation du domaine spécifique. Laurent Safa et Kelly (Kelly, et al., 2008a) proposent d'organiser cette étape sous forme d'un atelier détaillé dans la Figure 98 de l'Annexe D. En plus des documents de spécification, le principal artefact de ce workshop est sous forme d'un prototype de la solution DSM. Il doit inclure les principaux éléments du domaine et aussi doit être complet, c'est-à-dire, comportant le langage de modélisation, l'outil de modélisation et le générateur du code source. Ce prototype a pour objectif de mettre en exergue les avantages de l'approche DSM, et aussi d'éliminer les préjudices préconçus à l'instar de « la modélisation est une perte de temps », « le code généré est incomplet », etc. Le prototype de la solution DSM permet aussi d'avoir les retours des experts du domaine permettant ainsi de mieux appréhender leurs exigences. Cette étape doit être accomplie en étroite collaboration entre l'équipe DSM et l'équipe pilote.

À ce stade la préoccupation majeure des créateurs de la solution DSM doit être la capacité du langage de modélisation à capturer les informations nécessaires du domaine spécifique.

Phase II : Définition du langage spécifique au domaine

Le langage de modélisation est l'élément le plus visible dans une solution DSM. En réalité, c'est en utilisant la syntaxe concrète du langage que les développeurs créent leurs modèles. Le générateur de code source ainsi que le framework sont invisibles aux utilisateurs de la solution DSM. Le langage de modélisation, ou plus précisément, la syntaxe concrète, est un ensemble d'éléments graphique (ou textuels) avec lesquels sont représentés les modèles du domaine spécifique. Toute modification du langage aura un impact sur l'ensemble des

modèles déjà produits ainsi que sur l'expérience accumulées des développeurs en utilisant la solution DSM.

Cette phase est divisée en deux étapes :

1 – Spécification de la syntaxe abstraite du DSL :

C'est le volet qui nécessite le plus de concentration puisqu'il traite la modélisation des concepts du domaine ainsi que leurs interrelations. En résumé, cette étape s'occupe de la modélisation de la syntaxe abstraite du langage. La première action consiste en la définition des concepts. Ils sont à enrichir par les propriétés représentant les informations pertinentes qui doivent être modélisées et stockées.

De plus, la liaison entre les différentes occurrences des concepts du domaine doivent être liées, est à étudier, et ce pour former un produit complet. Dans ce sens, et comme le nombre de concepts et de relations peut augmenter suite à l'apparition de nouveaux besoins, il est préférable d'organiser le langage de modélisation en plusieurs sous langages. Chaque sous langage doit être fortement cohésif et présente un faible couplage avec le reste des sous langages.

Un DSL est sensé modéliser plusieurs variantes d'un produit spécifique. Par conséquent, il faut, dès les premières étapes de la création de la solution DSM, déterminer les différences à modéliser. Le principal artefact de cette étape est la syntaxe abstraite du langage.

2 - Implémentation de la Syntaxe concrète du DSL :

La notation est généralement graphique ou textuelle. Bien que 75% des personnes préfèrent une représentation visuelle au lieu d'une représentation textuelle (Kelly, et al., 2009b), le choix du type de représentation ne doit pas être basé uniquement sur des préjudices. Il y a aussi d'autres formes de représentation comme les matrices, les arbres etc (MataCase; Kelly, S., 2012). Le choix de la notation du langage dépend des utilisateurs finaux, de la structure de données et de la façon avec laquelle les utilisateurs vont exploiter les données (Kelly, et al., 2008a). Faire le mauvais choix peut considérablement augmenter le coût de la création, la lecture et la maintenance des modèles.

Il y a quelques années le coût élevé de la création des environnements de développement graphiques représentait l'obstacle majeur de l'utilisation d'une notation visuelle (Iseger, 2010). Actuellement, avec le développement des progiciels de méta-modélisation, commerciaux comme *MetaEdit+* ou *Actifsource*, ou open source comme *GMF* et *Sirius*, le choix d'une représentation visuelle est devenu une option très intéressante.

Puisque nous avons un cerveau visuel (MataCase; Kelly, S., 2012), les symboles des différents concepts doivent être facilement distinguables, avec plus de visibilité pour les principaux concepts. Cela est possible, par exemple, en utilisant des icônes spécifiques ou via des variations de la forme, du type de ligne, des couleurs de remplissage, de la largeur des lignes, des formes utilisées, etc.

En général, à ce stade, les créateurs des solutions DSM ne s'intéressent pas trop à l'aspect esthétique de la représentation graphique. Cela est justifiable qu'après la stabilisation de l'ensemble des concepts du domaine spécifique. Cependant, deux choses sont à éviter, même à ce stade précoce : i) Il ne faut pas distinguer tous les symboles des concepts uniquement par la couleur de remplissage : le cerveau perçoit ces éléments comme représentant des variations mineures de la même chose, ii) De même, il est rarement nécessaire d'inclure le nom du concept dans le cadre du symbole : DSM exploite la capacité du cerveau à reconnaître les choses comme dans le monde réel. Les principaux artefacts de cette étape sont la notation du DSL et l'outil de modélisation.

Phase III : Test du langage de modélisation

Le composant clef d'une solution DSM est le langage de modélisation. La meilleure façon de le qualifier est de vérifier qu'il capture l'ensemble des éléments du domaine. Pour ce faire, il faut intégrer les utilisateurs finaux (équipe pilote) dès les premières étapes du processus de développement de la solution DSM. En effet, l'intervention de l'équipe pilote est essentielle dans le test des différentes itérations produites du langage spécifique. Retenons que le développement du générateur du code source ne doit commencer qu'après la stabilisation du langage de modélisation. Dans ce sens, la flèche de retour de la phase de test du langage de modélisation vers la phase de spécification des concepts du domaine spécifique permet de faire des itérations *[phase 1 → phase 2 → phase 3]* jusqu'à la stabilisation du langage DSM.

Phase IV : Spécification et implémentation des règles du domaine

L'objectif principal de l'implémentation des règles du domaine est de tracer des garde-fous aux utilisateurs de la solution DSM. Ces garde-fous permettent la prévention contre la production de modèles avec des incohérences considérées interdites ou non souhaitables pour le domaine cible. On peut remarquer que l'implémentation des règles du domaine n'est préférable qu'après avoir disposé d'une version stable du langage de modélisation, voir qu'après avoir terminé le développement du générateur du code source. Cela permettra aussi d'interdire aux utilisateurs de produire des structures de modèles non supportés par le

générateur. Au niveau de l'outil de modélisation, le contrôle de la cohérence peut être tout au long de la modélisation comme il peut être lancé qu'après avoir terminé la création du modèle.

Phase V : Développement et test du générateur du code source et du Framework

Comme expliqué précédemment au niveau de la section II.2.2 l'approche DSM vise une génération totale du code source à partir des modèles spécifiques au domaine. Un développement sans risque du générateur du code source doit se faire qu'après la stabilisation du langage de modélisation. Ce dernier peut faire l'objet de changements mineurs, par exemple ajout d'une propriété, mais il ne devrait pas y avoir de modifications majeures touchant le cœur des concepts du domaine. En d'autres termes, procéder au développement précoce du générateur de code source représentera au mieux un gaspillage important d'effort.

En pratique le générateur de code source ainsi que le framework sont développés en parallèle. La bonne pratique dicte qu'après la réalisation de la première version du générateur du code source, il faut chercher les blocs de code source similaire généré. Ces blocs doivent être regroupés sous forme d'une fonction au niveau du framework, et puis il faut remplacer la partie responsable de la génération du code répétitif au niveau du générateur pour qu'elle ne produise qu'un simple appel à cette fonction. Notez que ceci est différent de la factorisation du code qu'il faut faire au niveau du générateur lui-même, où les blocs répétés des commandes du générateur seront factorisés sous forme de sous générateurs ou à mettre aussi dans le framework.

Ce travail est à effectuer en dernière lieu lors de la création d'une solution DSM, car, même si on déplace des blocs entre le générateur et le framework, l'utilisation des modèles existants pour générer du code source restera toujours valable et transparente pour les utilisateurs finaux. Le développement du générateur et du framework suit un cycle de développement itératif et incrémental, il est fait sur plusieurs itérations. Les principaux artefacts de cette phase sont le générateur du code source et le framework du domaine spécifique.

Phase VI : Déploiement de la solution DSM

Il est largement reconnu que l'introduction de modifications dans une organisation de travail entraîne des résistances, et ce, quelle que soit la nature de ces changements (BAREIL, 2004). L'adoption d'une solution DSM par un organisme est comparable à un passage de LaTex au World ou d'un simple éditeur HTML au FrontPage. Le choix d'une solution DSM

nécessite un effort considérable au niveau de la conduite de changement. Cette dernière vise principalement à faciliter l'acceptation des changements induits par la mise en œuvre d'un nouveau projet et à réduire les facteurs de rejet. Safa (Safa, 2006) considère que la plupart des défis de déploiement d'une solution DSM ne sont pas techniques mais humains et organisationnels. Cela a été aussi confirmé par « Marjan Mernik » lors de la conférence « Object-oriented programming systems languages and applications » (Gray, et al., 2008). En effet, cinq des dix raisons pour lesquelles les DSLs ne sont pas largement utilisés dans le monde industriel sont des raisons d'ordre psychologique.

N'oublions pas qu'il est fortement recommandé de documenter le langage du domaine spécifique en utilisant des exemples pratiques. Il faut aussi veiller à ce que la documentation soit toujours mise à jour avec l'évolution de la solution. En plus, la formation des utilisateurs finaux sur la solution DSM est une condition sine-qua-non pour un déploiement fluide.

Les utilisateurs finaux de la solution DSM doivent faire un effort d'apprentissage de la nouvelle solution. La tâche est rendue plus facile puisque le domaine leur sera familier, et les concepts du langage de modélisation sont directement mappés au domaine. Il est aussi à souligner qu'un bon outil de développement offre une documentation pour chaque concept du langage au sein de l'outil lui-même.

IV.3.3. Cycle de vie d'une solution spécifique au domaine

La Figure 51 résume le cycle de vie d'une solution spécifique au domaine. Ce schéma est une adaptation du schéma proposé par (Safa, 2006). L'effort principal de mise en place d'une solution DSM est déployé lors de la première itération ***(A)***. Au fil de l'évolution des exigences métiers du domaine spécifique, des adaptations doivent être introduite aux différents éléments de la solution DSM ***(B)*** (syntaxe du langage, outils de modélisation, générateur, etc.). L'utilisation principale d'une solution DSM est représentée par l'activité de modélisation spécifique au domaine ***(C)***.

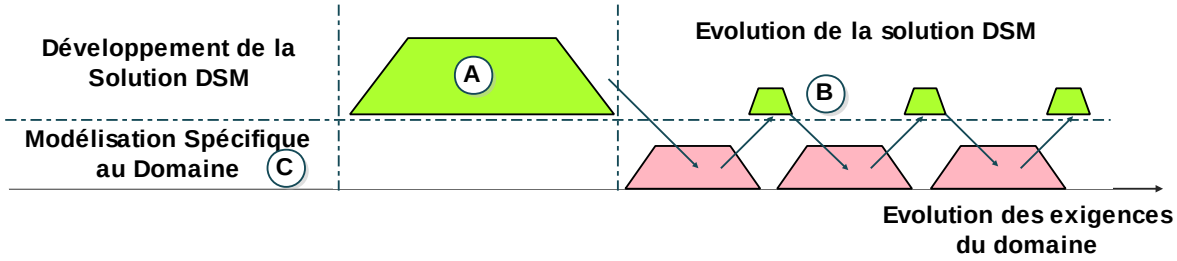


Figure 51 : Cycle de vie d'une solution spécifique au domaine

Bien que la gestion des versions[1] ait un rôle moins important dans le développement basé sur une solution spécifique au domaine en le comparant avec le développement dirigé par le code, son utilisation est recommandée, surtout s'il y aurait des évolutions de la solution spécifique. Il faut lier chaque modèle avec sa version du méta-modèle, et pour chaque itération il faut déterminer le méta-modèle, le générateur et le framework compatibles. En d'autres termes, la version d'un élément d'une solution DSM doit correspondre à la même version des autres éléments. Pour pallier ces préoccupations, il est fortement recommandé d'utiliser un outil de contrôle de versions pour la solution DSM ainsi que pour les modèles. En outre, chaque modèle produit doit contenir l'information indiquant la version du méta-modèle utilisé.

IV.4. Processus DSM pour le développement des SOSAs

Comme nous l'avons présenté dans le chapitre II, l'approche CADSSOMA vise la modélisation et la spécification des systèmes orientés services adaptables à base de DSLs.

Dans l'objectif de rationaliser le développement des systèmes orientés services adaptables et spécifiques au domaine et en se basant sur l'approche CADSSOMA, nous avons jugé nécessaire de définir un processus de développement permettant de guider la mise en œuvre de ce type de systèmes à partir des exigences métiers. Un tel processus définit les phases, les activités et les artefacts facilitant l'identification, la spécification et l'implémentation de la solution finale.

La Figure 52 illustre les principales phases de notre processus de développement des SOSAs spécifiques au domaine (Lethrech, et al., 2015).

Les sections suivantes décrivent les différentes phases de notre processus de développement.

Phase 1: Analyse du domaine spécifique cible

A l'instar du processus de développement DSM, cette phase est composée de deux étapes : 1- La validation de l'applicabilité de l'approche DSM au niveau du domaine cible, et 2- Spécification des concepts du domaine spécifique.

La première étape est identique à celle du processus de développement des systèmes spécifiques au domaine (voir la section IV.3.2).

[1] *Versionning* dans le vocabulaire anglo-saxon.

La principale activité de la deuxième étape est l'élaboration des modèles des cas d'utilisation d'UML. Ces derniers permettent de capturer et spécifier les acteurs du système ainsi que leurs besoins fonctionnels.

« Notre approche adopte les diagrammes des cas d'utilisations pour plusieurs raisons : i) ils jouent un rôle très important dans l'identification des besoins des utilisateurs en décrivant de manière exhaustive les exigences fonctionnelles du système ; ii) la majorité des analystes/concepteurs sont familiers avec l'utilisation des cas d'utilisations ; iii) aussi, les cas d'utilisations identifiés dans cette étape deviendront potentiellement des services, et les relations entre les cas d'utilisations sont un excellent indicateur de la collaboration et la composition des futurs services (Kim, et al., 2013) (Ibrahim, et al., 2006) » (Kenzi, 2010).

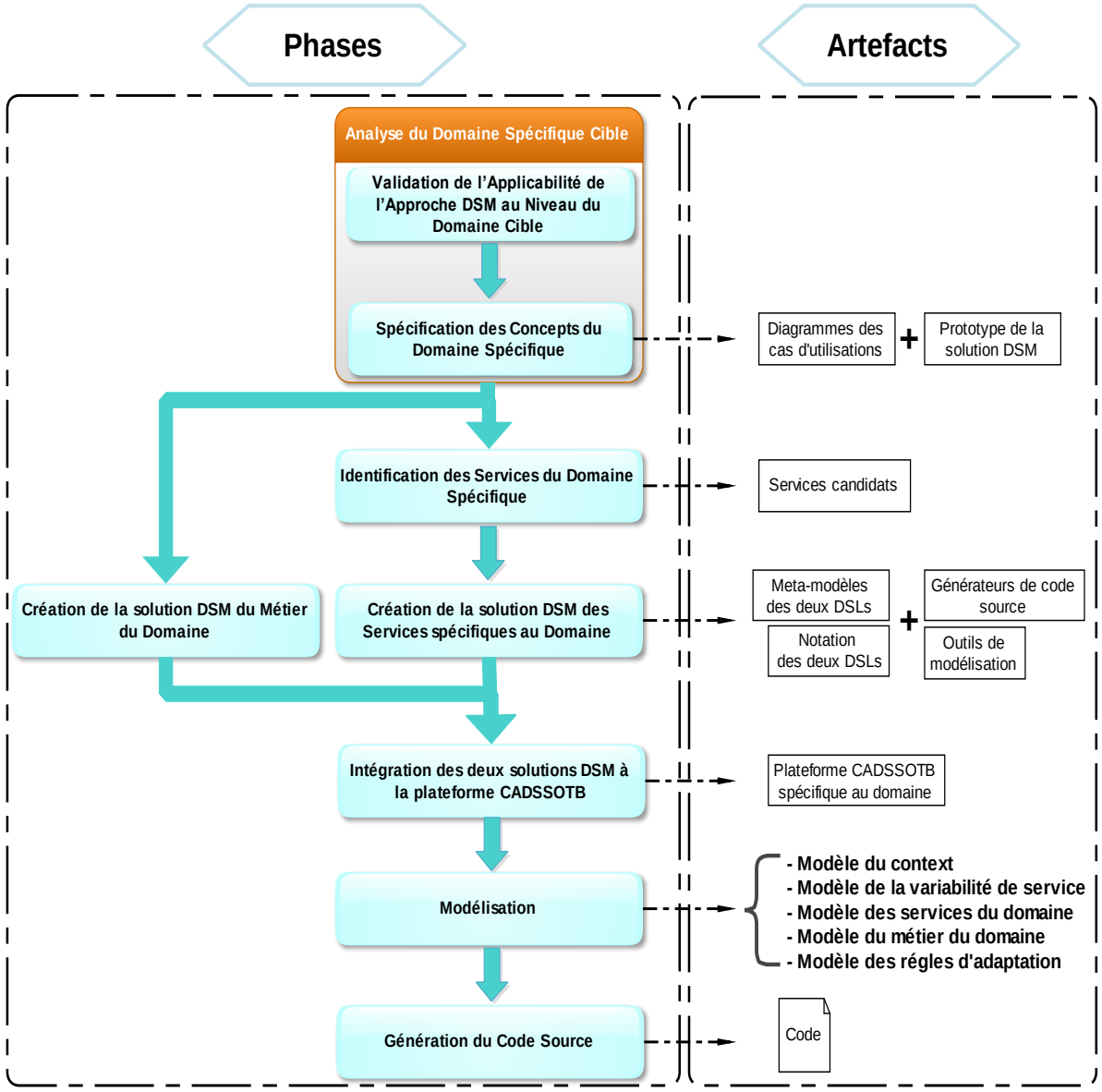


Figure 52 : Processus DSM pour le développement des SOSAs

L'artefact caractérisant cette phase est un diagramme de cas d'utilisations spécifiant les différents acteurs du système. La Figure 53 présente les cas d'utilisation du domaine de calcul et restitution d'impôts. Deux principaux cas d'utilisation sont identifiés : « Calculer l'impôt » et « Restituer l'impôt ». Ces derniers seront décomposés dans l'étape suivante.

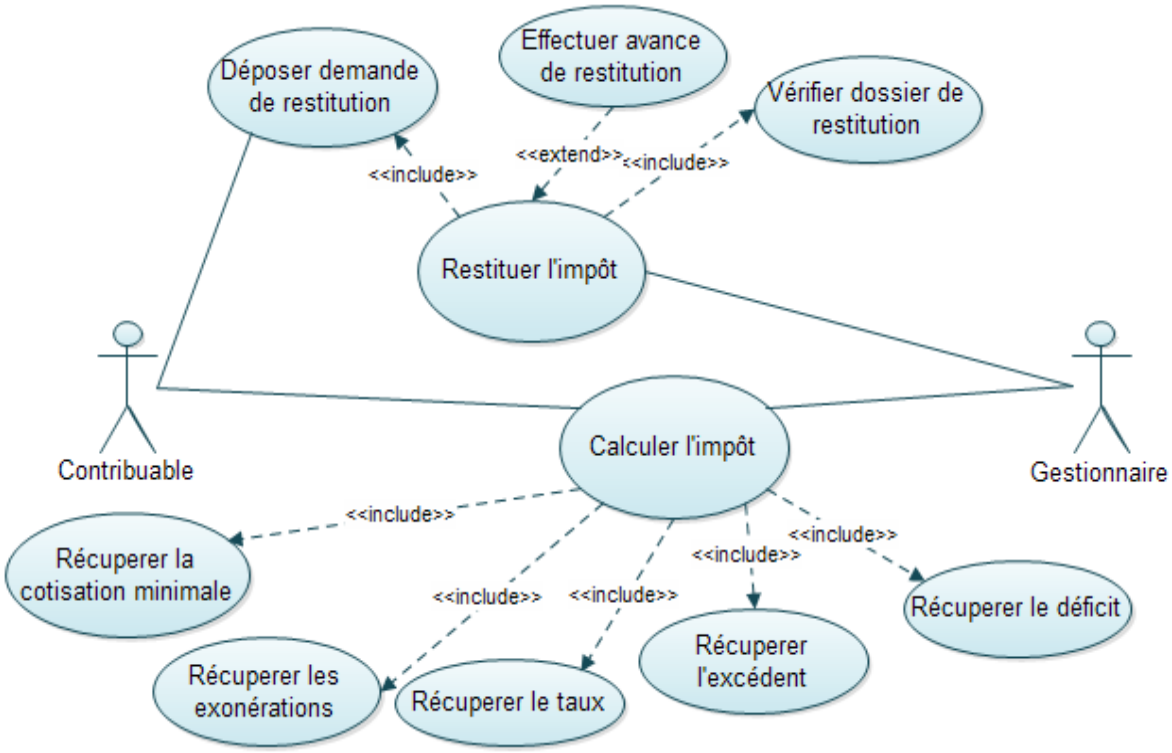


Figure 53 : Cas d'utilisation du domaine spécifique : calcul et restitution d'impôts

Phase 2: Identification des services du domaine spécifique

« La phase d'identification des services du domaine a pour objectif principal la transformation des exigences métiers capturées à travers les modèles des cas d'utilisation en une collection de services. Il s'agit principalement d'identifier les services candidats à partir des modèles des cas d'utilisation » (Kenzi, 2010). Dans ce sens, Kim et al. (Kim, et al., 2007) ont défini plusieurs règles de *Refactoring* permettant l'identification des services à partir des modèles de cas d'utilisation. « Chaque cas d'utilisation est décomposé en termes d'un ensemble de tâches permettant la réalisation des besoins des acteurs qui sont en interaction avec ce cas d'utilisation. Chaque tâche est définie comme étant une unité de travail indivisible apportant une valeur ajoutée à l'utilisateur dans le contexte d'un processus métier » (Kenzi, 2010).

Kim et al. (Kim, et al., 2007) identifient cinq règles de *Refactoring* :

- La règle de *Refactoring* par décomposition : « est appliquée lorsque nous avons des cas d'utilisation compliqués. Dans cette situation, il est nécessaire de faire décomposer ce cas d'utilisation en des sous-cas d'utilisation plus simples. Ceci est effectué en utilisant l'ensemble des tâches associées à chaque cas d'utilisation. En effet, pour chaque cas d'utilisation, nous élaborons les tâches permettant de réaliser le cas d'utilisation. A partir de ces tâches, nous pouvons déduire des fonctionnalités indépendantes et par conséquent, définir des sous-ensembles de tâches qui correspondent à des nouveaux cas d'utilisations » (Kenzi, 2010).
- Le *Refactoring* par équivalence : « s'applique à des cas d'utilisations ayant des tâches équivalentes. Dans ce cas, nous pouvons conclure que les deux cas d'utilisations ont le même comportement et on peut les traiter comme un seul cas d'utilisation correspondant à un seul service » (Kenzi, 2010).
- Le *Refactoring* par généralisation : « peut s'appliquer lorsque nous avons des cas d'utilisation qui partagent des tâches communes. Dans ce cas, nous créons un cas d'utilisation de base contenant les tâches communes ainsi que de nouveaux cas d'utilisation étendant le cas d'utilisation de base » (Kenzi, 2010).
- Le *Refactoring* par fusion : « est appliqué lorsque nous avons des cas d'utilisation de granularité fine qui modélisent des fonctionnalités sémantiquement reliées. Dans ce cas, nous fusionnons les deux cas d'utilisation pour avoir un cas d'utilisation plus consistant » (Kenzi, 2010).
- Le *Refactoring* par suppression : « est défini lorsque nous avons un cas d'utilisation qui n'a aucune relation que ce soit avec les autres cas d'utilisation ou avec les acteurs du système. Dans cette situation, le cas d'utilisation est redondant et par conséquent on peut le supprimer du système » (Kenzi, 2010).

Le cas d'utilisation « Calculer l'impôt » est décomposé en cinq cas d'utilisations : *Récupérer le déficit, Récupérer l'excédent, Récupérer la cotisation minimale, Récupérer le taux* et *Récupérer les exonérations*. Chaque cas d'utilisation est décrit par des tâches indépendantes. En conséquence, chaque cas d'utilisation représente un service.

Le cas d'utilisation « Restituer l'impôt » peut être décomposé en trois sous cas d'utilisation : *déposer demande de restitution, effectuer avance de restitution et vérifier dossier de restitution.* Chaque cas d'utilisation est décrit par des tâches indépendantes. Ainsi, chaque cas d'utilisation représente un service.

L'ensemble des services candidats est représenté au niveau de la Figure 54.

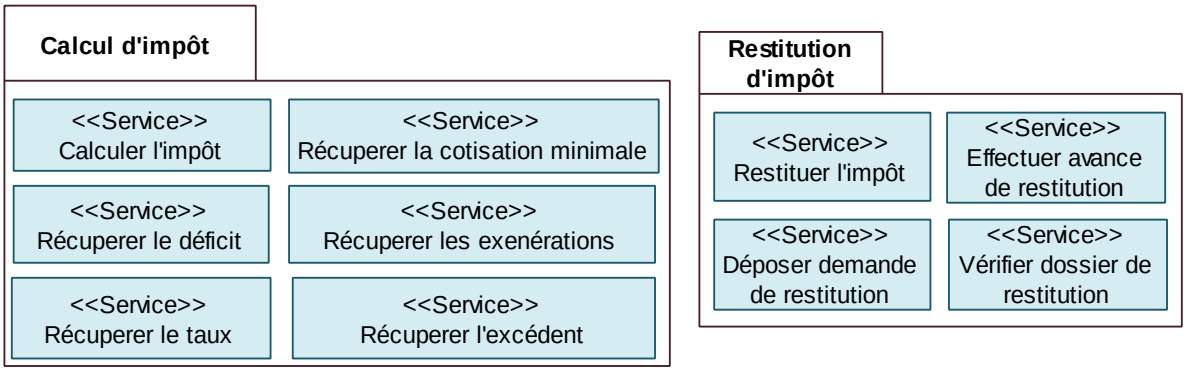


Figure 54 : Services candidats du domaine de calcul et restitution d'impôts

Phase 3 : Création des solutions DSM : métier et services spécifiques au domaine

Cette phase est divisée en deux étapes : i) Création de la solution DSM du métier du domaine ; et ii) création de la solution DSM des services spécifiques au domaine.

La première étape vise principalement la création d'une solution DSM du métier du domaine cible. Les artefacts de cette étape sont : la syntaxe abstraite du DSL, la notation du DSL, l'outil de modélisation et le générateur du code source. Ces derniers permettent la modélisation du métier du domaine spécifique et la génération du code source de la logique des opérations des services. Cette étape joue un rôle primordial dans la génération totale du code source. Les éléments du DSL de calcul et de restitution d'impôts sont représentés au niveau de la Figure 43.

La deuxième étape s'occupe de la création d'une solution DSM des services spécifiques au domaine. Les artefacts de cette étape sont : la syntaxe abstraite du DSL, la notation du DSL, l'outil de modélisation et le générateur du code source. La syntaxe abstraite est une spécialisation de notre méta-modèle générique des services représenté dans la Figure 36. Les éléments du DSL de modélisation des services de calcul et de restitution d'impôts sont représentés au niveau de la Figure 42.

Phase 4 : Intégration des deux solutions DSM à la plateforme CADSSOTB

L'objectif principal de cette phase est l'intégration des éléments des deux solutions DSM de la phase précédente au niveau de la plateforme CADSSOTB. Cette dernière comporte des solutions DSM pour modéliser le contexte, la variabilité de service et les règles d'adaptation. Il est à noter que la version actuelle de la plateforme CADSSOTB se base essentiellement sur les technologies de modélisation d'Eclipse (Gronback, 2009). Ce dernier est une plateforme de développement de logiciels conçus pour la construction des environnements de développement intégrés. Deux cas de figure se présentent :

- Si les technologies Eclipse qui sont utilisées pour créer les deux solutions DSM de la phase 3, alors dans ce cas l'intégration est plus simple. Ceci, grâce à l'architecture extensible d'Eclipse, basée sur le concept de plug-in. De même, la nature open source de la plateforme Eclipse permet d'étendre ou de modifier, au besoin, les Environnements de Développement Spécifiques au Domaine (EDSD) développés.
- Sinon, l'utilisateur doit reconstruire les trois solutions DSM de la plateforme CADSSOTB, à savoir celle du contexte, de la variabilité de service et des règles d'adaptation. Et ce, en se basant sur la syntaxe abstraite du contexte, de la variabilité de service et des règles d'adaptation, représentés respectivement au niveau des figures Figure 38, Figure 37 et Figure 40.

Une plateforme CADSSOTB complète, c'est-à-dire prenant en charge les cinq modèles de l'approche CADSSOMA, est le principal artefact de cette phase.

Phase 5 : Modélisation : Service, Variabilité de service, Contexte, Règles d'adaptation et Métier

L'activité principale de cette phase est la modélisation des cinq modèles de l'approche CADSSOMA, le modèle des services (III.3.3.1), le modèle de la variabilité de service (III.3.3.2), le modèle du contexte (0), le modèle des règles d'adaptation (III.3.3.4) et le modèle du métier (III.3.3.5). L'interconnexion entre les modèles est aussi faite graphiquement. La Figure 55 présente les relations entre les différents modèles de l'approche CADSSOMA. Chaque service est relié à ses variabilités de service. Les tables de décision, qui représentent le modèle des règles d'adaptation, relient les paramètres des *ServiceContextElement* aux variabilités de service. Enfin, il faut relier chaque variabilité de service à son implémentation. Cette dernière est représentée via les éléments « tâche du domaine spécifique » du modèle du métier du domaine.

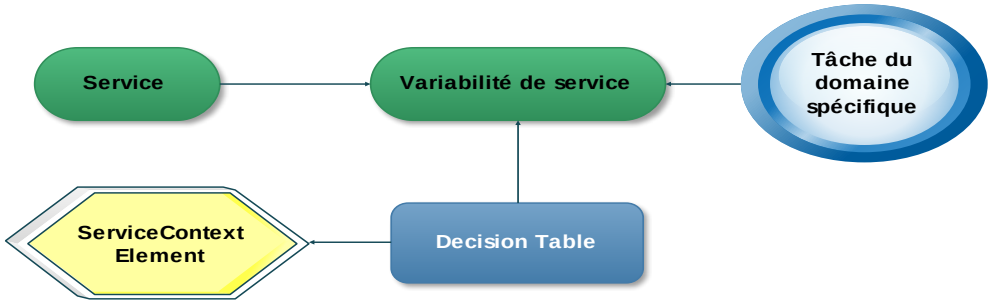


Figure 55 : Relations entre les modèles de l'approche CADSSOMA

Phase 6 : Génération du code source

L'objectif des phases précédentes est la spécification d'un système orienté services adaptable indépendamment des technologies d'implémentations (dotNet, JEE, etc.), et des standards de la technologie des services web (WSDL, UDDI, SOAP, BPEL4WS, etc.). Le résultat de ces phases est un ensemble de modèles de haut niveau. L'objectif de cette phase est la génération de code à partir des cinq modèles de l'approche CADSSOMA. Retenons que l'approche DSM se base sur une transformation directe des modèles spécifiques au code source (Kelly, et al., 2008a). Aucune transformation modèle à modèle n'est prévue. Après la définition des correspondances entre les modèles source et la plateforme cible, il faut déterminer les règles de transformations en utilisant un langage de transformation Modèle-à-Texte[1] donné tel que ATL (ATL, 2016), ACCELIO (OBEO, Acceleo, 2016), etc.

Le générateur du code source transforme l'ensemble des modèles de la phase précédente au code source finale. Dans notre étude de cas il permet de faire une génération vers la plateforme Java. Retenons que le générateur de code source doit être refait à chaque modification de la plateforme cible.

IV.5. Conclusion

Après avoir illustré le processus de développement des systèmes orientés services, nous avons présenté dans ce chapitre notre proposition de formalisme du processus de développement des applications spécifiques au domaine. Ce formalisme fournit une vision globale de l'ensemble des étapes à suivre pour produire une application spécifique au domaine.

Par ailleurs, nous avons aussi présenté dans ce chapitre notre proposition de processus DSM pour le développement des systèmes orientés services adaptables associé à l'approche CADSSOMA. Ce processus définit les phases, les artéfacts et les activités nécessaires pour identifier, spécifier et implémenter les services adaptables d'un domaine spécifique. Les principales phases de ce processus sont :

(1) Analyse du domaine spécifique cible en validant l'applicabilité de l'approche DSM au niveau du domaine cible et en spécifiant les concepts du domaine spécifique ;

(2) Identification des services du domaine spécifique ;

[1] *Model-to-Text Language (MTL)* dans le vocabulaire anglo-saxon.

(3) Création des solutions DSM : métier et services spécifiques au domaine ;

(4) Intégration des deux solutions DSM à la plateforme CADSSOTB ;

(5) Modélisation : modèle de services, modèle de la variabilité des services, modèle du contexte, modèle des règles d'adaptation et modèle métier ;

(6) Génération du code source.

Ce processus est illustré en utilisant une étude d'un système de calcul et de restitution d'impôts. Après avoir présenté l'approche CADSSOMA (chapitre II) ainsi que le processus DSM pour le développement des SOSAs, deux composantes de notre approche d'ingénierie des SOSAs spécifique au domaine, l'objectif du chapitre suivant est de valider notre approche en proposant un environnement de développement intégré (*CADSSO**TB**, **T**ool**B**ox*) qui facilite la mise en œuvre de l'approche CADSSOMA. CADSSOTB prend en charge la modélisation des cinq modèles de la phase de modélisation et la génération du code source.

Chapitre V

CADSSOTB : Un Environnement de Développement des SOSAs Spécifiques au Domaine

V.1. Introduction

Nous présentons dans ce chapitre l'outil support CADSSOTB (***C**ontext **A**ware, **D**omain **S**pecific and **S**ervice **O**riented **T**ool**B**ox*) ayant pour objectif l'industrialisation du processus de développement des applications orientées services adaptables, basé sur l'ingénierie des langages spécifiques au domaine. CADSSOTB peut être considéré comme étant un atelier de génie logiciel permettant la modélisation, l'intégration des modèles, la validation des modèles et la génération de la solution finale.

Il est à noter que les Environnements de Développement Intégrés (EDI) jouent un rôle important dans la facilitation des tâches du processus de production des logiciels, ce qui permet d'améliorer la productivité des développeurs. En effet, ils permettent d'automatiser les tâches récurrentes de développement comme la compilation, la transformation, etc. Dans cette visée, l'outil support CADSSOTB cible essentiellement la facilitation de l'adoption de l'approche CADSSOMA (Lethrech, et al., 2016).

CADSSOTB est défini dans le cadre de l'approche de modélisation spécifique au domaine (DSM) en se basant plus particulièrement sur les standards OMG. Comme déjà cité au niveau du chapitre II, l'approche DSM est une approche générative qui repose sur une boite à outils spécifique. En fait, chaque DSL possède sa syntaxe concrète, un outil de modélisation, un générateur de code source et éventuellement un framework du domaine spécifique. Le caractère spécifique de notre boite à outils permet de produire des modèles conformes aux règles métiers du domaine (en se basant sur l'étape de validation), ce qui améliore la qualité du code source généré. Il y a aussi lieu de rappeler que l'approche DSM ne repose sur aucune transformation Modèle-à-Modèle, vu que les modèles et le générateur de code source sont spécifiques au domaine.

La deuxième section est une étude comparative des environnements de méta-modélisation, la troisième section illustre les différentes étapes suivies pour produire notre boite à outils CADSSOTB, enfin le chapitre est clôturé par une conclusion.

V.2. Environnements de méta-modélisation : Etude comparative

Afin de mettre en place notre boite à outils CADSSOTB, nous avons jugé nécessaire de faire une étude comparative des environnements de méta-modélisation (EMM) du monde open source. Ces derniers représentent les générateurs de boites à outils des langages spécifiques appelés aussi Environnements de Développement Spécifiques au Domaine (EDSD).

Retenons que l'amélioration des plateformes et environnements de méta-modélisation (EMM) facilitant la création des EDSDs a nettement réduit l'effort de mise en place des solutions DSM (ISIS, MIC, 2011) (Kelly, et al., 2008a) (Iseger, 2010) (Achilleos, 2009).

Avant d'entamer l'étude comparative, nous allons mettre l'accent sur les spécificités de la pile de méta-modélisation de l'approche DSM. Ensuite nous allons mettre en lumière l'architecture de la plateforme Eclipse.

V.2.1. Pile de méta-modélisation de l'approche DSM

L'OMG propose deux manières pour produire un DSL (OMG, MOF, 2016) (Achilleos, 2009) (OMG; Soley, R. M., 2001). La première consiste en l'utilisation des profils UML, et la deuxième utilise la spécification MOF pour proposer un méta-modèle au même niveau que celui d'UML (Figure 56). Retenons que la spécification MOF est considérée comme le méta-méta-langage qui supporte la définition de la syntaxe abstraite des langages de modélisation sous forme d'un méta-modèle. La méta-modélisation est le processus qui encadre la définition d'un méta-modèle, qui décrit les éléments d'un domaine spécifique ainsi que leurs propriétés et relations (OMG).

La pile de modélisation utilisée est celle proposée par l'OMG. Elle est composée de plusieurs niveaux de modèles: le modèle, le méta-modèle et le méta-méta-modèle (Figure 56).

Un méta-modèle est « un modèle qui définit le langage pour exprimer un modèle » (OMG, 2003). Un méta-méta-modèle est « un modèle qui définit le langage pour exprimer un méta-modèle ». Quatre niveaux de modélisation sont proposés par l'OMG (Figure 56) :

- Le niveau M0 est le niveau du monde réel. Il représente les informations à modéliser ;
- Le niveau M1 est composé de modèles décrivant les informations du monde réel M0 ;

- Le Niveau M2 représente le niveau des méta-modèles, c'est-à-dire des langages de définition des modèles d'informations ;
- Le niveau M3 représente le niveau du méta-méta-modèle, c'est à dire l'unique langage de définition des méta-modèles appelé MOF (Méta-Object Facility) (OMG, MOF, 2016). Ce dernier permet de définir la structure des méta-modèles du niveau M2. Le niveau M3 correspond donc aux fonctionnalités universelles de modélisation logicielle.

Il est à souligner que le MOF est décrit par lui-même limitant ainsi la pile de modélisation de l'OMG à quatre niveaux.

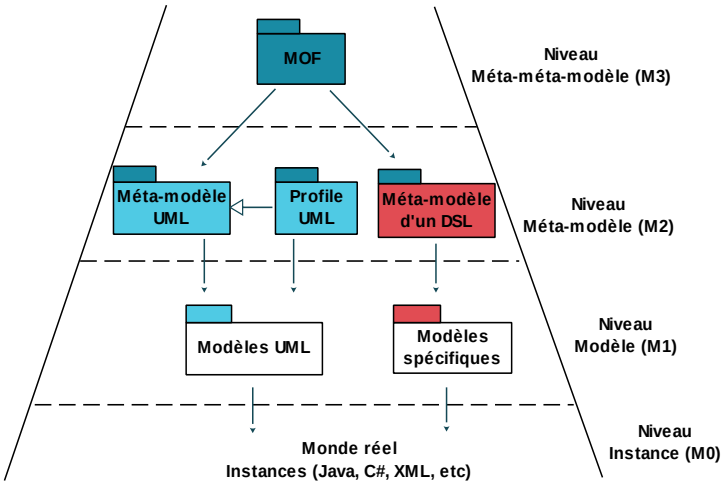


Figure 56 : Pile de modélisation de l'OMG (image adaptée de (Bonnet, 2005))

Il est connu que l'approche MDA, qui représente la concrétisation de l'ingénierie dirigée par les modèles de l'OMG, distingue entre deux types de modèles au niveau M1: un modèle indépendant de la plateforme (PIM) et un modèle spécifique à la plateforme (PSM). Le PIM modélise l'information abstraite nécessaire pour représenter efficacement des applications logicielles indépendamment de la technologie d'implémentation et de la plateforme technologique cible. Au contraire, le PSM comprend des détails spécifiques d'implémentation, et décrit par conséquent un système logiciel en parfaite connaissance de la plateforme d'implémentation. Le PIM est transformé au PSM en utilisant un langage de transformation M2M.

En revanche, l'approche DSM n'utilise aucune transformation Modèle-à-Modèle (Kelly, et al., 2008a). En effet, elle utilise un générateur de code source spécifique à une plateforme donnée, ce qui rend insensé l'utilisation d'un modèle indépendant de la

plateforme. Une solution DSM convertit un modèle d'un domaine spécifique directement à la solution finale en utilisant un générateur de code source spécifique au domaine.

V.2.2. Critères de l'étude comparative des EMMs

Comme nous l'avons expliqué au niveau de la section IV.3, le processus de mise en place d'une solution DSM est composé de plusieurs étapes :

- Elaboration de la syntaxe abstraite : représente la sémantique et les concepts de base du domaine spécifique ***(E1)*** ;
- Elaboration de la syntaxe concrète : représente la notation utilisée pour représenter les modèles du domaine spécifique ***(E2)*** ;
- Identification des contraintes de validation des modèles : représente les règles métier du domaine spécifique ***(E3)*** ;
- Transformations de modèles : conversion d'un modèle à texte (M2T) ou éventuellement conversion d'un modèle à un autre modèle (M2M) ***(E4)*** ;

L'EMM choisi doit pouvoir faciliter la production d'un EDSD qui prend en charge les étapes citées ci-dessus.

A cela s'ajoute le critère d'extensibilité ***(E5)*** qui est en relation avec notre proposition. En effet, un seul environnement doit être utilisé pour les cinq DSLs proposés. En plus, il doit intégrer les différentes étapes de notre approche CADSSOMA. En d'autres termes, CADSSOTB doit être sous forme d'un ensemble d'environnements de développement spécifiques au domaine (EDSD). N'oublions pas que le langage du métier du domaine spécifique est à modifier pour chaque domaine spécifique. Par conséquent, l'EMM doit permettre la modification de la composante responsable de la modélisation du métier du domaine spécifique de la manière la plus simple possible. En outre, le générateur utilise les cinq modèles de notre approche pour produire la solution finale. De ce fait, et pour simplifier sa maintenance et sa mise à jour, le générateur doit être développé d'une manière modulaire avec un faible couplage entre les sous-générateurs spécifiques pour chaque langage.

D'autres critères en relation avec l'EMM sont à considérer :

- Possibilité de générer un EDSD ***(E6)*** ;
- La qualité de la documentation : ce critère joue un rôle important dans la facilitation de l'appropriation d'un outil open source ***(E7)*** ;

- La conformité avec les standards OMG XMI (OMG, XMI, 2015) : ce qui facilite l'intégration et la communication avec d'autres solutions ***(E8)*** ;

V.2.3. Résultat de l'étude comparative des EMMs

Notre étude est basée sur une comparaison empirique qui s'intéresse aux méta-générateurs open source. Les projets propriétaires (MataEdit+, MS DSL, Borland Together, etc.) n'ont pas fait l'objet de test. Le Tableau 11 représente le résultat de notre étude comparative :

Tableau 11 : Résultat de l'étude comparative des environnements de méta-modélisation

	Exigence DSM	**MIC**	**DOME**	**AndroMDA**	**XMF-Mosaic**	**Sirius**	**Eclipse MP**
Critères en relation avec l'environnement de développement spécifique au domaine	**Syntaxe abstraite (E1)**	O	O	O	O	O	O
	Syntaxe concrète (E2)	O	O	N	O	O	O
	Contraintes de Validation (E3)	-	N	N	O	O	O
	Transformation M2M (E4)	N	N	N	O	N	O
	Transformation M2T (E4)	O	O	O	O	N	O
	Environnement Extensible (E5)	-	-	O/N	-	O	O
Critères en relation avec l'environnement de méta-modélisation	**Génération d'un EDSD (E6)**	O	O	N	O	O	O
	Qualité de la documentation (E7)	O/N	N	O	O/N	O/N	O
	Respect des standards (MOF, XMI) (E8)	N	N	O	O/N	O	O

O : Exigence prise en charge N : Exigence non prise en charge O/N : Exigence partiellement prise en charge

La boite à outils MIC (Model Integrated Computing) (ISIS, MIC, 2011) est composée de plusieurs outils œuvrant pour le développement dirigé par les modèles. Le framwork GME (*Generic Modeling Environment*) est utilisé dans la méta-modélisation. La définition de la syntaxe abstraite se base sur un modèle propriétaire appelé *MetaGME*[1]. Ce dernier est modélisé en utilisant la notation du diagramme de classe d'UML. Le framework *GReAT* s'occupe de la transformation modèle à texte (M2T). À noter que l'équipe MIC penche sur le développement d'un pont entre le *MOF* et le *MetaGME* permettant une méta-modélisation en utilisant MOF. Il y a lieu de signaler que la notation graphique est possible et que les contraintes sont exprimées via OCL.

[1] Version 16.7.22 année 2016.

DoME (DOmain Modelling Environment) (DOME, 2012) est un projet de l'université « University of South Australia ». Il utilise un langage de méta-modélisation propriétaire. L'utilisation d'une syntaxe concrète graphique est possible. Ce framework est probablement abandonné vu que sa dernière mise à jour a été faite en 2012 (DOME, 2012).

AndroMDA (AndroMDA, 2015) est un environnement extensible (via des plug-ins nommés *cartridges*) qui adhère à la plupart des spécifications MDA. Son extensibilité est utilisée pour permettre la prise en charge de générateurs de code personnalisés. En revanche, AndroMDA ne prend pas en charge la génération d'une boite à outils DSM, en outre il ne permet pas de faire une validation des modèles. Un plug-in AndroMDA d'Eclipse a été développé pour lui permettre de tirer profit des outils d'Eclipse.

XMF-Mosaic (XMF-Mosaic, 2011) est un environnement de développement de solutions DSM, basé sur la plateforme Eclipse. Il utilise le langage XMF pour le développement des langages spécifiques au domaine. XMF-Mosaic a été initialement un projet commercial ; actuellement il est accessible via la licence « projet Eclipse ». Pour la méta-modélisation, il utilise le modèle propriétaire XCore. Afin de respecter les standards OMG, il propose aussi un MOF-like langage de méta-modélisation. Il est conforme au standard XMI. La syntaxe concrète peut être graphique ou textuelle. Il permet de faire des transformations M2M et M2T. La validation des modèles est interactive en utilisant le langage propriétaire de contraintes XOCL. XMF-Mosaic peut être utilisé en étroite collaboration avec les capacités logicielles du projet de modélisation Eclipse (Eclipse Modeling Project) (Gronback, 2009).

Sirius (OBEO, Sirius, 2015) est un projet open source sponsorisé par les sociétés Obeo et Thales. C'est un projet très actif avec une communauté d'une taille importante[1]. La méta-modélisation est effectuée en se basant sur le model EMF (Eclipse Modeling Framework). Il permet d'utiliser une notation graphique basé sur le framework GMF (Graphical Modeling Framework). L'appropriation de Sirius est délicate, vu qu'il est faiblement documenté. Il fait aussi partie du projet de modélisation d'Eclipse.

Basé sur la plateforme Eclipse, le Projet de Modélisation d'Eclipse (PME) est un projet très prometteur qui offre une panoplie d'outils supportant le développement spécifique au domaine. Composé de plus que 30 projets (Eclipse, EMP, 2016), l'initiative de modélisation d'Eclipse offre un grand choix d'outils couvrant l'ensemble des étapes de

[1] Au 10/2016 Sirius possédait 18 validateurs ; la dernière validation date du 10/2016 (http://projects.eclipse.org/projects/modeling.sirius).

l'ingénierie dirigée par les modèles. Plusieurs initiatives de développement à base de modèles ont intégré la plateforme Eclipse (Sirius, XMF-Mosaic, AndroMDA, etc.), ce qui prouve que l'initiative d'Eclipse représente la locomotive du développement dirigé par les modèles dans le monde open source. La méta-modélisation est basée sur le modèle Eclipse Modeling Framework (EMF).

Pour conclure, le projet de modélisation Eclipse répond à l'ensemble des exigences de notre étude comparative (Eclipse, EMP, 2016). En outre, d'autres études empiriques prouvent que la boite à outils open source d'Eclipse est même meilleure que d'autres projets propriétaires à l'instar de MS DSL Tool (Langlois, et al., 2007) (Pelechano, et al., 2006). Cela essentiellement grâce au caractère extensible de la plateforme Eclipse.

V.2.4. Plateforme Eclipse : Un environnement de développement extensible

La plateforme de base utilisée pour produire notre boite à outils CADSSOTB est la plateforme Eclipse[1] (Eclipse, 2016). C'est un environnement de développement intégré gratuit et largement utilisé[2]. Nous citons ci-dessous les principales caractéristiques d'Eclipse qui ont influencé sa sélection :

- Architecture extensible : la plateforme Eclipse est basée sur le concept de plug-in. Un plug-in est un paquet structuré de code et/ou des données qui contribue aux fonctionnalités du système. Eclipse SDK est composé de la plateforme de base plus deux outils principaux de développement de plug-ins. Le premier est l'outil *Java Development Tools* (JDT), il représente un environnement de développement Java complet (Eclipse, help, 2016). Le deuxième est le framework *Plug-in Developer Environment* (PDE), il offre des outils spécialisés qui simplifient le développement des plug-ins et d'extensions (Figure 57).

  La mise en place d'un environnement de développement sans l'utilisation d'un outil de méta-modélisation représente la tâche la plus couteuse du processus de mise en place d'une solution DSM. Dans notre cas, et en vue d'optimiser l'effort d'implémentation de cette dernière, nous nous sommes basés sur les frameworks du projet de modélisation d'Eclipse (Eclipse Modeling Project). La plateforme Eclipse génère les EDSDs des différents DSLs utilisés sous forme de plug-in Eclipse (Figure

[1] Le projet Eclipse a été initialement créé par IBM (en Novembre 2001) et soutenue par un consortium d'éditeurs de logiciels. Eclipse est géré par la Fondation Eclipse qui a été créée en Janvier 2004 en tant que société indépendante sans but lucratif.
[2] Plus que 1400 développeurs en 2015 ; plus que 85 projets en 2015 ; 250 organisme membre en juin 2016 ; en 3 jours 180K téléchargements de la version Kepler (https://eclipse.org/org/foundation/reports/annual_report.php).

58). Ils seront intégrés à la plateforme Eclipse pour étendre ses capacités par celles des nouveaux EDSDs (Figure 58).

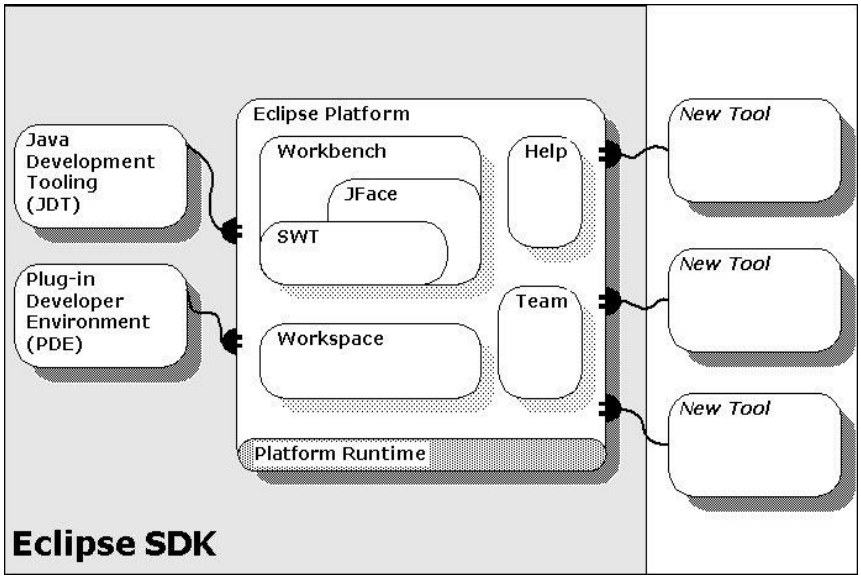
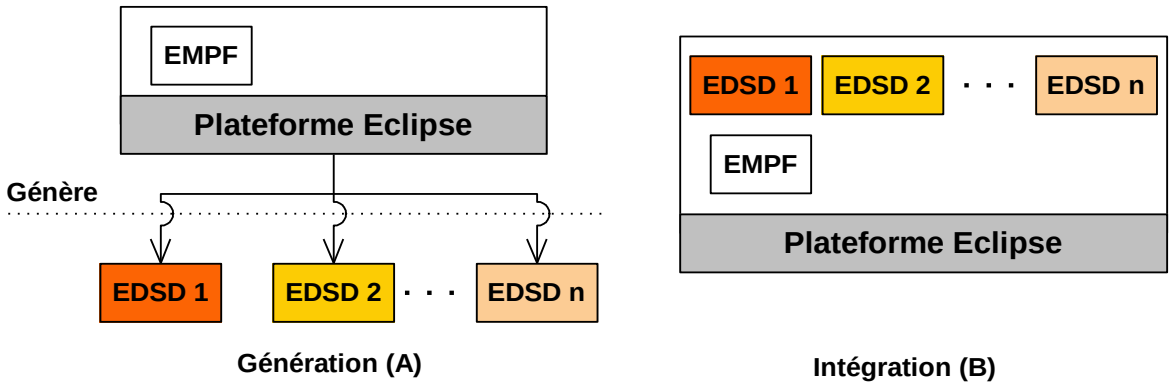


Figure 57 : Architecture d'Eclipse (Eclipse, help, 2016)



EMPF : Eclipse Modeling Project Frameworks

Figure 58 : Caractère extensible de l'environnement de développement Eclipse

- Comme déjà cité au niveau de la section V.2.3, le projet de modélisation d'Eclipse (Eclipse, EMP, 2016) œuvre pour l'évolution et la promotion des technologies de développement à base de modèles, en fournissant un ensemble de frameworks de modélisation, d'outils et d'implémentation de standards (Figure 59). Une multitude de choix d'outils sont proposés. Ils couvrent l'ensemble du processus de développement dirigé par les modèles, à savoir, la modélisation de la syntaxe concrète (EMF), la modélisation de la syntaxe concrète (GMF, Sirius, Xtext, etc.), la transformation de modèles (Acceleo, ATL, Xpand, etc.), la validation de modèles (Validation Framework) et les interfaces utilisateur (EMF client platform, EMF Forms, etc.), etc.

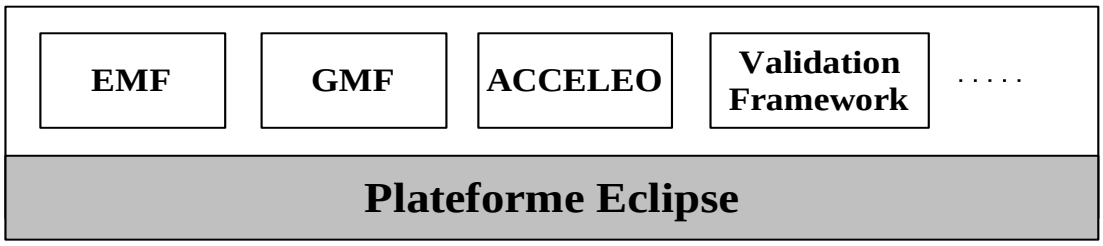


Figure 59 : Outils utilisés du projet de modélisation Eclipse

- C'est un projet open source avec une communauté très active (plus que 1400 développeurs en 2015[1]).

Pour conclure, la plateforme Eclipse permet de générer un environnement de développement spécifique au domaine (EDSD) avec un minimum de temps et d'effort. Encore plus, elle héberge les EDSDs produits pour former une dérivée de la plateforme Eclipse qui prend en charge les besoins de développement spécifiques au domaine.

V.3. Mise en place de l'outil support CADSSOTB

La Figure 60 illustre les étapes nécessaires pour produire l'une des cinq solutions DSMs de notre approche :

- Elaboration de la syntaxe Abstraite : elle est sous forme d'un méta-modèle qui représente la sémantique du domaine spécifique. La syntaxe Abstraite doit être conforme au MOF. Elle comprend les entités, les associations et les propriétés d'un domaine spécifique. A cette étape le framework EMF (Eclipse Modeling Framework) est utilisé ;
- Elaboration de la syntaxe concrète : identification de la notation à utiliser. C'est la syntaxe concrète utilisée lors de la création des modèles. Nous avons opté pour le framework GMF (Graphical Modeling Framework) ;

[1] https://eclipse.org/org/foundation/reports/annual_report.php.

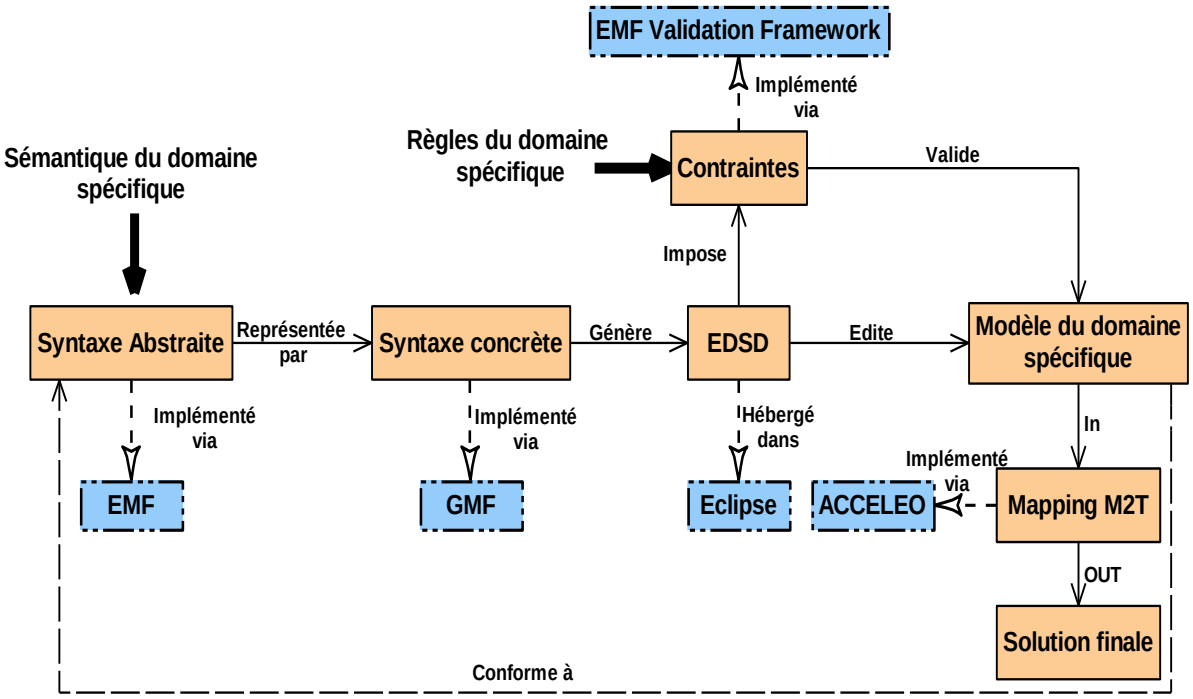


Figure 60 : Processus de mise en place d'une boite à outils d'un DSL

- Génération de l'EDSD : Un EDSD est à produire pour chaque DSL de l'approche CADSSOMA. En pratique, il faut refaire les étapes précédentes pour les cinq DSLs de notre approche. La plateforme de base utilisée est la plateforme Eclipse ;
- Spécification et développement des règles de conception à respecter à partir des règles de gestion du domaine spécifique. Ces règles sont implémentées en utilisant le Framework EMF Validation Framework ;
- Mapping M2T : Le mapping modèle-à-texte permet de convertir les cinq modèles de l'approche CADSSOMA à la solution finale. La correspondance est faite en utilisant l'outil ACCELEO.

Les étapes citées ci-dessus sont détaillées dans les sections ci-dessous.

V.3.1. Syntaxe abstraite

Si la modélisation s'intéresse à la représentation d'un système, la méta-modélisation s'intéresse à la modélisation d'un domaine, et ce afin de faciliter la manipulation de modèles (éditeurs, transformation, sérialisation, etc.). La méta-modélisation s'occupe de la définition des concepts et des relations d'un domaine, elle est presque identique à la définition d'une grammaire.

Les syntaxes abstraites des DSLs proposés ont été modélisées en utilisant Eclipse Modeling Framework (EMF) (Steinberg, et al., 2008a). EMF est un Framework de

modélisation et de génération d'outils basés sur un modèle de données structurées (Eclipse, EMP, 2016).

EMF est constitué de trois pièces fondamentales (Eclipse, EMP, 2016) (Figure 61) :

- **EMF Core** : Inclut le méta-modèle Ecore utilisé dans la description des modèles. Il offre aussi un support pour l'exécution des modèles, comprenant la notification de changement, la persistance avec sérialisation XMI par défaut, et une API de réflexion pour manipuler génériquement les objets EMF ;
- **EMF .Edit** : Comprend des classes génériques et réutilisables pour la construction d'éditeurs pour les modèles EMF ;
- **EMF .Codegen** : Représente le générateur de code EMF. Il est capable de générer tout le nécessaire pour construire un éditeur complet pour un modèle EMF. L'éditeur généré est un éditeur arborescent.

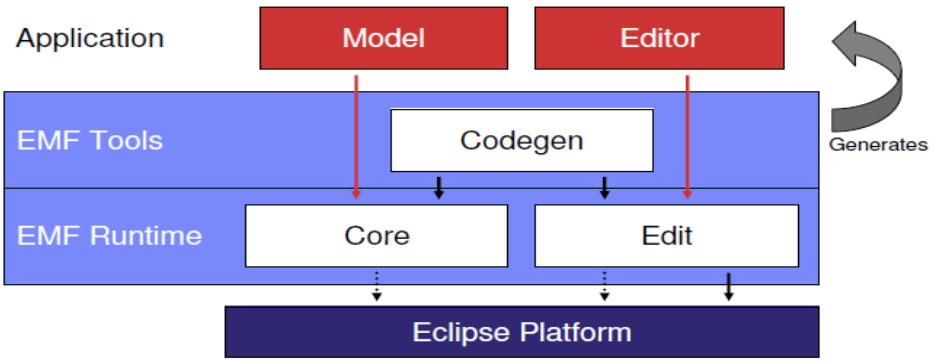


Figure 61 : Architecture d'EMF (Steinberg, et al., 2008b)

Le méta-modèle utilisé pour représenter les modèles dans EMF est appelé Ecore (Figure 62). Ce méta-méta modèle permet de définir la syntaxe abstraite d'un DSL donné en utilisant un éditeur arborescent.

EMF et MOF sont conceptuellement très proches (Maddeh, et al., 2007) (Langlois, et al., 2007). En effet, la version actuelle de MOF 2.5 incorpore un sous modèle appelé Essential MOF (EMOF). Il est identique au méta-modèle Ecore d'EMF à quelques exceptions de nomenclatures. EMF peut lire et écrire des sérialisations du méta-modèle EMOF (Achilleos, 2009). Une correspondance entre les éléments de MOF et EMF est présentée au niveau du travail de Maddeh et al. (Maddeh, et al., 2007) (Annexe A).

L'élément *EObject* est l'élément racine de l'ensemble des classes EMF (équivalent à *java.lang.Object* dans le monde Java). L'élément *EClass* (Figure 62) modélise un concept du domaine spécifique visé. Il est caractérisé par des *Eattributes* et des *Ereferences* (et des

EOperations voir Figure 89 annexe A). L'élément *EReference* permet de relier deux concepts et l'élément *EAttribute* représente un attribut d'une *EClasse*.

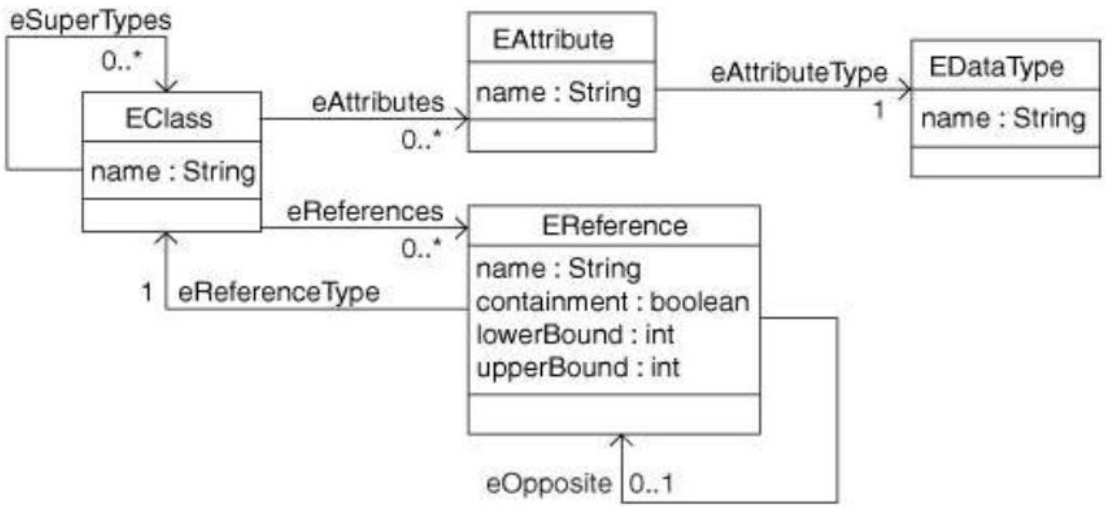


Figure 62 : Extrait du méta-méta-modèle Ecore (Steinberg, et al., 2008a)

Retenons que la modélisation en utilisant Ecore oblige tout méta-modèle Ecore d'avoir une *EClass* racine avec des liens "*containment*" vers toutes les classes concrètes (Steinberg, et al., 2008a). Cette classe racine représente le système à modéliser. Dans notre cas cette classe est nommée *AdaptableSOA,* elle est la classe racine de l'ensemble des *EClass* des cinq méta-modèles de notre approche CADSSOMA.

Pour des raisons organisationnelles, nous avons divisé le méta-modèle de l'approche CADSSOMA représenté au niveau de la Figure 48 en quatre packages (*cadsso.context, cadsso.decisionTable, baseservice, servicevariability*). Afin d'éviter des dépendances cycliques, EMF ne permet que des associations unidirectionnelles d'un élément d'un package vers un autre élément d'un autre package (Steinberg, et al., 2008a). Par exemple l'élément *ServiceVariationPoint* du package *servicevariability* est visible pour l'élément *service* du package *baseservice* mais l'inverse n'est pas possible (Figure 63).

Toutes les méta-classes du modèle Ecore du langage de modélisation des services de calcul d'impôts sont définies en utilisant l'élément *EClass* affiché sur la palette de l'éditeur GMF (Figure 63). De plus, chaque méta-classe contient les propriétés requises qui sont définies en utilisant l'élément *EAttribute.* Outre les méta-classes et les propriétés, la définition du méta-modèle comprend des associations (association, agrégations et héritage) définies par les outils *EReference* et *Inheritance*.

Plusieurs manières existent pour produire un modèle Ecore (interfaces Java annotées, schéma XML, modèle UML, etc.). Nous avons opté pour l'utilisation du générateur de

modèles EMF (EMF Generator Model) qui prend en entrée des interfaces Java annotées et génère les fichiers *.genmodel* et *.ecore* (Annexe B). Le premier est utilisé dans la génération du code Java du modèle et de l'éditeur. Le deuxième représente le modèle Ecore du domaine spécifique.

V.3.1.1. Syntaxe abstraite du langage de modélisation des services de calcul d'impôts

La Figure 63 représente le modèle Ecore du langage de modélisation des services de calcul d'impôts. Pour des raisons de lisibilité, la liaison de composition forte de l'élément *AdaptableSOA* avec l'ensemble des éléments du diagramme n'a pas été représentée.

L'énumération *Tax* peut représenter d'autres types d'impôts autres que *CorporationTax* et *IncomeTax*. On s'est contenté dans cette version du méta-modèle de la représentation du service métier de calcul d'impôts *TaxCalculation* et des méta-éléments du domaine spécifique de calcul d'impôts : *TaxDeclaration* et *RatePayer*. Plus de détail du méta-modèle des services de calcul d'impôts existe dans la section III.3.2.1.

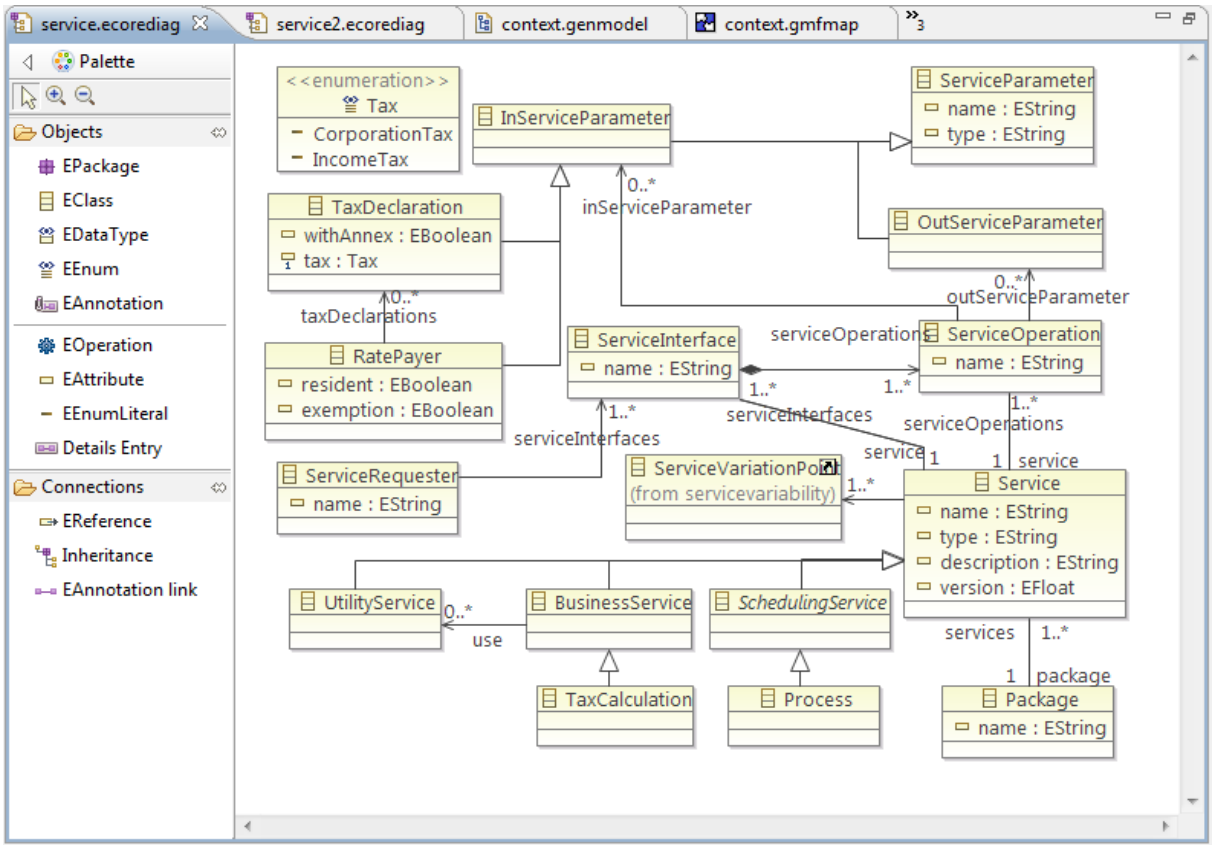


Figure 63 : Modèle ECore du langage de modélisation des services de calcul d'impôts

V.3.1.2. Syntaxe abstraite du langage de modélisation de la variabilité de service

La Figure 64 représente le modèle Ecore du langage de modélisation de la variabilité de service. On s'est contenté dans cette version du méta-modèle de la représentation de la variabilité de service des types de variabilités suivantes : *Logic* est représenté par *LogicalServiceVariationPoint*, *SchedulingStrategy* par *SchedulingStrategyServiceVariationPoint* et *GraphicalPresentation* par *GraphicalPresentationServiceVariationPoint.* Plus de détail du méta-modèle existe dans la section III.2.2.

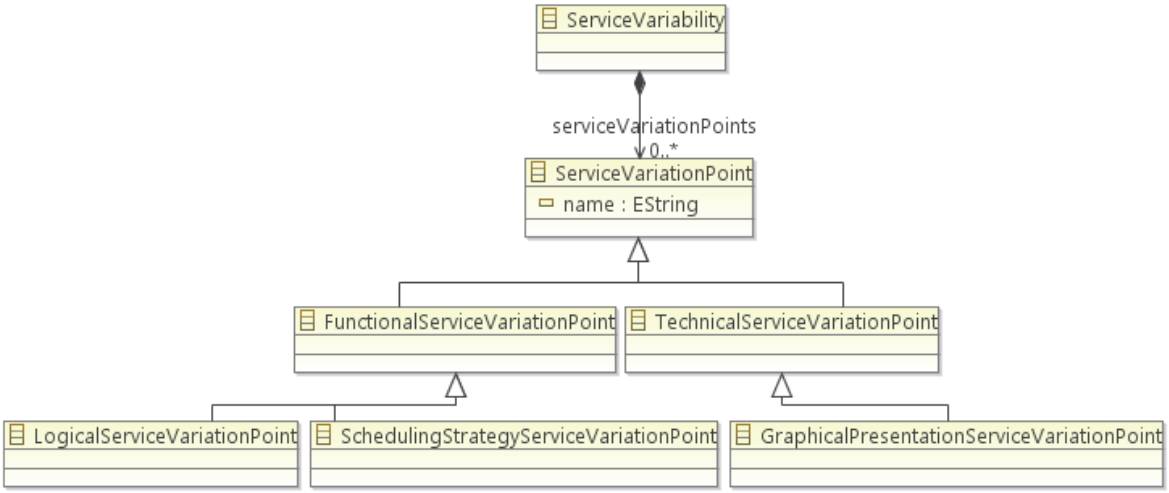


Figure 64 : Modèle ECore du langage de modélisation de la variabilité de service

V.3.1.3. Syntaxe abstraite du langage de modélisation du contexte

La Figure 65 représente le modèle Ecore du langage de modélisation du contexte. La classe racine *AdaptableSOA* est composée de l'ensemble des *EClass* du modèle. Plus de détail du méta-modèle du contexte existe dans la section III.2.3.

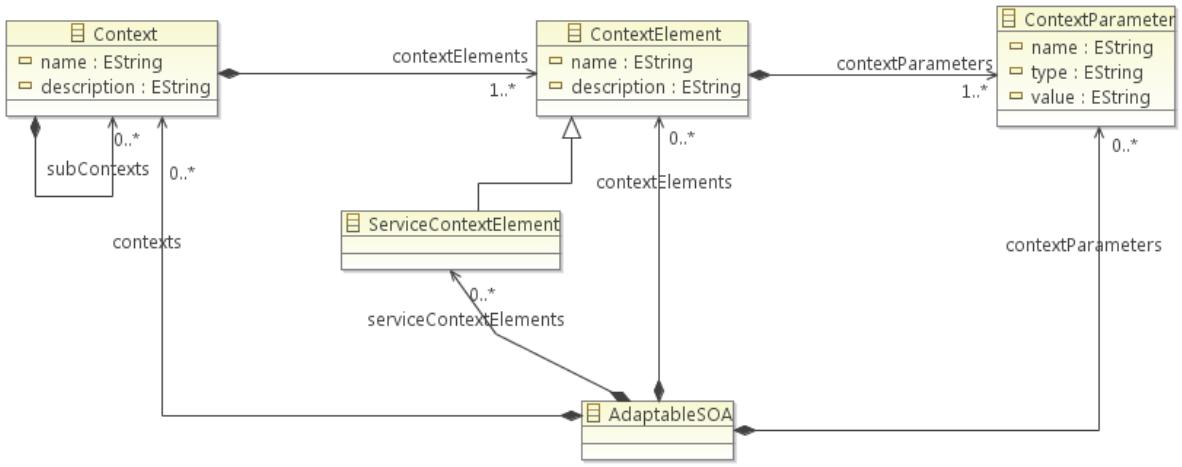


Figure 65 : Modèle ECore du langage de modélisation du contexte

V.3.1.4. Syntaxe abstraite du langage de modélisation des règles d'adaptation (tables de décision)

La Figure 66 représente le modèle Ecore du langage de modélisation des règles d'adaptation. La classe racine *AdaptableSOA* est composée de l'ensemble des *EClass* du modèle. En fait, le langage de modélisation des règles d'adaptation n'est qu'un langage de modélisation des tables de décision. Soulignons le fait que le méta-modèle du contexte et le méta-modèle de variabilité de service sont visibles par le méta-modèle des règles d'adaptation via, respectivement, les *EClass ContextParameter* et *ServiceVariationPoint*. Plus de détail du méta-modèle des règles d'adaptation existe dans la section III.2.4.

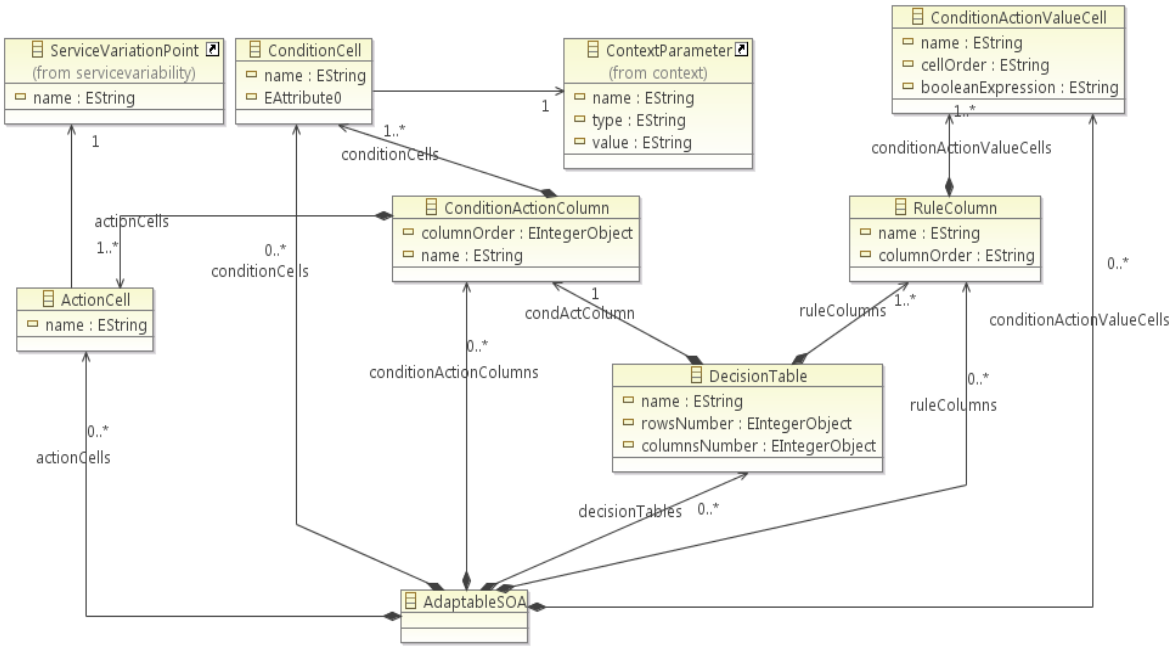


Figure 66 : Modèle ECore du langage de modélisation des règles d'adaptation

V.3.1.5. Syntaxe abstraite du langage de modélisation des règles de calcul d'impôts

La Figure 67 représente le modèle Ecore du langage des règles de calcul d'impôts. Plus de détail du modèle de calcul d'impôts existe dans la section III.3.2.2.

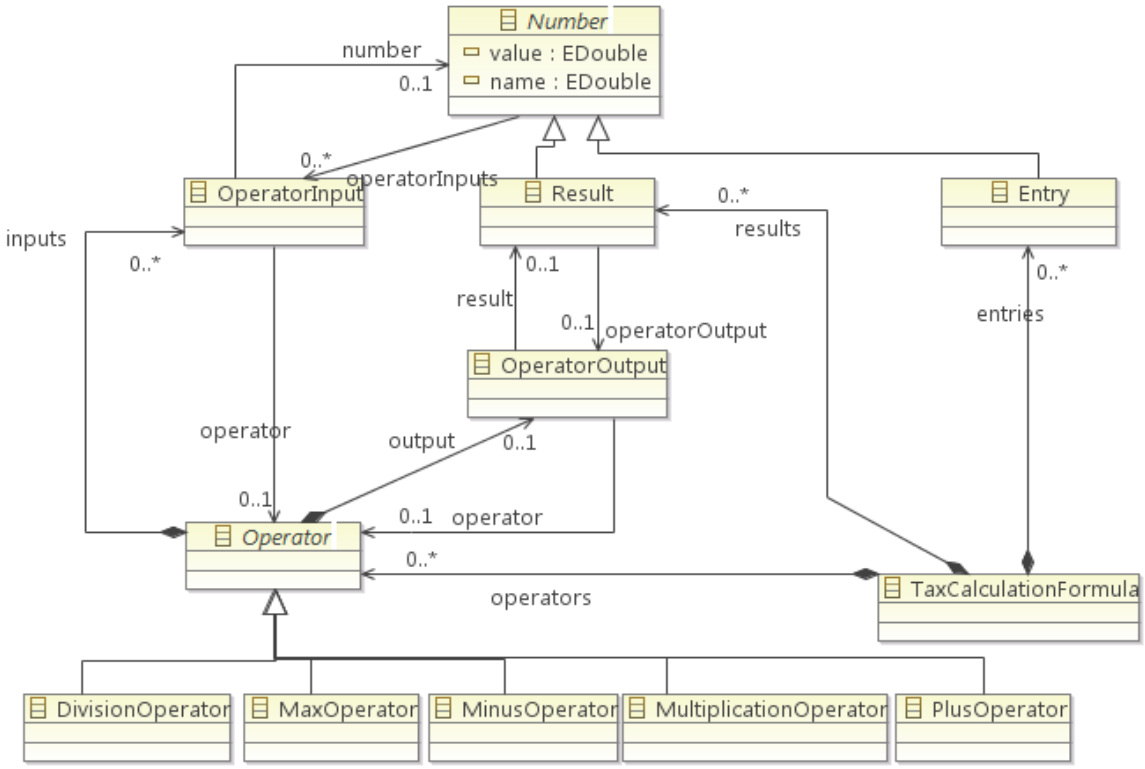


Figure 67 : Modèle ECore du langage de calcul d'impôts

Le Framework EMF représente le cœur du projet de modélisation Eclipse. En effet, les méta-modèles définis en utilisant les concepts du méta-méta-modèle Ecore représentent le point d'entrée aux autres outils de l'initiative de modélisation d'Eclipse.

V.3.2. Syntaxe concrète

V.3.2.1. Choix de la notation des DSLs utilisés : Syntaxe concrète

La notation d'un langage est la représentation visuelle de ses modèles. La définition de la notation d'un langage ne peut être judicieuse qu'après l'identification des concepts de modélisation (syntaxe abstraite). Bien que la représentation des modèles joue un rôle mineur dans la génération de la solution finale, elle est d'une grande importance en ce qui concerne l'acceptation et la facilité d'utilisation, surtout au début de l'utilisation d'un nouveau langage (MataCase; Kelly, S., 2012). En réalité, la partie qui est visible pour les utilisateurs d'un DSL est la notation d'un langage (syntaxe concrète). Par conséquent, il faut l'entretenir pour éviter des commentaires qui vont à l'encontre des efforts déployés au niveau des autres modules, plus complexes, de la solution DSM.

Une bonne pratique à suivre lors de la définition de la notation d'un langage consiste en l'intégration des utilisateurs finaux dans la sélection des symboles et des autres éléments

de représentation (Kelly, et al., 2008a). Nous tenons à souligner que les modèles peuvent être utilisés par des utilisateurs de l'extérieur de l'équipe DSM, par exemple par la direction générale, service commercial, etc.

En vue de faciliter la lecture des modèles, il est déconseillé d'utiliser le même symbole pour l'ensemble des concepts du langage, car cela complique la lecture des modèles. Idéalement, les représentations des concepts du domaine doivent être identiques à leurs représentations réelles. D'autres bonnes pratiques concernant la création de la syntaxe concrète existent au niveau des sections II.3.2.5 et IV.3.2.

En général, la syntaxe concrète des DSLs est graphique ou textuelle. Malgré que l'être humain préfère une représentation visuelle au lieu d'une représentation textuelle[1], n'oublions pas qu'il existe d'autres formes de représentations comme les matrices, les arbres, etc (MataCase; Kelly, S., 2012). Le même modèle peut être représenté en utilisant différents types de représentations. En outre, la manipulation des modèles est possible quel que soit le type de représentation utilisée. Le choix de la représentation à adopter dépend de l'utilisateur final, la structure des données du domaine spécifique et la façon avec laquelle les utilisateurs vont exploiter les données (Kelly, et al., 2008a). Faire un mauvais choix peut considérablement augmenter le coût de création, exploitation et maintenance des modèles (Kelly, et al., 2009b).

Nous avons adopté l'adage « Un schéma vaut mieux que mille mots », et ce en utilisant une notation graphique pour l'ensemble des syntaxes concrètes des DSLs proposés dans le cadre de notre approche CADSSOMA. Dans ce sens, une conversion de la syntaxe abstraite vers une syntaxe concrète graphique est à prévoir. Cette conversion doit être faite dans les règles d'art, c'est à dire en évitant d'avoir un excès ou un déficit de symboles (syntaxe abstraite ←→ syntaxe concrète) (Figure 68).

La notation graphique désigne essentiellement les figures graphiques utilisées pour concevoir des modèles de domaine au niveau d'un éditeur graphique (Figure 72 par exemple). Ce dernier est composé d'une palette de boutons représentant les différents concepts de la syntaxe abstraite et d'un espace de modélisation en utilisant la palette de boutons via la fonctionnalité glisser-déposer.

[1] Selon Kelly 75% de la population préfère une représentation graphique (Kelly, et al., 2009b).

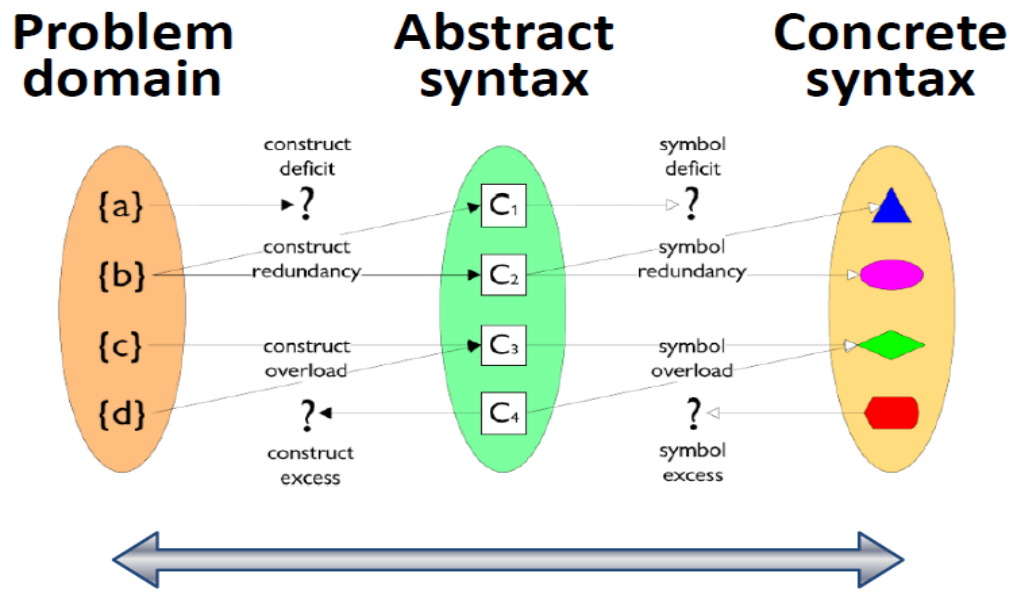


Figure 68 : Correspondance entre les concepts d'un domaine, la syntaxe abstraite et la syntaxe concrète (MataCase; Kelly, S., 2012)

V.3.2.2. GMF : Un outil de transformation d'une syntaxe abstraite à une syntaxe concrète graphique

Nous avons utilisé le framework de modélisation graphique (Graphical Modeling Framework (GMF)), qui fait partie du projet de modélisation Eclipse (Gronback, 2009) (Eclipse, EMP, 2016), pour implémenter la syntaxe concrète des cinq DSLs proposés.

GMF vise à faciliter le développement d'éditeurs graphiques en utilisant GEF (Graphical Editing Framework) (Rubel, et al., 2011) (Eclipse, EMP, 2016) et un modèle EMF sous-jacent.

GMF est composé de deux principaux composants (Gronback, 2009) :

- *Runtime* : permet de dresser un pont entre EMF et GEF en fournissant un ensemble de services et d'APIs permettant le développement d'éditeurs graphiques riches. Ce composant est exploité par le composant ci-dessous ;
- *Tooling Framework* : fournit une approche dirigée par les modèles permettant de définir les éléments graphiques, les outils (boutons glisser-déposer) et leurs correspondances au modèle du domaine. En effet, un éditeur graphique est défini en utilisant une collection de modèles (DSLs) qui dirigent la génération du code de l'éditeur (Figure 69 ; voir aussi la Figure 95 dans l'annexe C). Pour commencer, un projet GMF est créé en faisant référence à un modèle d'un domaine spécifique (fichier *.ecore*) (voir annexe B). Un modèle de définition graphique (Graphical Definition

Model *.gmfgraph*) spécifie les figures (les nœuds, les liens, les compartiments, etc.) utilisées pour représenter les concepts décrits au niveau du méta-modèle abstrait (ou modèle du domaine) sur l'espace de modélisation de l'éditeur. Un modèle de définition d'outillage (Tooling Definition Model *.gmftool*) prend en charge la définition de la palette d'outils de l'éditeur qui permet d'utiliser la fonctionnalité glisser-déposer sur son espace de modélisation.

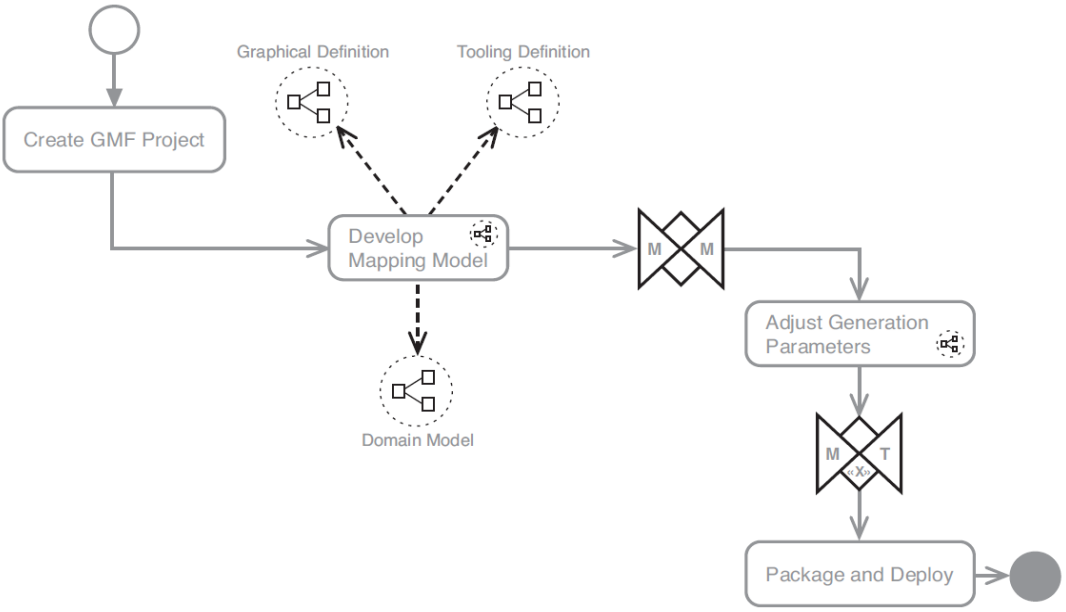


Figure 69 : Flux de travail du composant Tooling Framework de GMF (Gronback, 2009)

Le modèle de correspondance (Mapping Model *.gmfmap*) relie les éléments des définitions graphiques et d'outillage (syntaxe concrète) avec le modèle du domaine (syntaxe abstraite). Une transformation du modèle de correspondance au modèle de générateur (Generator Model *.gmfgen*) est suivie par la génération d'un plug-in Eclipse de l'éditeur graphique. Le modèle de générateur (*.gmfgen*) permet de faire une dernière personnalisation de l'éditeur cible comme par exemple la spécification de l'extension des fichiers XMI représentant les modèles produits.

La Figure 70 illustre un exemple du DSL de modélisation du contexte. Il est à noter que le fichier *context.ecorediag* est la représentation graphique du méta-modèle du contexte *context.ecore*.

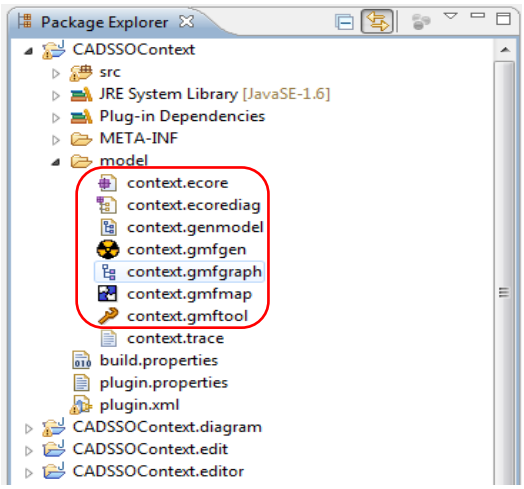


Figure 70 : Exemple d'utilisation du composant Tooling Framework du GMF (modèle du contexte)

En pratique, GMF permet de transformer automatiquement une syntaxe abstraite d'un DSL à une syntaxe concrète graphique. La Figure 71 illustre l'assistant de génération des représentations graphiques (*nœuds*, *connexions* et *labels*) à partir des éléments du méta-modèle de calcul d'impôts. La méta-classe de base utilisée est *TaxCalculationFormula*.

L'assistant exécute réellement une transformation de la syntaxe abstraite vers la syntaxe concrète, et génère le méta-modèle graphique qui utilise les figures par défaut. En outre, la personnalisation du méta-modèle graphique est possible en utilisant l'éditeur graphique GMF. La personnalisation offre la possibilité d'améliorer l'apparence de la représentation graphique et d'accroître la compréhension des modèles du domaine.

V.3.2.3. Syntaxe concrète des langages de modélisation des services et de la variabilité de service de calcul d'impôts

Le modèle des services de calcul de l'impôt sur les sociétés modélisé en utilisant l'outil de modélisation des services de calcul d'impôts est illustré dans la Figure 72. Le service *CTCalculationService* possède deux interfaces : *CTCalculationServiceInterface1* et *CTCalculationServiceInterface2*. La première interface possède une seule *ServiceOperation* nommée *cTaxCalculation*. Elle utilise deux *inServiceParameters* : *ratePayer1* et *cTaxDeclaration1*. Elle retourne un seul *outServiceParameter* nommé *ctCalculationResult*. Le service métier *CTCalculationService* utilise le service utilitaire *GetCTRate*. Ces derniers sont regroupés dans le package *ctpackage*.

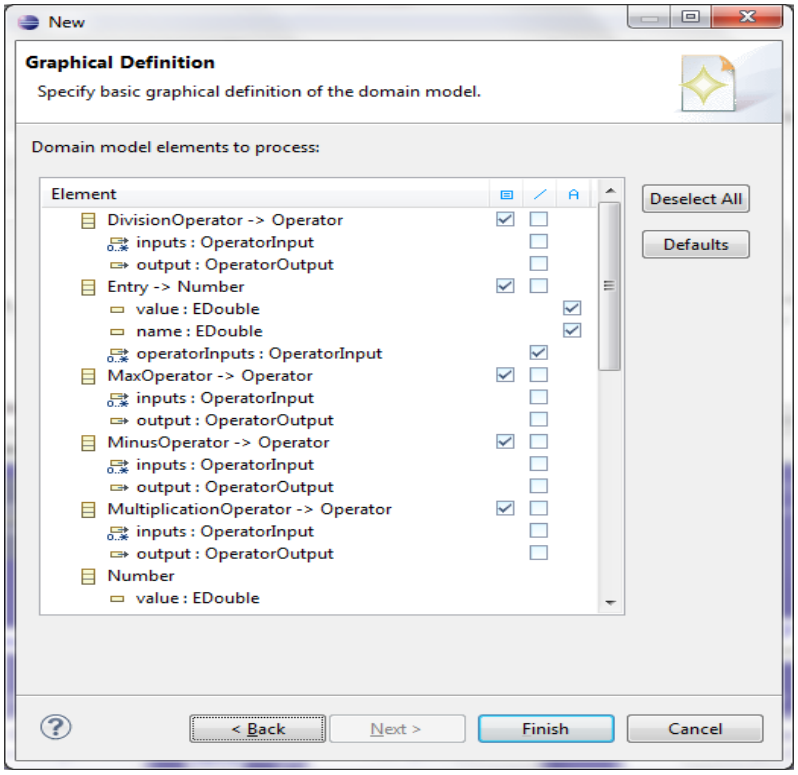


Figure 71 : Assistant GMF de transformation de la syntaxe abstraite (*.ecore*) à une syntaxe concrète (*.gmfgraph*)

De même pour le modèle de variabilité du service de calcul de l'impôt sur les sociétés (*CTCalculationService)* modélisé en utilisant l'outil de modélisation de la variabilité de services. Le service *CTCalculationService* possède quatre points de variation logiques *WithExemption, WithoutExemption, AutomaticTaxation* et *NotResident*. La Figure 73 illustre le modèle de variabilité de service du service de calcul de l'impôt sur les sociétés.

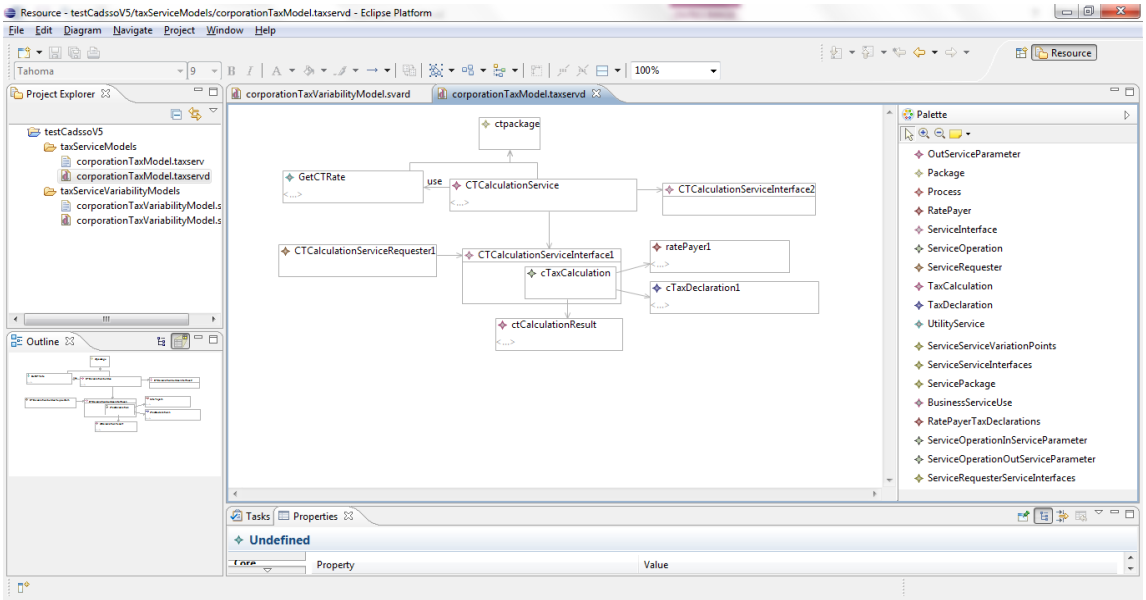


Figure 72 : Modèle des services de calcul de l'impôt sur les sociétés

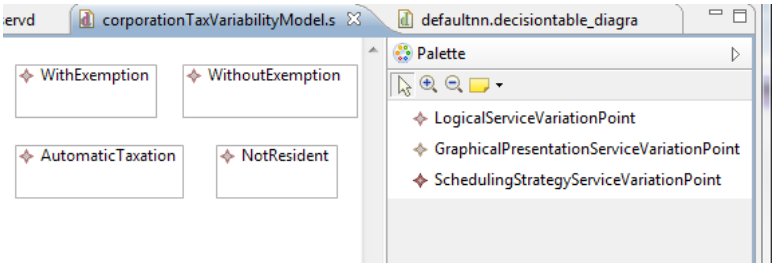


Figure 73 : Modèle de la variabilité de service du service *CTCalculationService*

La propriété « *Service Variation Point* » de l'élément *TaxCalculation*, figurant sur la palette de boutons, permet de lier un service à ses points de variation. L'exemple du service *CTCalculationService* est illustré dans la Figure 74.

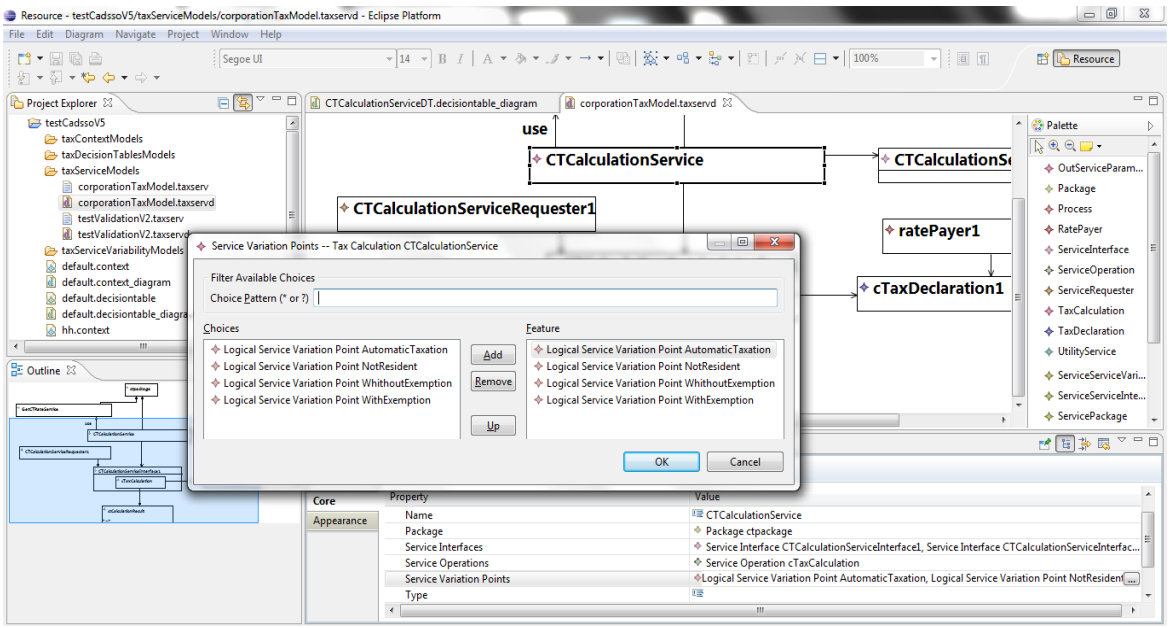


Figure 74 : Spécification des points de variation du service *CTCalculationService*

Le Tableau 12 illustre la correspondance entre la syntaxe abstraite et la syntaxe concrète du langage de modélisation des services de calcul d'impôts.

Tableau 12 : Correspondance entre la syntaxe abstraite et la syntaxe concrète du langage de modélisation des services de calcul d'impôts.

	Elément du méta-modèle	Bouton correspondant de la palette de boutons	Description
Nœuds	*OutServiceParameter*	*OutServiceParameter*	Représente le paramètre de sortie d'une opération d'un service
	Package	*Package*	Regroupe plusieurs services
	RatePayer	*RatePayer*	Représente un contribuable
	ServiceInterface	*ServiceInterface*	Représente une interface de service
	ServiceOperation	*ServiceOperation*	Représente une opération d'un service
	TaxCalculation	*TaxCalculation*	Représente le service de calcul d'impôts
	TaxDeclaration	*TaxDeclaration*	Représente une déclaration spécifique à un impôt
	UtilityService	*UtilityService*	Représente un service utilitaire
	LogicalServiceVariationPoint	*LogicalServiceVariationPoint*	Représente un point de variation logique d'un service
Connexions	Liaison entre Service et ServiceVariationPoint	Propriété « *Service Variation points* » du service *TaxCalculation*	Permet de spécifier les points de variation d'un service
	Liaison entre *Service* et *ServiceInterface*	*ServiceServiceInterfaces*	Permet de modéliser les connexions entre un service et ses interfaces (*ServiceInterface*)
	Liaison entre *Service* et *Package*	*ServicePackage*	Connexion modélisant l'appartenance d'un service à un package
	Liaison « *use* »	*BusinessServiceUse*	Représente la connexion entre un service métier (*BusinessService*) et un service utilitaire (*UtilityService*)
	Liaison entre *RatePayer* et *TaxDeclaration*	*RatePayerTaxDeclaration*	Représente la connexion entre un contribuable (*RatePayer*) et sa déclaration (*TaxDeclaration*)
	Liaison entre *ServiceOperation* et *InServiceParameter*	*ServiceOperationInServiceParameter*	Connexion permettant de spécifier les paramètres d'entrée (*InServiceParameter*) d'une opération de service (*ServiceOperation*)

V.3.2.4. Syntaxe concrète du langage de modélisation du contexte

Le modèle du contexte de l'impôt sur les sociétés modélisé en utilisant l'outil de modélisation du contexte est illustré dans la Figure 75. Nous tenons à souligner que nous avons utilisé les compartiments GMF (*compartment)* permettant de modéliser un élément à l'intérieur d'un autre élément. Par exemple les *ContextElements* sont modélisés à l'intérieur

d'un *Context* et les *ContextParameters* sont modélisés à l'intérieur d'un *ContextElement*. Les *ServiceContextElements* sont représentés en utilisant la couleur bleu claire et les *ContextParameters* en utilisant la couleur rose foncée. La couleur rose claire représente les *ContextElements*. Le contexte *TaxContext* est composé de deux sous contextes, *CTContext* représentant le contexte de l'impôt sur les sociétés et *OtherTaxContext* pour illustrer la possibilité d'ajouter d'autres sous contextes. *CTContext* est composé de deux *ContextParameters* : *RatePayer* et *CTDeclaration.* Quatre *ServiceContextElements* ont été déduits du contexte *CTContext, à savoir* : *CTCalculationServiceSC, GetCTRateSC, CTRestitutionProcessSC et CTAdvanceRestitutionSC*. Le premier comporte les paramètres nécessaires à l'adaptation du service *CTCalculationService*.

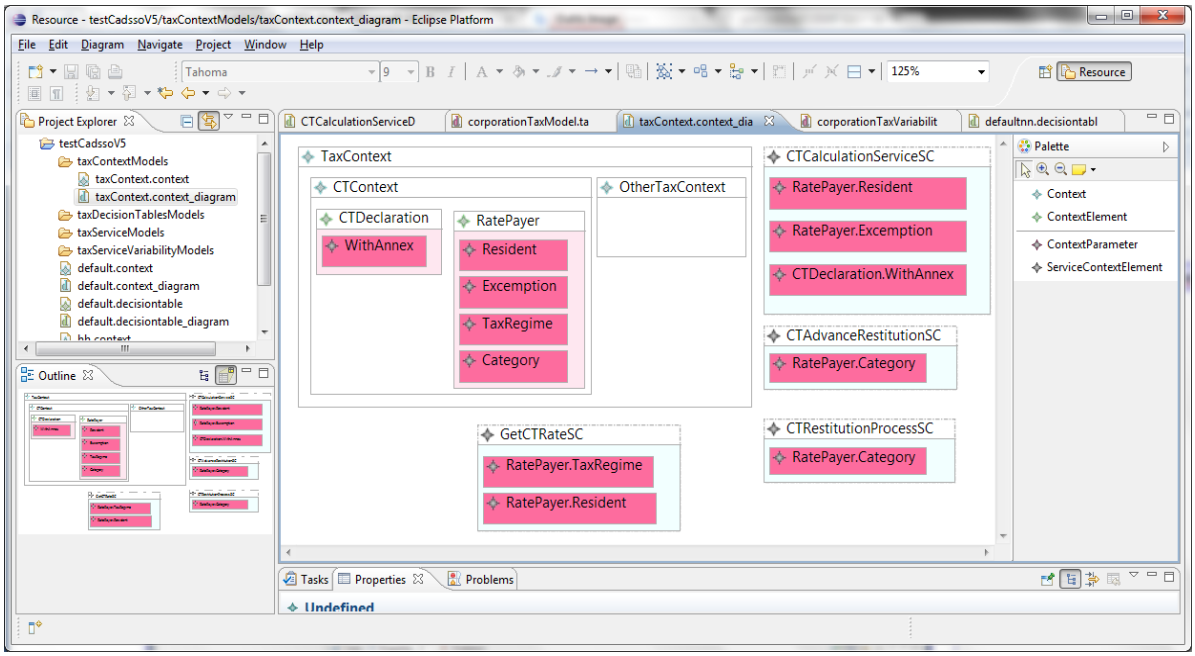


Figure 75 : Modèle du contexte de l'impôt sur les sociétés

Le Tableau 13 met en exergue la correspondance entre la syntaxe abstraite et la syntaxe concrète du langage de modélisation du contexte.

Tableau 13 : Correspondance entre la syntaxe abstraite et la syntaxe concrète du langage de modélisation du contexte

Elément du méta-modèle	Bouton correspondant de la palette de boutons	Description
Context	Context	Représente un conteneur de sous contexte et d'éléments de contexte
ContextElement	ContextElement	Représente un conteneur des paramètres du contexte
ContextParameter	ContextParameter	Modélise une information contextuelle qui influence l'adaptation d'un service
ServiceContextElement	ServiceContextElement	Regroupe les paramètres du contexte qui influencent l'adaptation d'un service

V.3.2.5. Syntaxe concrète du langage de modélisation des règles d'adaptation

Le modèle des règles d'adaptation du service de calcul de l'impôt sur les sociétés *CTCalculationService* conçu en utilisant l'outil de modélisation des règles d'adaptation, est illustré dans la Figure 76. Les conditions sont représentées en utilisant la couleur orange claire et les actions sont en orange foncé.

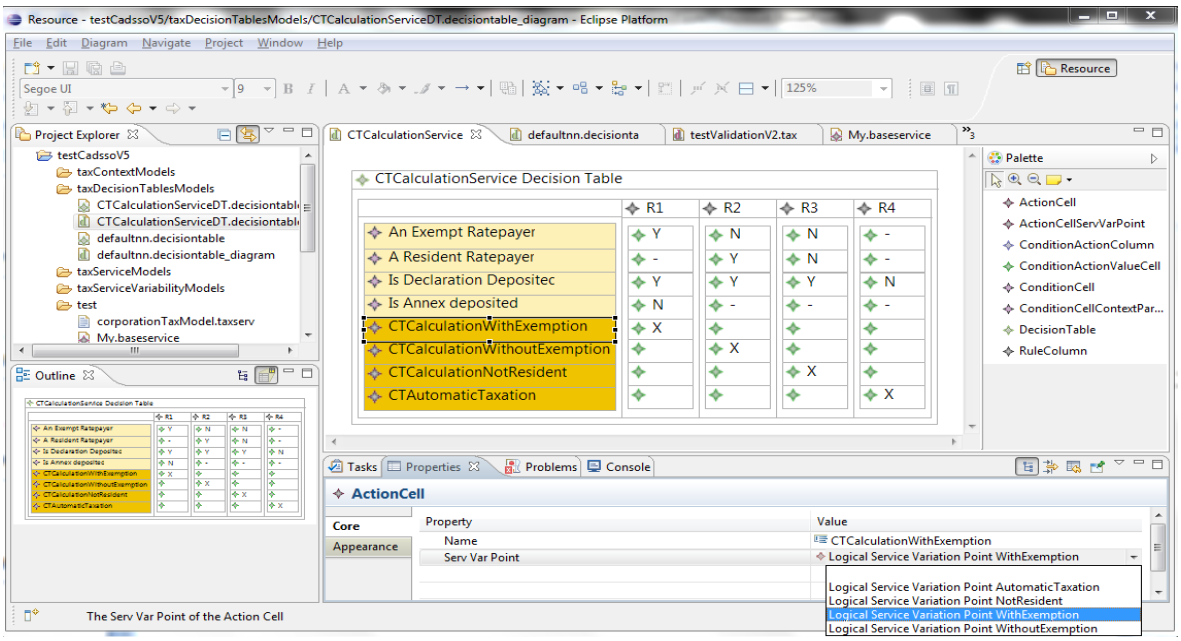


Figure 76 : Modèle des règles d'adaptation du service de calcul de l'impôt sur les sociétés *CTCalculationService*

Comme déjà cité, la table de décision du service de calcul de l'impôt sur les sociétés *CTCalculationService* relie le Modèle du contexte de l'impôt sur les sociétés et le Modèle de la variabilité de service du service *CTCalculationService*. En effet, la propriété « *Context Param* » de l'élément *ConditionCell*, figurant sur la palette de bouton, permet de relier un *ConditionCell* à un *ContextParameter* (Figure 77). En outre, la propriété « *Serv Var Point* »

de l'élément *ActionCell* permet de relier un *ActionCell* à un *ServiceVariationPoint* (Figure 76).

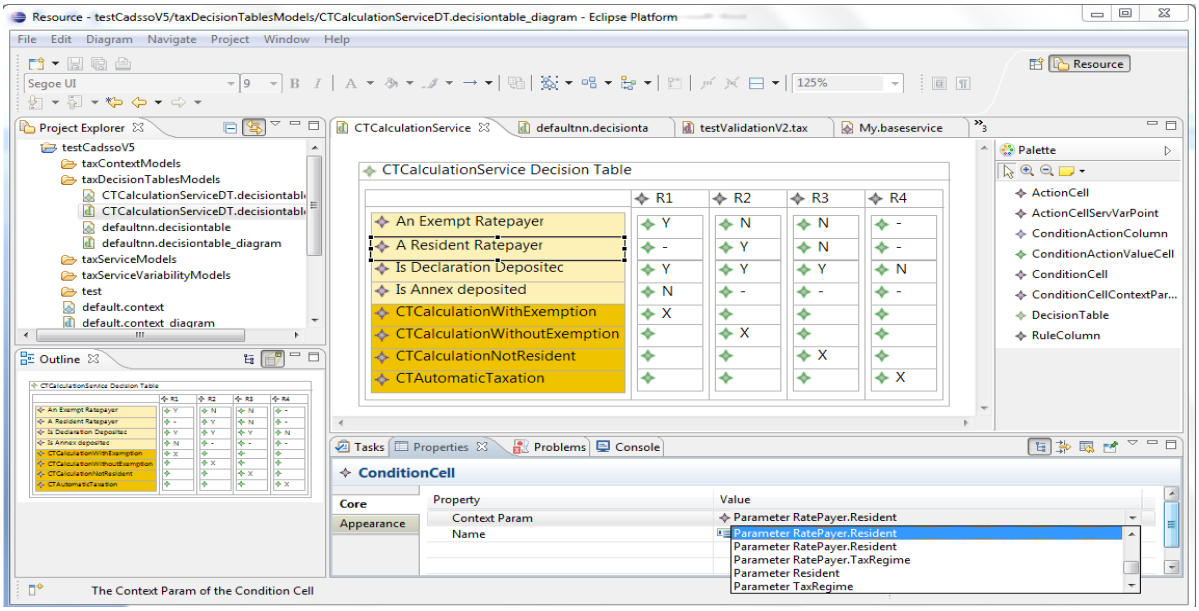


Figure 77 : Liaison d'une *ConditionCell* à un *ContextParameter*

Le Tableau 14 illustre la correspondance entre la syntaxe abstraite et la syntaxe concrète du langage de modélisation des règles d'adaptation. A l'instar de la syntaxe concrète du langage de modélisation du contexte, la syntaxe concrète du langage de modélisation des règles d'adaptation est basée sur l'utilisation des compartiments GMF (*compartment*). Par conséquent, aucune connexion n'est utilisée.

Tableau 14 : Correspondance entre la syntaxe abstraite et la syntaxe concrète du langage de modélisation des règles d'adaptation

Elément du méta-modèle	**Bouton correspondant de la palette de boutons**	**Description**
DecisionTable	*DecisionTable*	Représente la table de décision
ConditionActionColumn	*ConditionActionColumn*	Représente la colonne des conditions/actions
RuleColumn	*RuleColumn*	Représente les colonnes des règles d'adaptation
ConditionCell	*ConditionCell*	Représente les cellules des conditions (orange claire)
ActionCell	*ActionCell*	Représente les cellules des actions (orange foncé)
ConditionActionValueCell	*ConditionActionValueCell*	Représente les autres cellules de la table de décision

V.3.3. Environnement de développement CADSSOTB

Après la génération des cinq EDSDs des DSLs utilisés par notre approche (Figure 69), vient l'étape de leur intégration dans un seul environnement de développement spécifique intitulé CADSSOTB (Figure 78). Les EDSDs produits sont sous forme de plug-ins Eclipse facilement déployables au niveau de la plateforme de base d'Eclipse. Quatre plug-ins sont générés pour chaque DSL : *model*, *edit*, *editor* et *plugin* représentant l'éditeur visuel (Figure 79).

La boite à outils CADSSOTB permet de gérer l'intégration et la dépendance des cinq modèles de l'approche CADSSOMA. Les EDSDs (Figure 78) supportent l'édition et la validation des modèles. La dernière composante de la boite à outils CADSSOTB que nous allons détailler dans la section suivante est le générateur de code source (Figure 78).

ED : Environnement de développement

Figure 78 : Architecture de l'outil support CADSSOTB

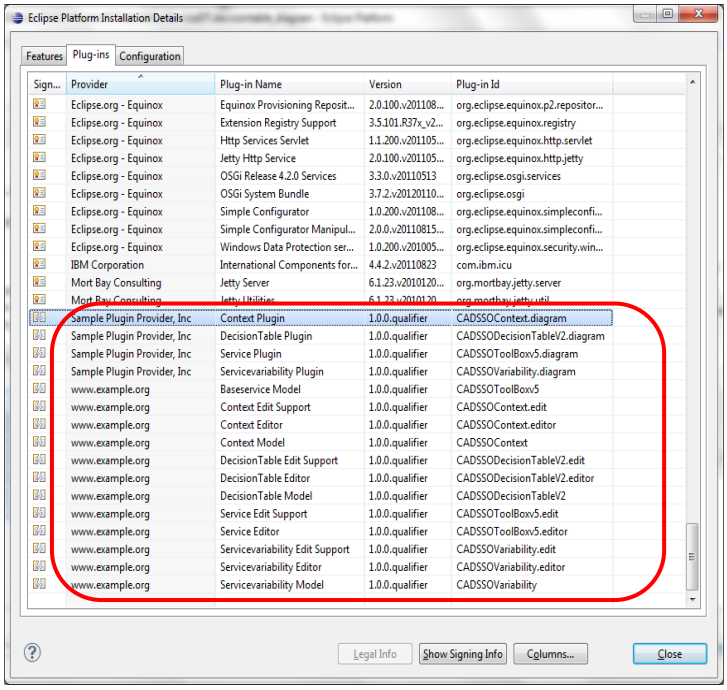


Figure 79 : Plug-ins de l'outil support CADSSOTB

La Figure 80 illustre l'assistant de création d'un modèle de l'approche CADSSOMA. Elle permet soit l'utilisation de l'éditeur arborescent d'EMF soit l'utilisation des éditeurs graphiques spécifiques.

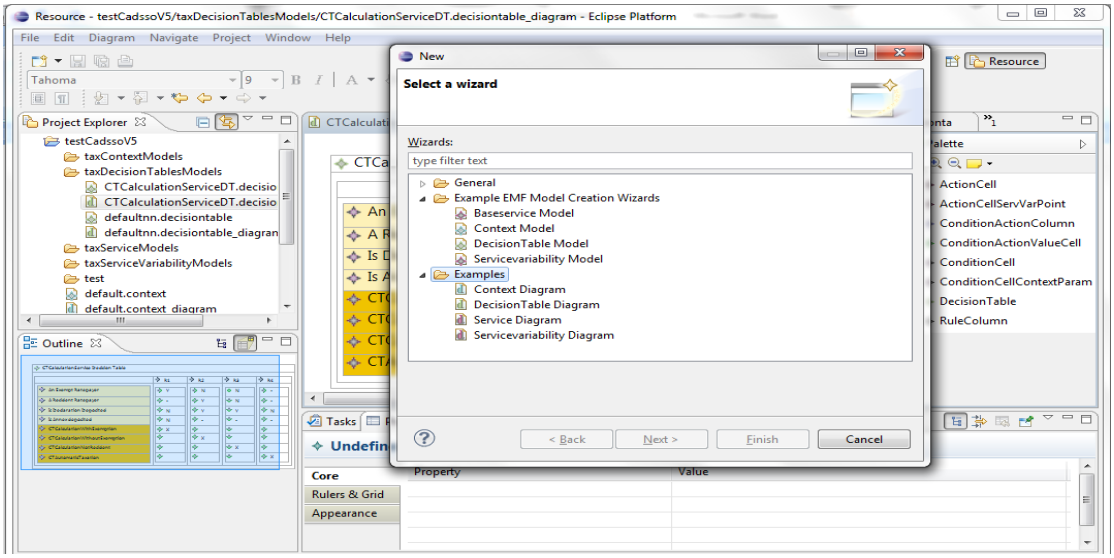


Figure 80 : Assistant de création des modèles de l'approche CADSSOMA

V.3.4. Spécification et développement des règles de conception

La validation des modèles permet de s'assurer de leur conformité aux règles du domaine spécifique. Cela permet d'éviter la génération de code source non conforme. Les contraintes de validation sont intégrées dans les EDSDs produits (Figure 60).

Nous avons utilisé le Framework de validation d'EMF (EMF Validation Framework) (Eclipse, EMP, 2016). Il permet de définir des contraintes au niveau des méta-modèles. Pour ce faire, plusieurs techniques sont offertes, nous citons entre autres le langage de contraintes OCL, les annotations (*EAnnotation* d'Ecore), *XML Schema*, etc.

Trois types de validation sont possibles en utilisant le Framework de validation d'EMF :

- Validation interactive : vérification des règles de validation tout au long du processus de modélisation ;
- Validation manuelle : la validation est déclenchée manuellement après l'achèvement du processus de modélisation, et ce en utilisant l'éditeur arborescent d'EMF (bouton droit sur la racine du modèle puis Validation ou utilisation du menu « *nom_package Editor* ») ;
- Validation lors de l'enregistrement d'un modèle : par défaut EMF n'exécute pas la validation lors de l'enregistrement d'une ressource. Pour ce faire, il faut utiliser le Framework *EMF Validation Builder*.

Pour CADSSOTB, nous avons opté pour l'utilisation d'une validation manuelle en utilisant des invariants (invariant). Un invariant [1] d'une méta-classe est une condition permanente que doit vérifier l'ensemble des modèles conformes à la méta-classe. EMF représente les invariants sous forme d'opérations de méta-classe (*EOperation*). Un invariant EMF doit retourner un *EBoolean* (c'est un *EDataType* du modèle Ecore) indiquant le résultat de la validation. Il possède deux paramètres d'entrée de type *EDiagnosticChain* et *EMap<EObject, EObject>* (Figure 81). Le premier est utilisé par le framework de validation pour agréger les résultats de la validation des invariants, tandis que le deuxième est utilisé comme *cache* des informations qui doivent être accessibles dans les différentes étapes du processus de validation (Steinberg, et al., 2008a). De plus, une *EOperation* sera convertie en une méthode Java et les *EDataTypes* seront transformés aux objets Java *boolean*, *DiagnosticChain* et *Map* (Figure 82). Le processus de validation déclenche l'exécution de l'ensemble des invariants implémentés.

La méthode *validate* a été ajoutée au modèle Ecore de la méta-classe *Service* du méta-modèle des services de calcul d'impôts (Figure 81). L'implémentation de la méthode *validate* est logée au niveau de l'implémentation de la méta-classe *Service ServiceImpl.java* (Figure 82).

[1] Similaire à UML : Un Invariants de classe est une condition qui doit être respectée par toutes les instances d'une classe.

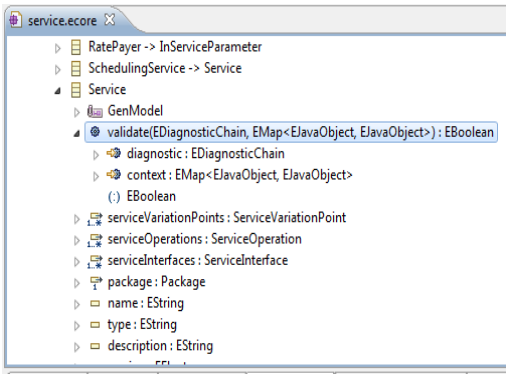


Figure 81 : Ajout de la méthode *validate* à la méta-classe *Service*

ServiceImpl.java

```
     * <!-- begin-user-doc -->
     * <!-- end-user-doc -->
     * @generated NOT
     */
    public boolean validate(DiagnosticChain diagnostic, Map<Object, Object> context) {
        boolean valid = true;
        if (diagnostic != null) {
            if (getName() == null || !getName().startsWith("service")) {
                valid = false;
                diagnostic.add(new BasicDiagnostic(Diagnostic.WARNING,

                        BaseserviceValidator.DIAGNOSTIC_SOURCE,
                        BaseserviceValidator.SERVICE__VALIDATE, "The service name of the service '"
                                + getName() + "' must start with 'service'.",
                        new Object[] { this }));
            }
            if (getType()==null || (!getType().equals("Rest") && !getType().equals("SOAP"))) {
                valid = false;
                diagnostic.add(new BasicDiagnostic(Diagnostic.ERROR,

                        BaseserviceValidator.DIAGNOSTIC_SOURCE,
                        BaseserviceValidator.SERVICE__VALIDATE, "The 'type' attribute  of the service '"
                                + getName() + "' is incorrect .",
                        new Object[] { this }));
            }
            if (getDescription()==null || getDescription().equals("")) {
                valid = false;
                diagnostic.add(new BasicDiagnostic(Diagnostic.WARNING,

                        BaseserviceValidator.DIAGNOSTIC_SOURCE,
                        BaseserviceValidator.SERVICE__VALIDATE, "The 'description' attribute  of the service '"
                                + getName() + "' can not be empty .",
                        new Object[] { this }));
            }
        }
        return valid;
    }
```

Figure 82 : Implémentation de la méthode *validate* de la classe *ServiceImpl.java*

La méthode *validate* de la Figure 82 implémente les contrôles ci-dessous :

- L'attribut *name* de la méta-classe *Service* doit être préfixé par la chaine de caractères « service » ;
- L'attribut *type* doit être égal à *SOAP* ou *Rest* ;
- L'attribut *description* de la méta-classe Service est obligatoire.

La première et la troisième règle sont des avertissements (`WARNING`), la deuxième est une erreur (`ERROR`) (Figure 83).

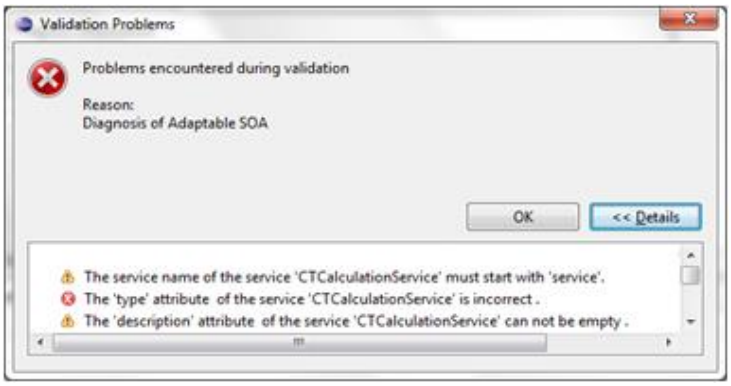


Figure 83 : Exemple du résultat de validation du modèle des services de calcul d'impôts

V.3.5. Transformation des modèles au code source

L'étape de génération de la solution finale permet la transformation des modèles spécifiques des cinq DSLs utilisés au code source final (

Figure 84).

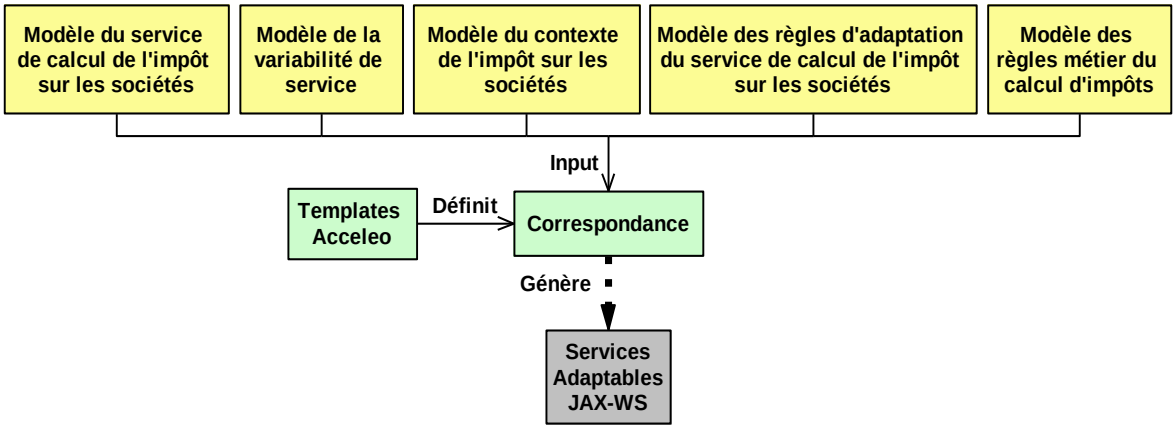


Figure 84 : Génération de la solution finale : Service adaptable de calcul d'impôts

Il est à rappeler que l'approche DSM vise la génération totale de la solution finale en se basant sur des modèles spécifiques. De même, le générateur de code source est un générateur spécifique implémenté spécialement pour une plateforme cible bien déterminée. De ce qui précède découle qu'une transformation Modèle à Modèle est superflue, alors qu'une transformation directe des modèles au code source est suffisante.

La première version de notre générateur de code n'utilise que deux modèles : le modèle des services de calcul d'impôts et le modèle de la variabilité de service. Le code des opérations des services sera généré automatiquement après l'intégration du modèle des règles métier du domaine spécifique, du modèle de contexte et du modèle des règles d'adaptation au niveau du générateur de code.

Nous avons utilisé le langage de transformation Modèle à Texte Acceleo (OBEO, Acceleo, 2016). Il implémente le standard OMG MOF MTL (OMG, MOF, M2T, 2016). Il a intégré le projet de modélisation d'Eclipse en 2009. Acceleo permet de générer des fichiers à partir de modèles UML, MOF, EMF, etc. Ces derniers doivent être sérialisés au format XMI et transformé en utilisant des *Templates* ou des *Chains*. Une *Chain* permet d'enchainer plusieurs générations et opérations sur les modèles. Elles offrent également des possibilités de paramétrage avancées.

La cible de notre générateur de code est l'implémentation *JAX-WS* (Java API for XML Web Services) des services web.

Le Tableau 15 illustre la correspondance entre les éléments des modèles de l'approche CADSSOMA et les éléments de l'API Java JAX-WS.

Tableau 15 : Correspondance entre les éléments des modèles du domaine spécifique calcul et restitution d'impôt et JAX-WS

Modèle	**Eléments des modèles CADSSOMA**	**JAX_WS**
Service	Package	Package
	ServiceInterface	Interface (@WebService)
	TaxCalculation	Class (@WebService)
	ServiceOperation	Method (@Override)
	OutServiceParameter	method return value
	InServiceParameter	method parameter
	RatePayer	Class
	TaxDeclaration	Class
	UtilityService	Class (@WebService)
	Process	Class (@WebService)
Variabilité de service	ServiceVariationPoint	Method

La première version de notre générateur utilise deux Templates Acceleo, le premier pour l'implémentation des services (Figure 85) et le deuxième pour les interfaces des services (*generateInterfaces*, Annexe C Figure 96). Le résultat de notre générateur de code est représenté dans les Figure 86 (voir aussi Figure 97 au niveau de l'annexe C).

```
[template public generateImplementations(anAdaptableSOA : AdaptableSOA)]
[comment service implementations generation /]
[for (s: Service | anAdaptableSOA.services) ]
    [file (s.name.toUpperFirst().concat('.java'), false)]      …
public class [s.name.toUpperFirst()/] implements [for (si: ServiceInterface | s.serviceInterfaces) separator (', ')][si.name/]
[/for]{
            [for (svp: ServiceVariationPoint | s.serviceVariationPoints) ]
                 [for (si: ServiceInterface | s.serviceInterfaces) ]
                      [for (so: ServiceOperation | si.serviceOperations) ]
             @Override
             public [so.outServiceParameter -> at(1).type/] [so.name.concat(svp.name) /](
             [for (insp: InServiceParameter | so.inServiceParameter) separator (', ') ] [insp.type/]      [insp.name/] [/for]
                                 ){
                      [so.outServiceParameter -> at(1).type/] [so.outServiceParameter -> at(1).name/] = 0;
                                                  // business
                      return [so.outServiceParameter -> at(1).name/];
                                 }
                      [/for]
                 [/for]
            [/for]
              }    …
```

Figure 85 : Extrait du template Acceleo "*generateImplementations*"

Une classe Java est créée par service. Chaque variation de service est transformée à une méthode de service qui prend en entrée les paramètres du service *InServiceParameters* et retourne le paramètre de sortie *outServiceParameter.*

```
package ctpackage;
import javax.jws.WebService;
@WebService(endpointInterface = "ctpackage.CTCalculationServiceInterface1")
public class CTCalculationService implements CTCalculationServiceInterface1 , CTCalculationServiceInterface2 {
    @Override
    public double cTaxCalculationAutomaticTaxation( TaxDeclaration cTaxDeclaration1 ,  RatePayer   ratePayer1  ){
             double ctCalculationResult = 0;
              // business
              return ctCalculationResult;
    }
    @Override
    public double cTaxCalculationNotResident(TaxDeclaration cTaxDeclaration1 ,  RatePayer ratePayer1       ){
             double ctCalculationResult = 0;
             // business
             return ctCalculationResult;
     }
     @Override
     public double cTaxCalculationWhithoutExemption(TaxDeclaration cTaxDeclaration1 ,  RatePayer ratePayer1 ){
             double ctCalculationResult = 0;
             // business
               return ctCalculationResult;
     }
     @Override
     public double cTaxCalculationWithExemption(TaxDeclaration cTaxDeclaration1 ,  RatePayer ratePayer1 ){
             double ctCalculationResult = 0;
              // business
            return ctCalculationResult;
     }
}
```

Figure 86 : Extrait du résultat de la génération de code source du template Acceleo *« generateImplementations »*

V.4. Conclusion

Au niveau de ce chapitre nous avons décrit le processus de mise en place de l'Environnement de Développement Dirigé par les Modèles utilisé par notre approche CADSSOMA. CADSSOTB est une boite à outils complète qui permet de modéliser les cinq modèles de la phase de modélisation, de gérer les dépendances des modèles, de les valider et de les transformer au code source final.

La mise en place d'une boite à outils sans l'utilisation d'un outil support est une tâche très couteuse en matière de temps et d'effort (Achilleos, 2009) (Kelly, et al., 2008a). Par conséquent, l'utilisation d'un outil de méta-modélisation est une obligation plus qu'un choix. Une étude comparative des outils de méta-modélisation nous a permis d'adopter le Projet de Modélisation Eclipse (PME) qui repose sur l'IDE extensible Eclipse. Les outils proposés par ce projet permettent de modéliser la sémantique d'un domaine spécifique sous forme d'un méta-modèle appelé aussi syntaxe abstraite. Ils permettent également de représenter cette dernière en utilisant une syntaxe concrète, graphique ou textuelle. Ils assurent aussi la génération d'un Environnement de Développement Spécifique au Domaine (EDSD).

La syntaxe concrète des DSLs utilisés est une syntaxe graphique. En effet, nous avons utilisé des notations visuelles qui représentent les différents concepts des cinq langages utilisés. Dans ce sens, une conversion de la syntaxe abstraite vers une syntaxe concrète graphique est proposée.

La boite à outils CADSSOTB est un ensemble d'EDSDs générés en utilisant le framework GMF et intégrés dans la plateforme de base Eclipse. CADSSOTB intègre aussi un générateur de code source qui permet de transformer les cinq modèles spécifiques au code source final.

Chapitre VI

Conclusion générale

VI.1. Résumé de la contribution de la thèse

Le principal objectif de cette thèse est de proposer une approche DSM pour le développement des Systèmes Orientés Services Adaptables (SOSA). Ce type de système repose essentiellement sur les deux paradigmes : SOC (Service Oriented Computing) et CAC (Context-Aware Computing). Le premier vise la réorganisation du système d'information sous forme d'un ensemble de services faiblement couplés, cohésifs et réutilisables. Le deuxième traite l'intégration de la capacité d'adaptation aux systèmes informatiques.

Dans cette visée, nous avons proposé une approche, intitulée CADSSOMA (***C****ontext-****A****ware, ****D****omain ****S****pecific and ****S****ervice ****O****riented ****M****odeling ****A****pproch*), qui définit principalement : (i) Cinq DSLs pour l'étape de modélisation ; (ii) un processus de développement ; et (iii) un outil logiciel facilitant l'édition des modèles et la génération automatique de code source.

La séparation des préoccupations représente la pierre angulaire de la flexibilité et de la gestion de la complexité des systèmes. Dans cette optique, notre approche assure une séparation entre la variabilité de service, le contexte et les règles d'adaptation. La phase de modélisation est basée sur cinq modèles : (i) le modèle des services spécifiques au domaine, il illustre l'ensemble des services du système cible ; (ii) le modèle de la variabilité des services, prenant en charge la modélisation des variabilités des services ; (iii) le modèle du contexte, permettant de modéliser le contexte d'exécution des services ; (iv) le modèle des règles d'adaptation, il relie le modèle du contexte et le modèle de la variabilité de services ; et finalement (v) le modèle des règles métier du domaine spécifique, il est responsable de la modélisation de la logique métier du domaine spécifique.

En général, pour un domaine spécifique donné, il faut produire le méta-modèle (syntaxe abstraite) du métier du domaine spécifique, et le méta-modèle des services spécifiques au domaine en héritant de notre méta-modèle générique des services.

Afin d'initier les développeurs à l'ingénierie des langages spécifiques au domaine, nous avons jugé utile de proposer un formalisme du processus de développement des systèmes spécifiques au domaine, regroupant l'ensemble des phases, activités et artefacts. Ce formalisme est basé sur la documentation étudiée traitant l'ingénierie des langages spécifiques au domaine.

Nous avons aussi proposé un processus DSM pour le développement des SOSA associé à notre approche. Il définit les phases, les activités et les artefacts nécessaires pour la transformation du métier d'un domaine spécifique en des services flexibles et adaptables.

Finalement, nous avons validé notre approche en proposant un environnement de développement intégré (*CADSSO**TB, T**ool**B**ox*) qui facilite la mise en œuvre de l'approche CADSSOMA. Après la spécification graphique des cinq modèles de la phase de modélisation, la boite à outils s'occupe de la génération automatique de la solution finale en se basant sur une seule transformation Modèle-à-Texte. L'étude de cas utilisée correspond au domaine de calcul et restitution d'impôts.

VI.2. Travaux en cours et perspectives

VI.2.1. Optimisation des règles d'adaptation

Nous nous sommes basés dans la modélisation des règles d'adaptation des SOSAs sur le concept de table de décision. L'optimisation des règles d'adaptation est un point qui nous intéresse. A ce sujet, nous projetons l'exploitation de la théorie mathématique des tables de décision : Elimination des redondances de règles, Vérification de la complétion, Minimisation, etc. Nous prévoyons l'ajout d'une nouvelle étape prenant en charge ces optimisations. Elle est à intégrer dans la boite à outils CADSSOTB avant l'étape de génération du code source.

VI.2.2. Utilisation de l'adaptation dynamique

L'adaptation traitée par notre proposition est une adaptation du type « *designed* », elle nécessite l'analyse de toutes les possibilités d'adaptation lors de la phase de conception du système. Nous jugeons que l'utilisation de l'adaptation du type « *at run-time* » ou adaptation dynamique ouvrira de nouvelles perspectives aux travaux réalisés. Nous comptons en particulier explorer les possibilités offertes par d'autres théories de décision en ce qui concerne la modélisation mathématique des règles d'adaptation, comme par exemple la théorie des jeux et le processus de décision de Markove.

VI.2.3. Mesure de l'apport de l'approche DSM

Afin de mettre en exergue les avantages de l'utilisation de l'approche DSM par rapport à l'approche manuelle, il est préférable de faire une comparaison chiffrée sur deux niveaux :

- Le premier concerne le code généré : une comparaison empirique basée sur la métrique « LoC[1] » entre l'approche DSM et l'approche manuelle est envisageable (Biehl, et al., 2014). D'autres métriques de la qualité logicielle peuvent être utilisées comme par exemple la complexité cyclomatique et sa relation avec la consommation des ressources matérielles.
- Le deuxième niveau s'intéresse à l'effort déployé pour produire la solution finale (nombre de jour-homme). Cela permet de mesurer le retour sur investissement de la production de la boite à outils DSM (langage, environnement de développement et générateur du code source) (Kärnä, et al., 2009). De toute évidence, il faut tenir compte des développements futurs en utilisant la solution DSM produite. Dans ce sens, nous prévoyons l'utilisation du modèle de coût logiciel COnstructive COst MOdel (COCOMO) qui permet d'estimer l'effort à déployer dans un développement logiciel (COCOMO II, 2000).

VI.2.4. Amélioration de l'outil support CADSSOTB

En ce qui concerne la boite à outils CADSSOTB, nous comptons :

- La compléter par l'implémentation de la syntaxe concrète du domaine spécifique étudié, à savoir « calcul et restitution d'impôts ». De même, la relation entre la variabilité de service est le modèle du métier du domaine est à mettre en œuvre ;
- Améliorer le générateur de code source en intégrant le modèle du contexte, le modèle des règles d'adaptation et le modèle du calcul et restitution d'impôts. L'implémentation des règles d'adaptation sera sous forme de feuilles de règles à exploiter par un moteur de règles. La réalisation du premier point est une condition sine-qua-none à la génération totale du code source. Nous comptons aussi permettre la génération des interfaces graphiques permettant l'utilisation des services générés (probablement en utilisant un sixième DSL).
- Etudier la possibilité d'utiliser l'outil de méta-modélisation Sirius (OBEO, Sirius, 2015) à la place de GMF[2]. En effet, Sirius suscite un intérêt grandissant de la part de la

[1] Lignes de Code, *Lines of Code* dans le vocabulaire anglo-saxon.
[2] Le mois 10/2016 GMF possédait 4 développeurs ; la dernière validation date du moins 12/2015.

communauté de recherche spécialisée dans l'ingénierie des langages spécifiques au domaine. C'est un projet open source de la société Obeo avec une communauté très active[1].

VI.3. Liste des publications

Cette section liste par ordre chronologique descendant, les publications effectuées au cours de notre travail de recherche.

VI.3.1. Revues internationales

1- M. Lethrech, A. Kenzi, I. Elmagrouni, M. Nassar and A. Kriouile, A MDSD Approach for Adaptable Service Oriented Systems based on Domain Specific Language Engineering. *International Journal of Adaptive, Resilient and Autonomic Systems (IJARAS)*, IGI-Global, 2016, vol. 7, no 1, p. 1-25.

2- M. Lethrech, A. Kenzi, I. Elmagrouni, M. Nassar and A. Kriouile, CADSSO: A Development Approach for Context-Aware Service Oriented Systems. *Journal of Theoretical and Applied Information Technology*, 2015, vol. 78, no 2, p. 236.

3- I. Elmagrouni, A. Kenzi, M. Lethrech, M. Nassar and A. Kriouile, A development process for adaptable services-oriented systems. *International Review on Computers and Software*, 2015, vol. 10, no. 7.

4- I. Elmagrouni, M. Lethrech, A. Kenzi, and A. Kriouile, An approach for the Development of Adaptable Services-Oriented Systems. *Mediterranean Telecommunication Journal*, Vol. 5, N° 2, June 2015.

VI.3.2. Conférences internationales

1- I. Elmagrouni, A. Kenzi, M. Lethrech, and A. Kriouile, Adaptation services-oriented systems lifecycle, *Proceedings of the 18th International Conference on Enterprise Information Systems*, ICEIS, Rome, Italy, 2016.

2- M. Lethrech, A. Kenzi, I. Elmagrouni, M. Nassar and A. Kriouile, A process definition for domain specific software development. In: *2015 Third World Conference on Complex Systems (WCCS)*. IEEE, Marrakech, 2015. p. 1-7.

3- M. Lethrech, I. Elmagrouni, A. Kenzi, M. Nassar and A. Kriouile, Domain Specific Modeling approach for context-aware service oriented systems. In: *2014 International Conference on Multimedia Computing and Systems (ICMCS)*. IEEE, Marrakech, 2014. p. 575-581.

[1] Le mois 10/2016 Sirius possédait 18 développeurs ; la dernière validation date du 10/2016.

4- M. Lethrech, I. Elmagrouni, A. Kenzi, M. Nassar and A. Kriouile, A generic metamodel for adaptable service oriented systems modeling using DSM approach. In: *2013, 3rd International Symposium, ISKO-Maghreb*. IEEE, Marrakech, 2013. p. 1-6.

5- I. Elmagrouni, M. Lethrech, A. Kenzi, and A. Kriouile, Process 3TUP. In: *2013 3rd International Symposium, ISKO-Maghreb*. IEEE, Marrakech, 2013. p. 1-7.

VI.3.3. Workshops

1- M. Lethrech, A. Kenzi, I. Elmagrouni, M. Nassar and A. Kriouile, A Modeling Approach for Adaptable Service Oriented Systems based on Domain Specific Language Engineering. In: *International Workshop on new wireless technologies and Distributed Systems (WITS 2014)*. Fès, 2014.

2- M. Lethrech, I. Elmagrouni, A. Kenzi, M. Nassar and A. Kriouile, DSL et SOA: une Etude Exploratoire. In: *Les 4èmes Journées Doctorales en Technologies de l'Information et de la Communication* (JDTIC), Casablanca, 2012.

Références bibliographiques

Abu-Matar, M., & Gomaa, H. (2011a). Feature based variability for service oriented architectures. In IEEE (Ed.), *9th Working IEEE/IFIP Conference on Software Architecture (WICSA)*, (pp. 302-309).

Abu-Matar, M., & Gomaa, H. (2011b). Variability modeling for service oriented product line architectures. In IEEE (Ed.), *15th International Software Product Line Conference (SPLC)* (pp. 110-119). IEEE.

Achilleos, A. (2009). *Model-driven petri net based framework for pervasive service creation.* Thèse de doctorat, University of Essex.

AndroMDA. (2015). Retrieved 12 25, 2016, from http://andromda.org/

apache.org. (2016). *http://cxf.apache.org/*. Retrieved from http://cxf.apache.org/

Appeltauer, M., Hirschfeld, R., Haupt, M., & Masuhara, H. (2011). ContextJ: Context-oriented programming with Java. *Information and Media Technologies* , 399-419.

Arsanjani, A. (2004). *Service-oriented modeling and architecture.* Retrieved from https://www.ibm.com/developerworks/library/ws-soa-design1/

Arsanjani, A., Ghosh, S., Allam, A., Abdollah, T., Ganapathy, S., & Holley, K. (2008). SOMA: A method for developing service-oriented solutions. *IBM systems Journal , 47* (3), 377-396.

ATL. (2016). *ATL*. Retrieved 2016, from http://www.eclipse.org/atl/

Ayed, D., Delanote, D., & Berbers, Y. (2007). MDD approach for the development of context-aware applications. In S. B. Heidelberg (Ed.), *International and Interdisciplinary Conference on Modeling and Using Context* (pp. 15-28). Springer Berlin Heidelberg.

Baidouri, H., Hafiddi, H., Nassar, M., & Kriouile, A. (2012). Towards a context-aware composition of services. *International Journal of Computer Science and Network Security (IJCSNS) , 12* (3), 133.

Baldauf, M., Dustdar, S., & Rosenberg, F. (2007). A survey on context-aware systems. *International Journal of Ad Hoc and Ubiquitous Computing , 2* (4), 263-277.

BAREIL, C. (2004). *LA RÉSISTANCE AU CHANGEMENT : SYNTHÈSE ET CRITIQUE DES ÉCRITS.* HEC Montreal, Centre d'études en transformation des organisations .

Batory, D. (2006). A tutorial on feature oriented programming and the ahead tool suite. (S. B. Heidelberg, Ed.) *Generative and Transformational Techniques in Software Engineering* , 3-35.

Berg, H., Møller-Pedersen, B., & Krogdahl, S. (2011). Advancing generic metamodels. In ACM (Ed.), *Proceedings of the compilation of the co-located workshops on DSM'11, TMC'11, AGERE'11, AOOPES'11, NEAT'11, & VMIL'11* (pp. 19-24). ACM.

Bettini, C., Brdiczka, O., Henricksen, K., Indulska, J., Nicklas, D., Ranganathan, A., et al. (2010). A survey of context modelling and reasoning techniques. *Pervasive and Mobile Computing , 6* (2), 161-180.

Beydoun, G., Low, G., Henderson-Sellers, B., Mouratidis, H., Gomez-Sanz, J. J., Pavon, J., et al. (2009). FAML: a generic metamodel for MAS development. *IEEE Transactions on Software Engineering , 35* (6), 841-863.

Bharadwaj, R., & Mukhopadhyay, S. (2007). SOLj: a domain-specific language (DSL) for secure Service-Based systems. In IEEE (Ed.), *11th IEEE International Workshop on Future Trends of Distributed Computing Systems (FTDCS'07)* (pp. 173-180). IEEE.

Biehl, M., El-Khoury, J., Loiret, F., & Törngren, M. (2014). On the modeling and generation of service-oriented tool chains. *Software & Systems Modeling , 13* (2), 461-480.

Bierhoff, K., Liongosari, E. S., & Swaminathan, K. S. (2006). Incremental Development of a Domain-Specific Lan-guage That Supports Multiple Application Styles. *OOPSLA 6th Workshop on Domain Specific Modeling*, (pp. 67-78).

Black, A. P. (2013). Object-oriented programming: Some history, and challenges for the next fifty years. *Information and Computation , 231*, 3-20.

Boffoli, N., Cimitile, M., Maggi, F. M., & Visaggio, G. (2009). Managing SOA system variation through business process lines and process oriented development. *Workshop on Service-Oriented Architectures and Software Product Lines (SOAPL)*, (pp. 61-68).

Bonnet, S. (2005). *Une démarche dirigée par les modèles pour la personnalisation des applications embarquées dans les cartes à puce.* Thèse de doctorat, UNIVERSITÉ DE LILLE 1.

Booch, G., Brown, A. W., Iyengar, S., Rumbaugh, J., & Selic, B. (2004). An MDA manifesto. *MDA Journal* .

Boukadi, K. (2009). *Coopération interentreprises à la demande: Une approche flexible à base de services adaptables.* Thèse de doctorat, Ecole Nationale Supérieure des Mines de Saint-Etienne; Université Jean Monnet-Saint-Etienne.

Bucchiarone, A., Cappiello, C., Di Nitto, E., Kazhamiakin, R., Mazza, V., & Pistore, M. (2010b). Design for adaptation of service-based applications: main issues and requirements. In S. B. Heidelberg (Ed.), *Service-Oriented Computing. ICSOC/ServiceWave 2009 Workshops* (pp. 467-476). Springer Berlin Heidelberg.

Bucchiarone, A., Kazhamiakin, R., Cappiello, C., Di Nitto, E., & Mazza, V. (2010a). A context-driven adaptation process for service-based applications. In ACM (Ed.), *Proceedings of the 2nd International Workshop on Principles of Engineering Service-Oriented Systems* (pp. 50-56). ACM.

Bucchiarone, A., Marconi, A., Mezzina, C. A., Pistore, M., & Raik, H. (2013). On-the-fly adaptation of dynamic service-based systems: incrementality, reduction and reuse. In S. B. Heidelberg (Ed.), *International Conference on Service-Oriented Computing*, (pp. 146-161).

Cai, X., Lyu, M. R., Wong, K. F., & Ko, R. (2000). Component-based software engineering: technologies, development frameworks, and quality assurance schemes. *In Seventh Asia-Pacific Software Engineering Conference, APSEC* (pp. 372-379). IEEE.

Cameron, K. (2008). *DSL + UML = Pragmatic Modeling*. (Microsoft) Retrieved 11 25, 2016, from http://blogs.msdn.com/b/camerons/archive/2008/06/25/dsl-uml-pragmatic-modeling.aspx

Canal, C., Murillo, J. M., & Poizat, P. (2006). Software Adaptation. *L'objet WCAT'04*, *12*, pp. 9-31.

Cardozo, N., Günther, S., D'Hondt, T., & Mens, K. (2011). Feature-oriented programming and context-oriented programming: Comparing paradigm characteristics by example implementations. *Intl. Conf. on Software Engineering Advances IARIA.*

Casati, F., & Shan, M.-C. (2002). Event-Based Interaction Management for Composite E-Services in eFlow. *Information Systems Frontiers , 4* (1), 19-31.

Caseau, Y. (2011). *Urbanisation, SOA et BPM-4e éd.: Le point de vue du DSI.* (Dunod, Ed.)

Cauvet, C., & Guzelian, G. (2008). Business Process Modeling: a Service-Oriented Approach,. *41st Hawaii International Conference on System SciencesHawaii.*

Chang, S. H., & Kim, S. D. (2007a). A service-oriented analysis and design approach to developing adaptable services. In IEEE (Ed.), *IEEE International Conference on Services Computing (SCC 2007)*, (pp. 204-211).

Chang, S. H., La, H. J., & Kim, S. D. (2007b). A comprehensive approach to service adaptation. In IEEE (Ed.), *IEEE International Conference on Service-Oriented Computing and Applications (SOCA'07)* (pp. 191-198). IEEE.

Chen, G., & Kotz, D. (2000). *A survey of context-aware mobile computing research.* Technical Report TR2000-381, Dept. of Computer Science, Dartmouth College.

Chen, K., Sztipanovits, J., & Neema, S. (2004). *Towards Formalizing Domain-specific Modeling Languages.* Presented at the Object Management Group (OMG) OMG's First Annual Model-Integrated, Vanderbilt University Institute for Software Integrated Systems.

Clotet Martínez, R., Dhungana, D., Franch Gutiérrez, J., Grünbacher, P., López Cuesta, L., Marco Gómez, J., et al. (2008). *Dealing with changes in service-oriented computing through integrated goal and variability modelling.* ICB-Research report, (22) 43-52.

COCOMO II. (2000). *COCOMO II*. (Center for Systems and Software Engineering) Retrieved 2016, from http://csse.usc.edu/csse/research/COCOMOII/cocomo_main.html

Dalgarno, M., & Fowler, M. (2008). UML vs. domain-specific languages. *Methods and Tools , 16* (2), 2-8.

David, P. C. (2005). *Développement de composants Fractal adaptatifs: un langage dédié à l'aspect d'adaptation.* Thèse de doctorat, Université de Nantes.

Davis, J. (2009). *Open source SOA.* Manning Publications Co.

Dey, A. K. (2001). Understanding and using context. *Personal and ubiquitous computing , 5* (1), 4-7.

Dhungana, D., Grünbacher, P., & Rabiser, R. (2007). Domain-specific adaptations of product line variability modeling. In S. US (Ed.), *Situational Method Engineering: Fundamentals and Experiences* (pp. 238-251). Springer US.

Diaw, S., Lbath, R., & Coulette, B. (2008). Etat de l'art sur le développement logiciel dirigé par les modèles. *Technique et Science Informatique TSI , 29*, 505-536.

Dijkstra, E. (1968). Go To Statement Considered Harmful. (ACM, Ed.) *Communications of the ACM , 11* (3), 147-148.

DOME. (2012). *DOME*. Retrieved 2016, from http://dome.ggrossmann.com/

DSM Forum. (2016). *DSM Forum*. Retrieved 11 25, 2016, from DSM Forum: http://www.dsmforum.org/

Eclipse. (2016). *Eclipse*. Retrieved 2016, from https://eclipse.org/

Eclipse, EMP. (2016). *Eclipse Modeling Project*. Retrieved 2016, from https://eclipse.org/modeling

Eclipse, help. (2016). *Eclipse documentation*. Retrieved 12 25, 2016, from http://help.eclipse.org/neon/index.jsp

ElAsri, B. (2005). *Vers des composants multivues distribués.* Thèse de doctorat, ENSIAS.

Esfahani, F. S., Murad, M. A., Sulaiman, M. N., & Udzir, N. I. (2011). Adaptable decentralized service oriented architecture. *Journal of Systems and Software , 84* (10), 1591-1617.

Gray, J., Fisher, K., Consel, C., Karsai, G., Mernik, M., & Tolvanen, J. P. (2008). DSLs: the good, the bad, and the ugly. In ACM (Ed.), *Companion to the 23rd ACM SIGPLAN conference on Object-oriented programming systems languages and applications* (pp. 791-794). ACM.

Gronback, R. C. (2009). *Eclipse modeling project: a domain-specific language (DSL) toolkit.* Pearson Education.

Hafiddi, H. (2012). *Ingénierie Dirigée par les Modèles des Systèmes Orientés Services Sensibles au Contexte.* Thèse de doctorat, Université Mohamed V- Souissi Rabat Ecole Nationale Supérieure d'Informatique et d'Analyse des Systèmes.

Hair, A., El Asri, B., Kriouile, A., & Coulette, B. (1998). Outil support de la méthode VBOOM, Fonctionnalités Fusion et Génération du code. *4ème Colloque Africain sur la Recherche en Informatique (CARI'98),* (pp. 497-508). Dakar, Sénégal.

He, X., Fu, Y., Sun, C. A., Ma, Z., & Shao, W. (2015). Towards Model-Driven Variability-Based Flexible Service Compositions. In IEEE (Ed.), *39th Annual Computer Software and Applications Conference (COMPSAC). 2*, pp. 298-303. IEEE.

Hirschfeld, R. (2011). *Context-Oriented Programing Project*. Retrieved 11 25, 2016, from https://www.hpi.uni-potsdam.de/hirschfeld/trac/Cop/wiki/ContextJ

Hirschfeld, R., Costanza, P., & Nierstrasz, O. (2008). Context-oriented programming. *Journal of Object Technology , 7* (3).

Hofer, T., Schwinger, W., Pichler, M., Leonhartsberger, G., Altmann, J., & Retschitzegger, W. (2003). Context-awareness on mobile devices-the hydrogen approach. In IEEE (Ed.), *Proceedings of the 36th Annual Hawaii International Conference on System Sciences* (p. 10). IEEE.

Ibrahim, D., & Misic, V. B. (2006). Service views: a coherent view model of the SOA in the enterprise. In IEEE (Ed.), *IEEE International Conference on Services Computing (SCC'06)* (pp. 230-237). IEEE Computer Society.

Iseger, M. (2006). Retrieved 11 25, 2016, from QPR Software Company: softwarechimps.blogspot.com/2006/01/uml-profiles-vs-dsl.html

Iseger, M. (2010). *Domain-specific modeling for generative software development.* Retrieved 11 25, 2016, from http://www.developerfusion.com/article/84844/domainspecific-modeling-for-generative-software-development/

ISIS, MIC. (2011). *Model Integrated Computing*. Retrieved 11 25, 2016, from Institute for Software Integrated Systems: http://www.isis.vanderbilt.edu/research/MIC

JCAF. (2009). *JCAF The Java Context-Awareness Framework*. (AARHUS universitet) Retrieved 11 25, 2016, from http://www.daimi.au.dk/~bardram/jcaf/

Jones, C. (1996). *Programming Languages Table, Release 8.2.* Software Productivity Research, Burlington, MA.

Josuttis, N. M. (2007). *SOA in practice: the art of distributed system design.* O'Reilly Media, Inc.

Kabbaj, M. I. (2009). *Gestion des déviations dans la mise en œuvre des procédés logiciel.* Thèse de doctorat, Université Toulouse 2.

Kärnä, J., Tolvanen, J. P., & Kelly, S. (2009). Evaluating the use of domain-specific modeling in practice. *Proceedings of the 9th OOPSLA workshop on Domain-Specific Modeling.*

Kazhamiakin, R., Benbernou, S., Baresi, L., Plebani, P., Uhlig, M., & Barais, O. (2010). Adaptation of service-based systems. (S. B. Heidelberg, Ed.) *Service research challenges and solutions for the future internet* , 117-156.

Kelly, S. (2008b). *Microsoft DSLs + UML = ???* Retrieved 11 25, 2016, from http://www.metacase.com/blogs/stevek/blogView?showComments=true&entry=3396256561

Kelly, S., & Pohjonen, R. (2009a). Worst practices for domain-specific modeling. *IEEE software , 26* (4), 22-29.

Kelly, S., & Tolvanen, J. P. (2008a). *Domain-specific modeling: enabling full code generation.* (J. W. Sons, Ed.)

Kelly, S.; MetaCase. (2009b). Domain-Specific Modeling 76 cases of MDD that works.

Kenzi, A. (2010). *Ingénierie des Systèmes Orientés Services Adaptables : Une Approche Dirigée par les Modèles.* Thèse de doctorat, Université Mohamed V- Souissi Rabat Ecole Nationale Supérieure d'Informatique et d'Analyse des Systèmes.

Kieburtz, R. B., McKinney, L., Bell, J. M., Hook, J., Kotov, A., Lewis, J., et al. (1996). A software engineering experiment in software component generation. In I. C. Society (Ed.), *Proceedings of the 18th international conference on Software engineering (pp.* (pp. 542-552). IEEE Computer Society.

Kim, S. D., Her, J. S., & Chang, S. H. (2005). A theoretical foundation of variability in component-based development. *Information and Software Technology , 47* (10), 663-673.

Kim, Y., & Doh, K. G. (2007). The service modeling process based on use case refactoring. In S. B. Heidelberg (Ed.), *International Conference on Business Information Systems* (pp. 108-120). Springer Berlin Heidelberg.

Kim, Y., & Doh, K. G. (2013). Use-case driven service modelling with XML-based tailoring for SOA. *International Journal of Web and Grid Services , 9* (1), 35-53.

Kriouile, A. (1995). *VBOOM, une méthode orientée objet d"analyse et de conception par points de vue.* Thèse de doctorat, Université Mohammed V de Rabat.

Kumar, B. V., Narayan, P., & Ng, T. (2009). *Implementing SOA Using Java EE.* Pearson Education.

La, H. J., & Kim, S. D. (2009). A systematic process for developing high quality saas cloud services. In S. B. Heidelberg (Ed.), *IEEE International Conference on Cloud Computing* (pp. 278-289). Springer Berlin Heidelberg.

Lane, S., Bucchiarone, A., & Richardson, I. (2012). SOAdapt: A process reference model for developing adaptable service-based applications. *Information and Software Technology , 54* (3), 299-316.

Langlois, B., Jitia, C. E., & Jouenne, E. (2007). DSL classification. *OOPSLA 7th workshop on domain specific modeling.*

Lethrech, M., Elmagrouni, I., Kenzi, A., Nassar, M., & Kriouile, A. (2014). Domain Specific Modeling approach for context-aware service oriented systems. In IEEE (Ed.), *International Conference on Multimedia Computing and Systems (ICMCS)* (pp. 575-581). IEEE.

Lethrech, M., Elmagrouni, I., Nassar, M., Kriouile, A., & Kenzi, A. (2013). A generic metamodel for adaptable service oriented systems modeling using DSM approach. In IEEE (Ed.), *3rd International Symposium ISKO-Maghreb,* (pp. 1-6).

Lethrech, M., Kenzi, A., Elmagrouni, I., Nassar, M., & Kriouile, A. (2016). A MDSD Approach for Adaptable Service Oriented Systems based on Domain Specific Language Engineering. (I. Global, Ed.) *International Journal of Adaptive, Resilient and Autonomic Systems (IJARAS) , 7* (1), 1-25.

Lethrech, M., Kenzi, A., Elmagrouni, I., Nassar, M., & Kriouile, A. (2014). A Modeling Approach for Adaptable Service Oriented Systems based on Domain Specific Language Engineering. *In International Workshop on new wireless technologies and Distributed Systems.*

Lethrech, M., Kenzi, A., Elmagrouni, I., Nassar, M., & Kriouile, A. (2015). A process definition for domain specific software development. In IEEE (Ed.), *Third World Conference on Complex Systems (WCCS)* (pp. 1-7). IEEE.

Lethrech, M., Kenzi, A., Elmagrouni, I., Nassar, M., & Kriouile, A. (2015). CADSSO: A DEVELOPMENT APPROACH FOR CONTEXT-AWARE SERVICE ORIENTED SYSTEMS. *Journal of Theoretical and Applied Information Technology , 78* (2), 236.

Lethrech, M., Magrouni, I., Kenzi, A., Nassar, M., & Kriouile, A. (2012). DSL and SOA: an Explorative Study. *Proc. JDTIC.*

Liu, S. H., Cardenas, A., Xiong, X., Mernik, M., Bryant, B. R., & Gray, J. (2010). Can domain-specific languages be implemented by service-oriented architecture? In ACM (Ed.), *Proceedings of the 2010 ACM Symposium on Applied Computing* (pp. 2491-2492). ACM.

Long, E., Misra, A., & Sztipanovits, J. (1998). Increasing productivity at Saturn. *Computer , 31* (8), 35-43.

López-Sanz, M., & Marcos, E. (2012). ArchiMeDeS: A model-driven framework for the specification of service-oriented architectures. *Information Systems , 37* (3), 257-268.

López-Sanz, M., Acuña, C. J., Cuesta, C. E., & Marcos, E. (2007). UML profile for the platform independent modelling of service-oriented architectures. In S. B. Heidelberg (Ed.), *European Conference on Software Architecture* (pp. 304-307). Springer Berlin Heidelberg.

Lopez-Velasco, C. (2005). AWSDL: une extension de WSDL pour des services Web adaptés. *INFORSID*, (pp. 133-148).

Luoma, J., Kelly, S., & Tolvanen, J. P. (2004). Defining domain-specific modeling languages: Collected experiences. *4 th Workshop on Domain-Specific Modeling.*

Maddeh, M., Romdhani, M., & Ghedira, K. (2007). MOF-EMF Alignment. *Third International Conference on Autonomic and Autonomous Systems (ICAS'07).*

Madkour, M., El Ghanami, D., Maach, A., & Hasbi, A. (2013). Context-aware service adaptation: an approach based on fuzzy sets and service composition. *Journal of Information Science and Engineering , 29* (1), 1-16.

MataCase; Kelly, S. (2012). *Concrete Syntax Matters.*

Maximilien, E. M., Wilkinson, H., Desai, N., & Tai, S. (2007). A domain-specific language for web apis and services mashups. In S. B. Heidelberg (Ed.), *International Conference on Service-Oriented Computing* (pp. 13-26). Springer Berlin Heidelberg.

Mernik, M., Heering, J., & Sloane, A. M. When and how to develop domain-specific languages. *ACM computing surveys (CSUR)*, 2005, *37* (4), 316-344.

MetaCase. (2012). DOMAIN-SPECIFIC MODELING WITH METAEDIT+: 10 TIMES FASTER THAN UML. *WHITE PAPER* .

MODELWARE. (2006). *Industrial ROI, Assessment, and Feedback (Deliverable 5.3).*

Mohabbati, B., Asadi, M., Gašević, D., Hatala, M., & Müller, H. A. (2013). Combining service-orientation and software product line engineering: A systematic mapping study. *Information and Software Technology , 55* (11), 1845-1859.

Monfort, V., & Hammoudi, S. (2009). Towards adaptable SOA: model driven development, context and aspect. In S. B. Heidelberg (Ed.), *Service-Oriented Computing*, (pp. 175-189).

Mostefaoui, S. K., Gassert, H., & Hirsbrunner, B. (2004). Context meets web services: enhancing WSDL with context-aware features. *Proc. of 1st Intl. Workshop on Best Practices and Methodologies in Service-Oriented Architectures: Paving the Way to Web-services Success*, (pp. 1-14). Vancouver, British Columbia, Canada.

Narendra, N. C., & Ponnalagu, K. (2010). Towards a variability model for soa-based solutions. In IEEE (Ed.), *2010 IEEE International Conference on Services Computing (SCC)* (pp. 562-569). IEEE.

Nassar, M., Coulette, B., Crégut, X., Ebsersold, S., & Kriouile, A. (2003). Towards a View based Unified Modeling Language. *5th International Conference on Enterprise Information Systems (ICEIS"2003).* Angers, France.

NATO. (1968). NATO SOFTWARE ENGINEERING CONFERENCE. Garmisch, Germany: Peter Naur and Brian Randell.

Northrop, L. (2008). *Software Product Lines Essentials.* Retrieved 11 25, 2016, from http://www.sei.cmu.edu/library/assets/spl-essentials.pdf

OASIS. (2006). *Reference Model for Service Oriented Architecture 1.0.* Retrieved 11 25, 2016, from http://docs.oasis-open.org/soa-rm/v1.0/

OASIS, BPEL4WS. (2007). *OASIS Web Services Business Process Execution Language (WSBPEL) TC*. Retrieved 2016, from https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsbpel

OBEO, Acceleo. (2016). *Acceleo*. Retrieved 11 25, 2016, from http://www.eclipse.org/acceleo/

OBEO, Sirius. (2015). *Sirius*. Retrieved 2016, from http://www.eclipse.org/sirius/

Oberortner, D. I. (2011). *Monitoring Quality of Service in Service-oriented Systems: Architectural Design and Stakeholder Support.* Thèse de doctorat, Universitat Wien.

Oberortner, E., Zdun, U., & Dustdar, S. (2008). Domain-specific languages for service-oriented architectures: An explorative study. In S. B. Heidelberg (Ed.), *European Conference on a Service-Based Internet* (pp. 159-170). Springer Berlin Heidelberg.

Oberortner, E., Zdun, U., & Dustdar, S. (2009). Tailoring a model-driven quality-of-service DSL for various stakeholders. *ICSE Workshop on Modeling in Software Engineering* (pp. 20-25). IEEE.

OMG, MOF. (2016). Retrieved 2016, from http://www.omg.org/mof/.

OMG, MOF. (2015). *OMG Meta Object Facility (MOF) Core Specification.* Specification OMG, OMG.

OMG, MOF, M2T. (2016). *MOF Model To Text Transformation Language (MOFM2T), 1.0*. Retrieved 2016, from http://www.omg.org/spec/MOFM2T/1.0/

OMG, SoaML. (2009). *SoaML*. Retrieved 12 25, 2016, from http://www.omg.org/spec/SoaML/20091101/

OMG, SoaML. (2012). *SoaML*. Retrieved 11 25, 2016, from http://www.omg.org/spec/SoaML/1.0.1/, 2012

OMG, UML. (2016). *Unified Modeling Language*. Retrieved 11 25, 2016, from UML: http://www.uml.org/

OMG, XMI. (2015). *XML Metadata Interchange*. Retrieved 11 25, 2016, from http://www.omg.org/spec/XMI/

OMG; Soley, R. M. (2001). *Modeling All the Way Up…Modeling All the Way Down.* Retrieved 11 25, 2016, from www.omg.org/~soley/mdaupdown.ppt

Ortiz, G. (. (2012). *Adaptive Web Services for Modular and Reusable Software Development: Tactics and Solutions.* IGI Global.

Papazoglou, M. P., & Heuvel, W. (2006). Service-oriented design and development methodology. *International Journal of Web Engineering and Technology (IJWET) , 2* (4), 412-442.

Papazoglou, M. P., Traverso, P., Dustdar, S., & Leymann, F. (2008). Service-oriented computing: a research roadmap. *International Journal of Cooperative Information Systems , 17* (02), 223-255.

Parigot, D., & Boussemart, B. (2008). *Architecture Orientée Service Dynamique: D-SOA.* INRIA Sophia Antipols. INRIA.

Parra, C. A., Quinton, C., & Duchien, L. (2012). CAPucine: Context-aware service-oriented product line for mobile apps. *ERCIM News, 88, 38-39. , 88* (1).

Parra, C., Blanc, X., & Duchien, L. (2009). Context awareness for dynamic service-oriented product lines. In C. M. University (Ed.), *Proceedings of the 13th International Software Product Line Conference* (pp. 131-140). Carnegie Mellon University.

Pautasso, C., Zimmermann, O., & Leymann, F. (2008). Restful web services vs. big web services: making the right architectural decision. In ACM (Ed.), *Proceedings of the 17th international conference on World Wide Web* (pp. 805-814). ACM.

Pelechano, V., Albert, M., Muñoz, J., & & Cetina, C. (2006). Building Tools for Model Driven Development. Comparing Microsoft DSL Tools and Eclipse Modeling Plug-ins. *JISBD* .

Perez, B., & Correal, D. (2011). A model driven approach to the analysis of quality scenarios within self-adaptable SOA systems. *Electronic Notes in Theoretical Computer Science , 281,* 113-126.

Pitkänen, R., & Mikkonen, T. (2006). Lightweight Domain-Specific Modeling and Model-Driven Development. *OOPSLA 6th Workshop on Domain Specific Modeling,* (pp. 159-168).

Pooch, U. W. (1974). Translation of decision tables. *ACM Computing Surveys (CSUR) , 6* (2), 125-151.

Qiu, L., Shi, Z., Lin, F., & Shi, Z. (2007). Context Optimization of AI Planning for Semantic Web Services Composition. *Service Oriented Computing and Applications , 1* (2), 117-128.

Rahien, A. (2010). *DSLs in Boo: Domain Specific Languages in. Net.* Manning Publications Co.

Raik, H., Bucchiarone, A., Khurshid, N., Marconi, A., & Pistore, M. (2012). Astro-captevo: Dynamic context-aware adaptation for service-based systems. *IEEE Eighth World Congress on Services* (pp. 385-392). IEEE.

Rosenberg, F., Celikovic, P., Michlmayr, A., Leitner, P., & Dustdar, S. (2009a). An end-to-end approach for QoS-aware service composition. In IEEE (Ed.), *IEEE International Enterprise Distributed Object Computing Conference, 2009. EDOC'09* (pp. 151-160). IEEE.

Rosenberg, F., Leitner, P., Michlmayr, A., Celikovic, P., & Dustdar, S. (2009b). Towards composition as a service-a quality of service driven approach. In IEEE (Ed.), *2009 IEEE 25th International Conference on Data Engineering* (pp. 1733-1740). IEEE.

Rubel, D., Wren, J., & Clayberg, E. (2011). *The Eclipse Graphical Editing Framework (GEF).* Addison-Wesley Professional.

Safa, L. (2006). The practice of deploying DSM, report from a Japanese appliance maker trenches. *Proceedings of the 6th OOPSLA Workshop on Domain Specific Modeling (DSM'06).*

Saleem, M., Jaafar, J., & Hassan, M. (2012). A domain-specific language for modelling security objectives in a business process models of soa applications. *AISS , 4* (1), 353-362.

Sánchez-Ruíz, A. J., Saeki, M., Langlois, B., & Paiano, R. (2007). Domain-specific software development terminology: Do we all speak the same language? *Proceedings of the seventh OOPSLA Workshop on Domain-Specific Modeling.*

Schilit, B., Adams, N., & Want, R. (1994). Context-aware computing applications. In IEEE (Ed.), *First Workshop on Mobile Computing Systems and Applications, 1994. WMCSA* (pp. 85-90). IEEE.

Schmidt, A., Beigl, M., & Gellersen, H. W. (1999). There is more to context than location. *Computers & Graphics , 23* (6), 893-901.

Sheng, Q. Z., & Benatallah, B. (2005). ContextUML: a UML-based modeling language for model-driven development of context-aware web services. In IEEE (Ed.), *International Conference on Mobile Business (ICMB'05)* (pp. 206-212). IEEE.

Sinnema, M., Deelstra, S., & Hoekstra, P. (2006). The COVAMOF derivation process. *Proceedings of the International Conference on Software Reuse (ICSR),* (pp. 101–114).

Steinberg, D., & Committer, E. M. (2008b). Fundamentals of the eclipse modeling framework. *Eclipse Foundation, editor, eclipsecon.*

Steinberg, D., Budinsky, F., Merks, E., & Paternostro, M. (2008a). *EMF: eclipse modeling framework.* Pearson Education.

Stojanovic, Z. (2005). *A Method for Component-Based and Service-Oriented Software Systems Engineering.* Thèse de doctorat, TU Delft, Delft University of Technology.

Strang, T., & Linnhoff-Popien, C. (2004). A context modeling survey. *Workshop Proceedings.*

Strang, T., Linnhoff-Popien, C., & Frank, K. (2003). CoOL: A context ontology language to enable contextual interoperability. In S. B. Heidelberg (Ed.), *International Conference on Distributed Applications and Interoperable Systems*, (pp. 236-247).

Sun, C. A., Rossing, R., Sinnema, M., Bulanov, P., & Aiello, M. (2010b). Modeling and managing the variability of Web service-based systems. *Journal of Systems and Software , 83* (3), 502-516.

Sun, C. A., Xue, T., & Aiello, M. (2010a). ValySeC: A variability analysis tool for service compositions using VxBPEL. In IEEE (Ed.), *2010 IEEE Asia-Pacific Services Computing Conference (APSCC)* (pp. 307-314). IEEE.

Suraci, V., Mignanti, S., & Aiuto, A. (2007). Context-aware Semantic Service Discovery. *16th IST Mobile and Wireless Communications Summit, Budapest, Hungary.*

Tao, A. T., & Yang, J. (2007). Supporting Differentiated Services With Configurable Business Processes. In IEEE (Ed.), *IEEE International Conference on Web Services (ICWS 2007)* (pp. 1088-1095). IEEE.

Vale, S., & Hammoudi, S. (2009). An Architecture for the Development of Context-aware Services based on MDA and Ontologies. *Proceedings of the International MultiConference of Engineers and Computer Scientists, 1*.

Van Deursen, A., Klint, P., & Visser, J. (2000). Domain-Specific Languages: An Annotated Bibliography. *Sigplan Notices , 35* (6), 26-36.

Vanthienen, J., & Dries, E. (1992). *Developments in decision tables: Evolution, applications and a proposed standard.* ONDERZOEKS RAPPORT NR 9227.

Wada, H., & Suzuki, J. (2005c). Modeling turnpike frontend system: A model-driven development framework leveraging UML metamodeling and attribute-oriented programming. In S. B. Heidelberg (Ed.), *International Conference on Model Driven Engineering Languages and Systems,* (pp. 584-600).

Wada, H., Suzuki, J., & Oba, K. (2005a). Modeling turnpike: a model-driven framework for domain-specific software development. *In Companion to the 20th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications* (pp. 128-129). ACM.

Wada, H., Takada, S., Suzuki, J., & Doi, N. (2005b). Leveraging metamodeling and attribute-oriented programming to build a model-driven framework for domain specific languages. *8th JSSST Conference on Systems Programming and its Applications.*

Weiss, D., & Lai, C. (1999). *Software Product-line Engineering.* Addison Wesley, Longman.

Winograd, T. (2001). Architectures for Context. *Human Computer Interaction Journal* .

Xiaohang, W. (2003). *The context gateway : a parvesive computing infrastructures for context aware services.* In Technical report, National university of singapore.

XMF-Mosaic. (2011). Retrieved 2016, from http://www.eis.mdx.ac.uk/staffpages/tonyclark/Software/xmf.html

Yahyaoui, H., Wang, L., Mourad, A., Almullah, M., & Sheng, Q. Z. (2011). Towards context-adaptable Web service policies. *Procedia Computer Science , 5*, 610-617.

Zimmermann, O., Krogdahl, P., & Gee, C. (2004). *Elements of Service-Oriented Analysis and Design.* Retrieved 11 25, 2016, from http://www.ibm.com/developerworks/library/ws-soad1/

Annexe A

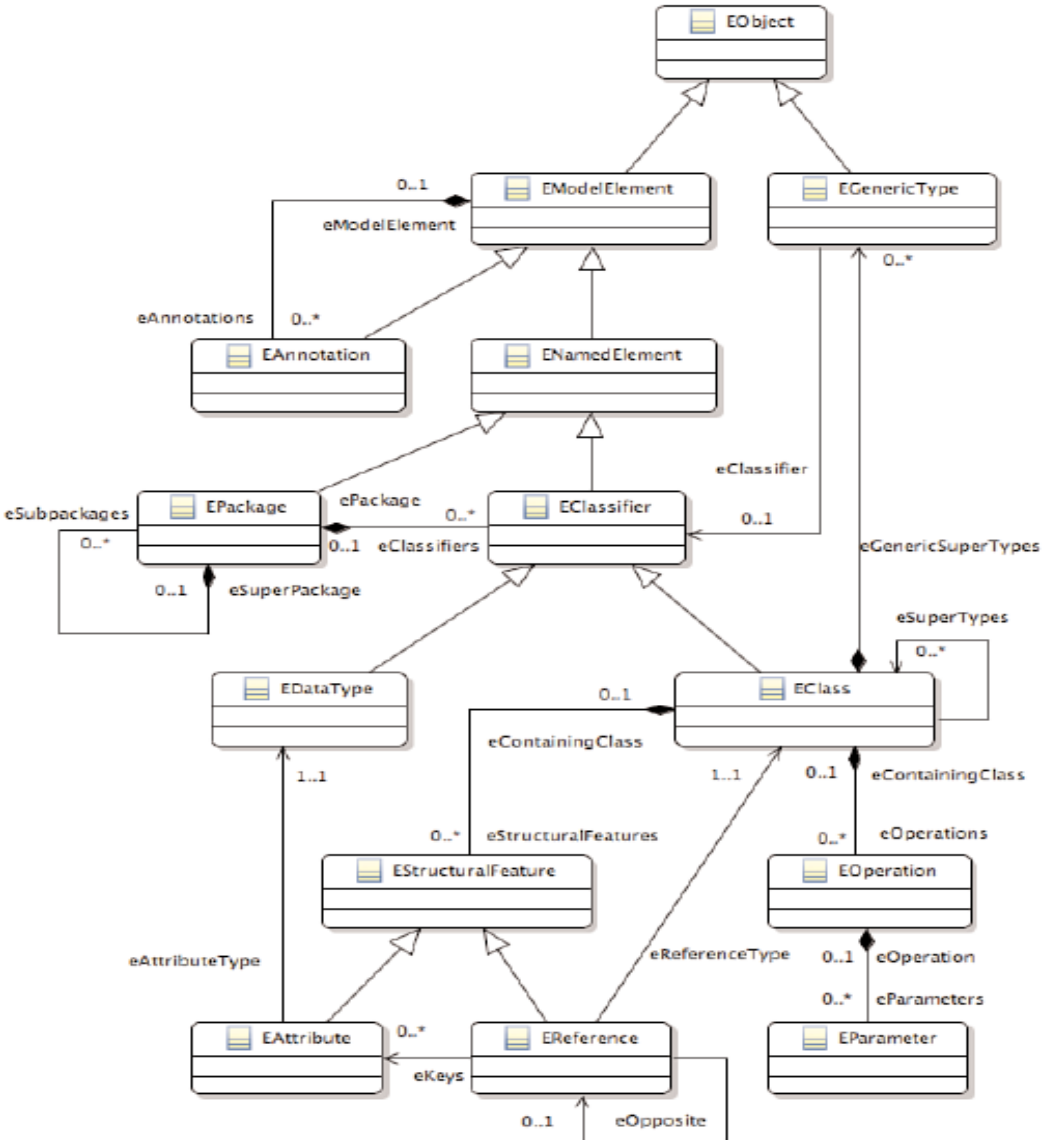


Figure 87 : Modèle Ecore de base (Gronback, 2009)

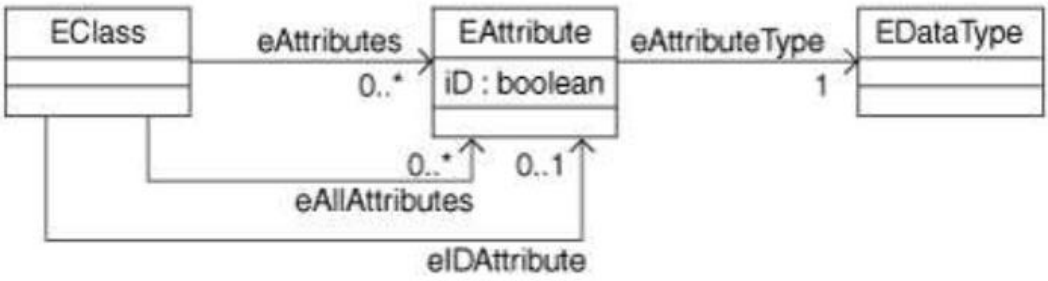


Figure 88 : Les attributs Ecore (Steinberg, et al., 2008a)

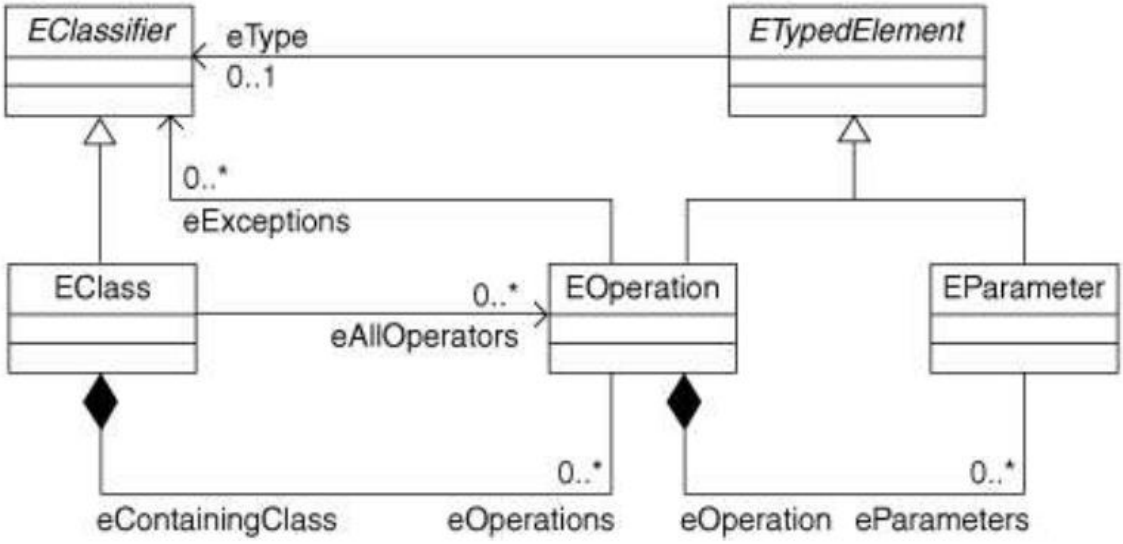


Figure 89 : Les opérations et les paramètres Ecore (Steinberg, et al., 2008a)

Tableau 16 : Correspondance entre MOF et EMF (Ecore) (Maddeh, et al., 2007)

MOF 2.0 concept	EMF concept	Comments
Package	EPackage	A grouping mechanism in both universes. The attributes, EMF::nsURI and EMF::nsPrefix does not have any equivalence in MOF metamodel.
Class	EClass	Meets the same requirement, defining "things" in both universes. The attribute "isSingleton" for which there is no corresponding EMF concept is not supported in MOF 2.0, It can be simulated by inserting a Constraint on the class: self.metaobject.allInstances.size =1.
Enumeration	EEnum, EEnumLiteral	Represents the same concept.
Parameter	EParameter	Represents the same concept.
Operation	EOperation	Meets the same requirement.
Attribute where the type is a DataType	EAttribute	Meets the same requirement.
Attribute where the type is a Class	EReference	Meets the same requirement.

Reference	EReference	But it can be omitted since in MOF 2.0 the reference is redundant as a separate element from the referenced AssociationEnd (which is now a Property). The fact that the AssociationEnd had a Reference means that the new Property corresponding to the AssociationEnd should be owned by the Class not the Association. This information can be lost without any repercussion on the MOF model.
Association with Association End as attribute	EReference	A 2-way-navigable Association in MOF is mapped to a pair of "opposite" EReferences. A 1- way-navigation Association in MOF in EMF, AssociationEnd became a property associated with an Association via memberEnd attribute. To make the Property (i.e., the AssociationEnd) exposed in a class (similarly to using a reference in MOF 1.4), it needs to be owned by the class (while still being a memberEnd of the association). The mappings of same relevant attributes as follows: AssociationEnd::aggregationin MOF 1.4 is changed in MOF 2.0 to Property::isComposite (composite maps to true, none maps to false, shared was underspecified in MOF 1.4 –maps to false).
		AssociationEnd::aggregation, Not supported MOF 2.0 defines navigability in terms of being able to navigate to the end directly from a type – this is analogous to having a reference in MOF 1.4. AssociationEnd::isChangeable, is mapped as explained above for StructuralFeature
Constant	EDataType	The EDatatype nominates a Java class with only a single attribute "constant" which is of the appropriate type and value and is final.
Constraint	Not supported	There is no equivalent concept in EMF.
Extension : Tag	EAnnotation	Last EMF version does not have an equivalent concept to MOF::Tag, which make difficult saving the information that are not support by EMF metamodel, but the new concept EAnnotation solve this problem (see section 5).
Not supported	EFactory	The package factory in MOF does not modelled but is generated for the implementation.

Annexe B

Etapes de génération d'un modèle Ecore en utilisant le générateur de modèles EMF (EMF Generator Model) : exemple langage de calcul d'impôts

Après la création d'un projet GMF (CADSSOTaxCalculation) et d'un package (cadsso.taxCalculation) contenant l'ensemble d'interfaces Java annotées, il faut suivre les étapes suivantes :

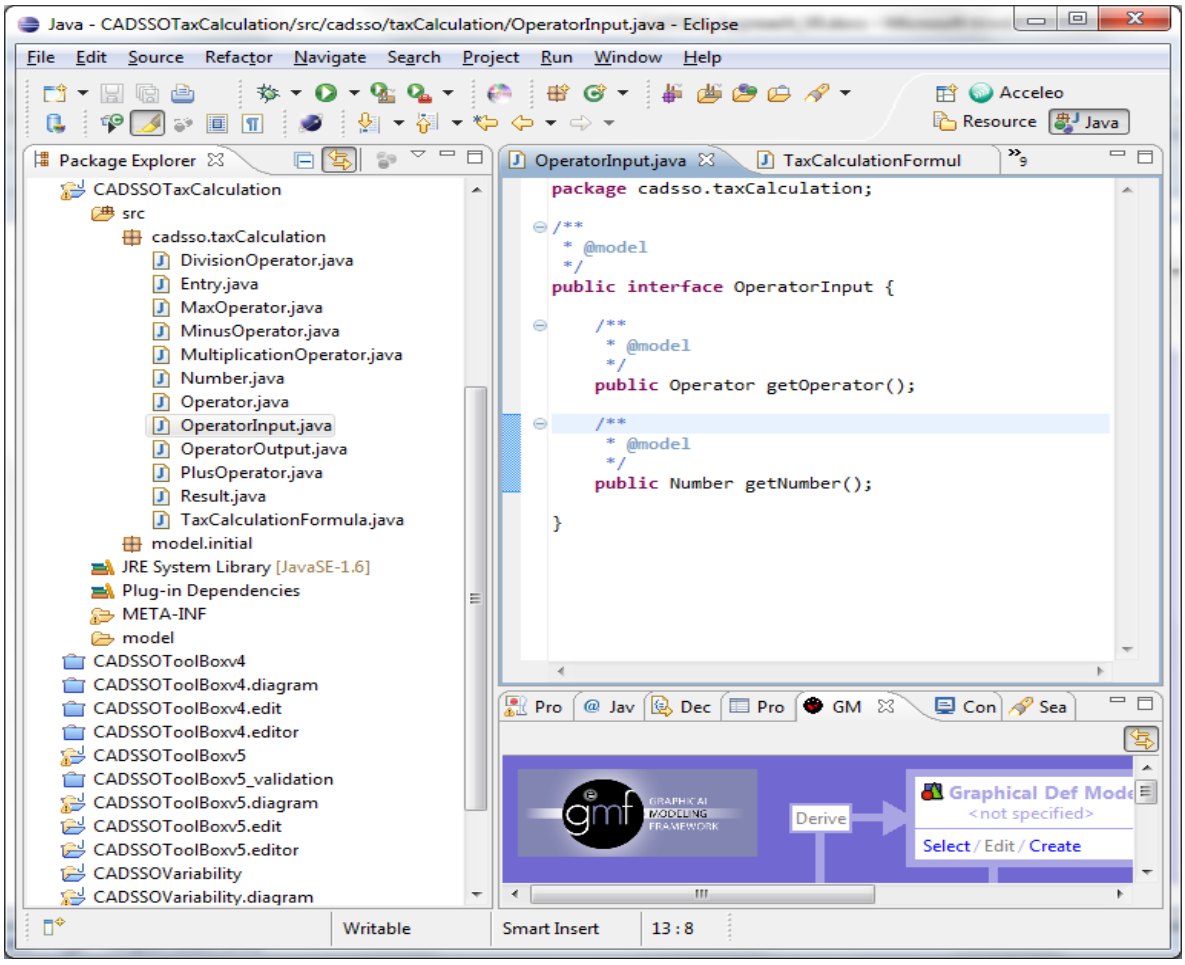


Figure 90 : Génération d'un modèle Ecore : Modèle java de base

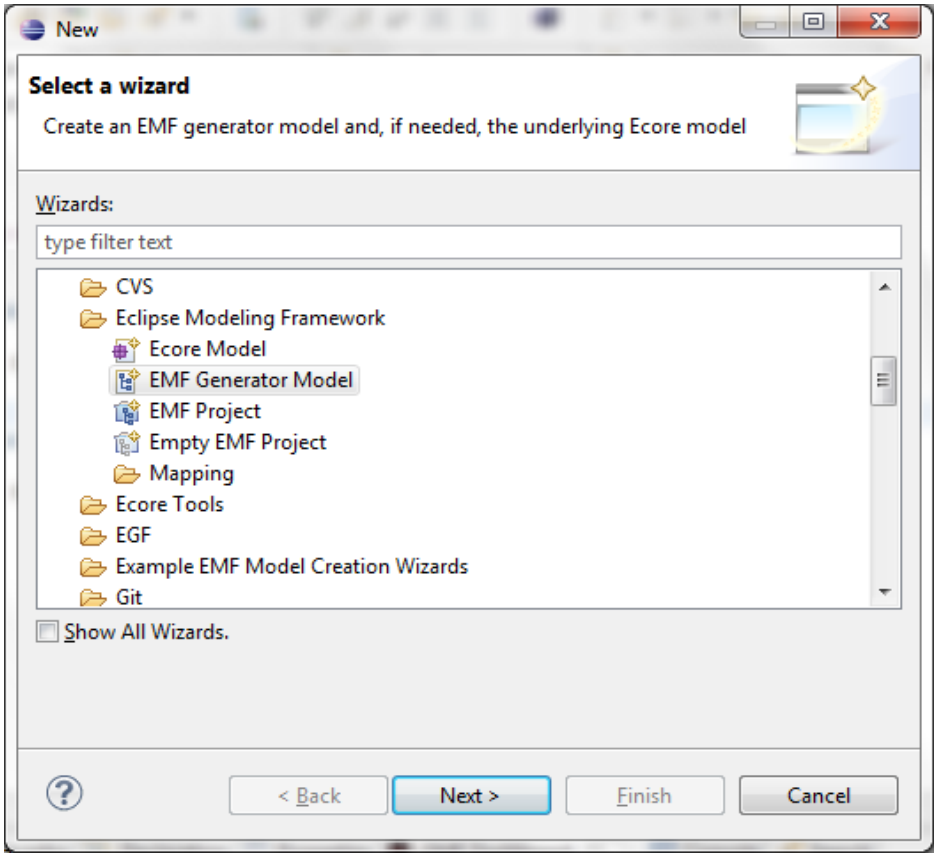


Figure 91 : Génération d'un modèle Ecore : Choix du générateur de modèles EMF

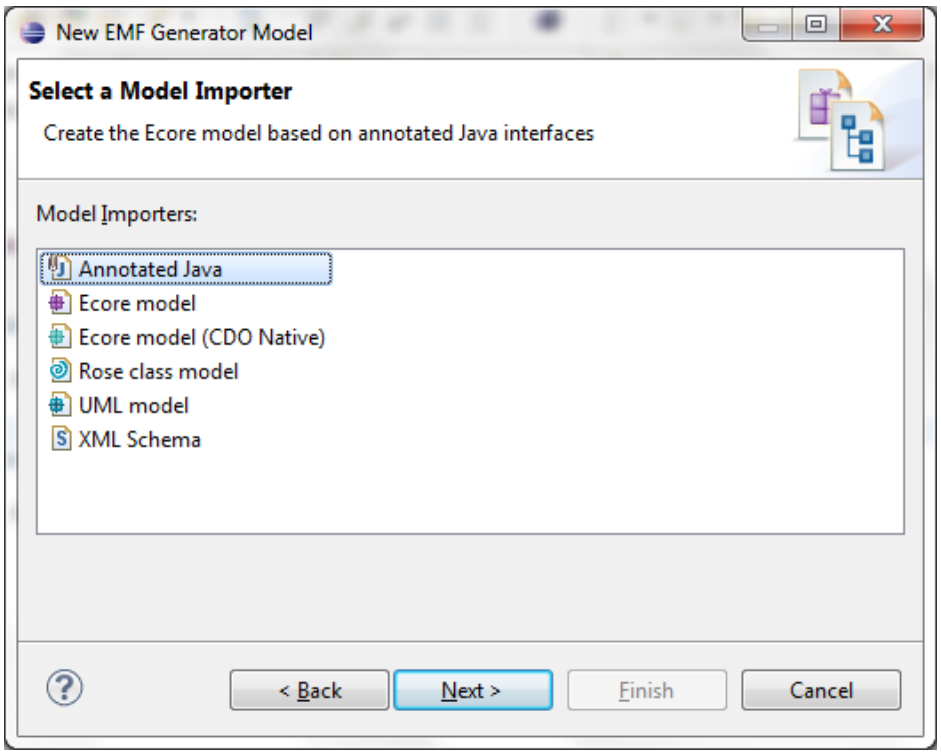


Figure 92 : Génération d'un modèle Ecore : Choix de l'utilisation d'interfaces Java annotées

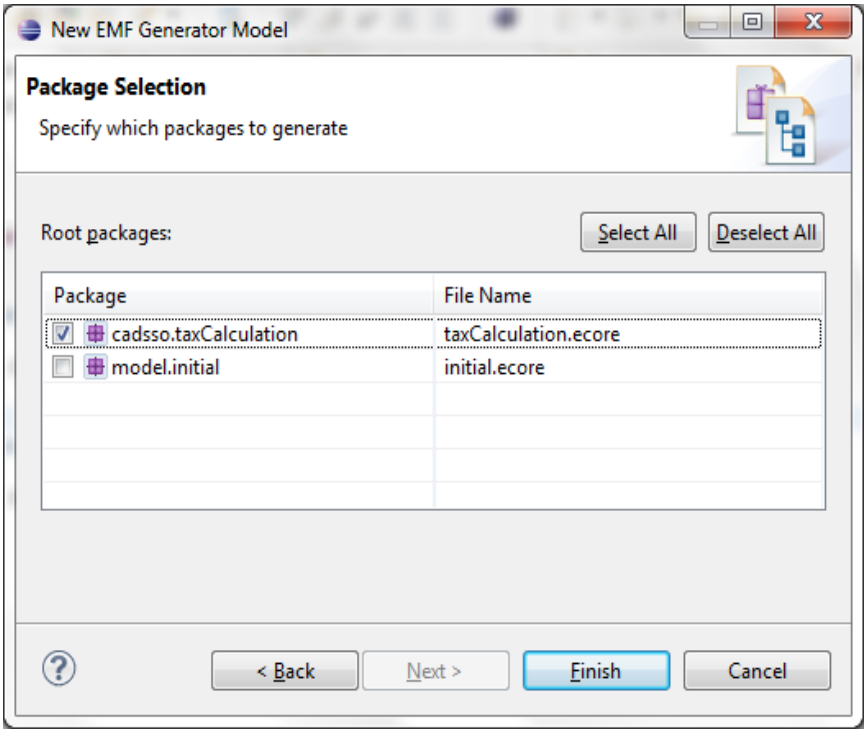


Figure 93 : Génération d'un modèle Ecore : Choix du package

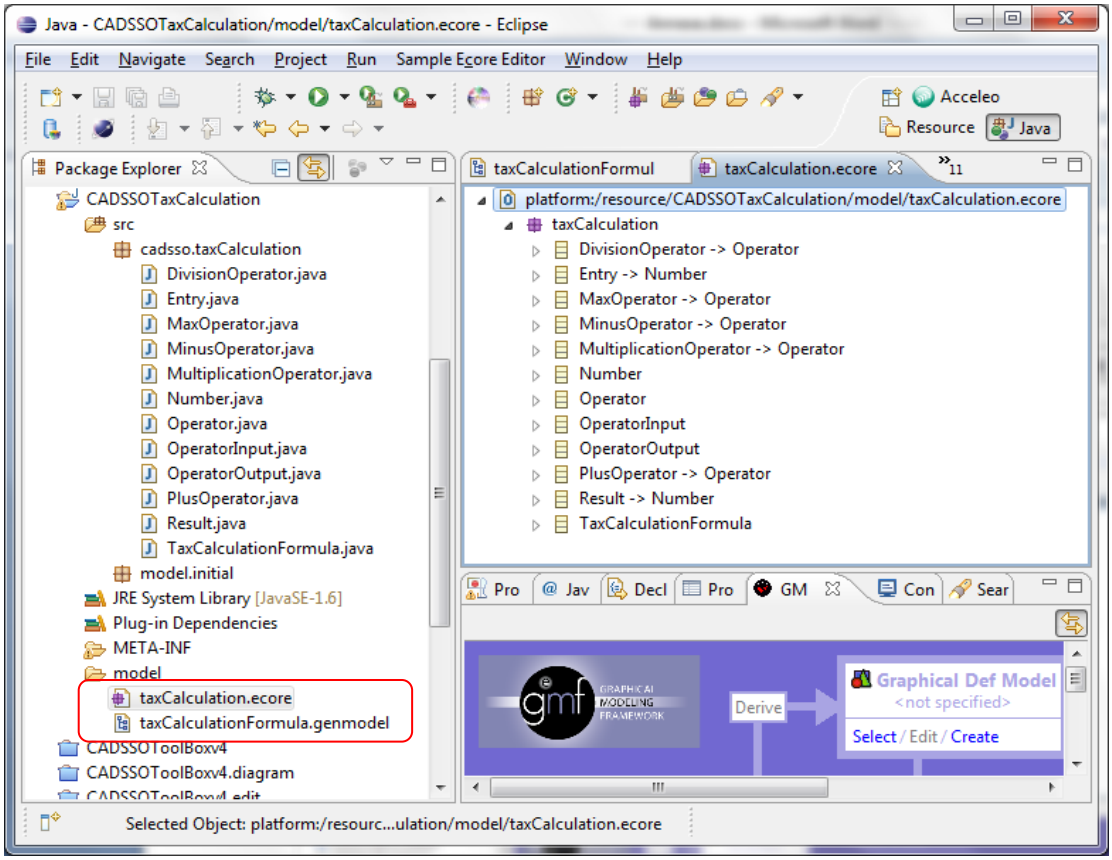


Figure 94 : Génération d'un modèle Ecore : Génération des fichiers *.ecore* et *.genmodel*

Annexe C

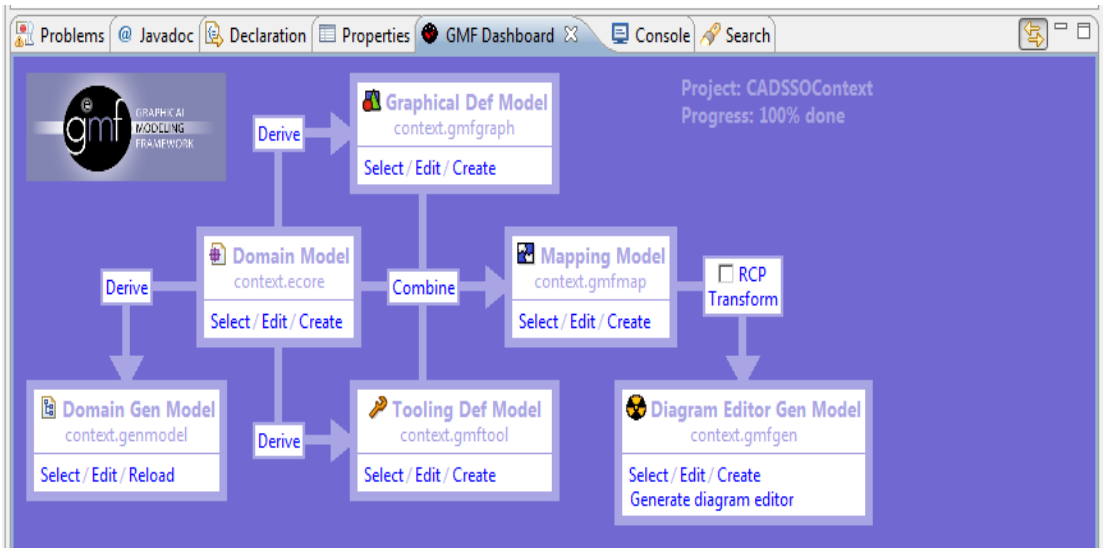


Figure 95 : Tableau de bord GMF des étapes de production de l'outil de modélisation du contexte

```
[template public generateInterfaces(anAdaptableSOA : AdaptableSOA)]
[comment service interfaces generation/]
[for (s: Service | anAdaptableSOA.services) ]
        [for (si: ServiceInterface | s.serviceInterfaces) ]
                [file (si.name.toUpperFirst().concat('.java'), false)]
package [s._package.name/];     …
public interface [si.name.toUpperFirst()/] {
   [for (svp: ServiceVariationPoint | s.serviceVariationPoints) ]
      [for (si: ServiceInterface | s.serviceInterfaces) ]
        [for (so: ServiceOperation | si.serviceOperations) ]
       @WebMethod
       [so.outServiceParameter -> at(1).type/] [so.name.concat(svp.name) /](
       [for (insp: InServiceParameter | so.inServiceParameter) separator (', ') ] [insp.type/] [insp.name/] [/for]    );
        [/for]
       [/for]
    [/for]    …
```

Figure 96 : Extrait du template Acceleo "generateInterfaces"

```
package ctpackage;
import javax.jws.WebMethod;
import javax.jws.WebService;
import javax.jws.soap.SOAPBinding;
import javax.jws.soap.SOAPBinding.Style;
@WebService
@SOAPBinding(style = Style.DOCUMENT)
public interface CTCalculationServiceInterface1 {
        @WebMethod
        double cTaxCalculationAutomaticTaxation( TaxDeclaration cTaxDeclaration1 ,  RatePayer ratePayer1);
        @WebMethod
        double cTaxCalculationNotResident( TaxDeclaration cTaxDeclaration1 ,  RatePayer ratePayer1);
```

```
    @WebMethod
        double cTaxCalculationWhithoutExemption(TaxDeclaration cTaxDeclaration1 ,  RatePayer ratePayer1);
    @WebMethod
    double cTaxCalculationWithExemption(TaxDeclaration cTaxDeclaration1 ,  RatePayer ratePayer1);
}
```

Figure 97 : Extrait du résultat de la génération de code source du template Acceleo « generateInterfaces »

Annexe D

Tableau 17 : Questionnaire facilitant la prise de décision concernant l'adoption de l'approche DSM pour un domaine donné (Kelly, et al., 2008a)

How mature is the target business area in your company?		
Sub question	**Score**	**Description**
An established business	+3	Clear business concepts, needs, Stakeholders
Turning an existing customizable product into a platform for use by others	+1	Existing demand for product and its customisation; DSM makes this fast and palatable for those who are not in-house experts
Development of a new product	0	Risk of the unknown balanced by agility stemming from lack of legacy code and processes
Creating a new platform for use by others	-1	Desire to offer familiarity and lowest common denominator works against DSM. Uncertain commitment, risk of project failure
A research project	-3	High risk of failure from other factors, and no commitment to project if it hits difficulties
How much are development and core business processes in the target area coupled to external organizations?		
Joint venture	-3	Serious concerns over equality of commitment and ownership, ability to agree
Internal except for third-party components and tools	+1	Clear control over destiny, components give stability
Totally internal, build all own components and tools	-1	Possibly indicative of fear of commitment, can succeed only if DSM is truly seen as "theirs"
How closely is this domain related to other candidate domains?		
Simplest or smallest of a group of several similar domains	+2	Good possibilities for extension to other domains later, or switching if this turns out to be a bad choice
One of several similar domains	+1	Indicative of an important business area
Different domains work independently of each other	-1	Sometimes indicates lack of desire to codify and build on what is known
How much do you customize or configure your software for each customer?		
Extensively	+3	DSM will allow you to capture just the variable aspects, and code generation will improve delivery times and the quality of the finished product

Only for large customers	+2	DSM can integrate many of these bespoke cases back into the mainstream software artifacts, benefiting both large and small customers
Often but only in limited ways	+1	Possible small language to describe configuration
Little or not at all	0	Useful only if DSM will be used for the main application
Do you have good source code examples available, for example for teaching new staff?		
Yes	+3	Offers a basis for the generator and is indicative of a mature domain where development is managed well
No, but we are always saying we should make some	0	Basis exists, DSM can help bring about wanted change, but process will be harder
Some available, but out of date and from different times	-1	Often indicative of a lack of stability and of reworking upon reworking, and hack upon hack. DSM may be an effective way out if there is a strong expert developer
No, our staff are all expert Developers	-3	if said by manager
	+1	if said by developer
Do you have an in-house application framework?		
Yes, with established guidelines, best practices and sample code	+3	Clear existing framework, expected code generator output, domain concepts
Yes, the previous version is the Guideline	+2	Domain concepts clear, framework quite easy to build from existing code
No, but we try to reuse legacy components	-2	Would have to define the framework, domain concepts probably also unclear
No, we develop everything from scratch	Abort	"First of a kind" development, not worth targeting
How would you describe the maturity of your software development process?		
Precisely defined and developers must follow it	0	May be too caught up in CMM and processes to accept DSM or its subsequent evolution
The majority is documented and we generally follow it	+2	Shows a desire to take the effort to codify what is known and follow it as long as it is useful: fertile ground for DSM
We have a good idea where we are at any given time	+1	Developer-led
No clear process	-2	Hard to bring about organizational change without organization

Can you assign the following kinds of people to the DSM project?		
One of the top three who built the framework/first product	+5	
An experienced developer	+3	Can build code generator
A small team of normal Developers	+2	Vital for piloting, can build example apps to be the input for creating generators
No, but you can have a summer intern or two	-3	May have brains and enthusiasm, but lack domain knowledge and software development experience in large teams and long projects
No	Abort	DSM cannot be successful without the developers' experience

Proof of Concept Workshop

The aim of this document is to collect the key elements of your domain for the purposes of the domain-specific modeling language implementation workshop.

The goal of the workshop is to define and implement a small, but still significant, part of your domain-specific modeling language to demonstrate the benefits with concrete examples for executives and product developers. The result of the workshop is a partial modeling and code generation environment for your organization's own domain-specific modeling language.

As domains differ widely, not all questions may be strictly applicable in your domain: feel free to change the questions or omit irrelevant parts.

1. Introduction

Please give a short introduction to your domain area. What parts of the work in the domain do you want to include in your modeling language, and why are the benefits of domain specific modeling important in these parts?

2. Usage

Describe briefly how you intend to use the modeling language: what is the input on which the models are based, who makes the models, what is generated, and what other tools are involved. Where possible, give approximate indications of how many users, models, files, and so on there are, and how much various parts are reused between products, features, or models.

3. Sample material

Pick a small but representative existing example whose implementation would take about one week for one developer. A good example is often the kind of feature you would give as a first task for a new programmer. The example should include the major, most central elements of the domain.

Describe the example briefly here, along with the approximate time to implement it, any notes about special features of this example, things that have changed since it was made, and how it fits in with other related applications. Then continue to fill in Sections 3.1–3.4 with more specific information (attach separate documents where necessary, making links to them from this document).

3.1 Sample requirements

Give the requirements for the example functionality, including how it interfaces with other parts of the system.

3.2 Sample design

Give the design documents, including graphical models and text where available. As far as possible, the design should be in step with the code below.

3.3 Sample code

Give the code for this example. If there are several files then also provide an overview of what each file does. Where possible, comment the code with references to the design documents. These comments will help in analyzing which parts of models could produce which parts of code.

3.4 Sample user's manual

If the code will have a human user present when run, either using or monitoring the application, give user instructions for that person using this code.

4. Rough modeling language sketches

Draw the diagrams below in whatever is the easiest format: hand drawn is fine.

4.1 Product contents

Often a product (or range of products) consists of several parts (features, modules, etc.). Sketch a (partial) typical product along with its parts and subparts. Mark which parts stay largely the same between products (e.g., existing code libraries, components) and which parts are to be modeled in the other diagrams below.

4.2 Product behavior

Sketch a diagram of a typical feature (possibly the sample feature above) as might be drawn by designers on a whiteboard at an initial design session. Try to avoid "standard" design languages such as UML, which concentrate on the resulting code components, and use informal concepts commonly used and understood by designers. Sometimes there may need to be more than one diagram at this level, for example, one for the user interface and one for the behavior.

Figure 98 : Template d'un atelier DSM (Kelly, et al., 2008a)