Génération automatique d'alternatives de structuration de données pour systèmes orientés document

Paola Gómez [1], **Rubby Casallas** [2], **Claudia Roncancio** [1]

1. *Univ. Grenoble Alpes, CNRS, Grenoble INP* [*], *LIG, 38000 Grenoble, France*
   {paola.gomez-barreto,claudia.roncancio}@univ-grenoble-alpes.fr
2. *TICSw, Universidad de los Andes, Bogotá - Colombia*
   rcasalla@uniandes.edu.co

RÉSUMÉ. *Les possibilités de structuration des données dans les systèmes orientés document sont très nombreuses même lorsque les données à traiter sont simples. Le choix de la structuration est néanmoins très important car il conditionne des aspects cruciaux de la base et des applications. Nos travaux visent à aider l'utilisateur à comprendre, évaluer et choisir, la structuration des données d'une base orientée document selon les besoins. Dans cet article, nous proposons de générer automatiquement des alternatives de structuration orientées document, à partir d'un schéma UML, afin de pouvoir les comparer. Nous adoptons une approche originale de génération inspirée des lignes de produits logiciels. Dans notre cas, le produit est une structure de données orientée document basée sur JSON. Nous utilisons des modèles de caractéristiques pour représenter les variantes et les points communs entre les produits tout en contrôlant l'explosion du nombre de combinaisons. Cet article présente l'approche, les algorithmes de génération et le prototype qui permet aux utilisateurs d'apprécier diverses alternatives de structuration pour une base de documents.*

ABSTRACT. *The possibilities of data structuring in document-oriented systems are numerous even for simple data. The choice of the structuring is nevertheless very important because of its impact on crucial aspects of the base and the applications. Our work aims to help users to understand, evaluate and choose the data structuring of a document-oriented database. In this paper, we propose to automatically generate document-oriented structuring alternatives from a UML model to compare them. We adopt an original approach of generation inspired by Software Product Lines. In our case, the product is a document-oriented structure based on JSON. We use feature models to represent variations and common points between products by controlling the explosion of combinations. This paper presents the approach, the generation algorithms and the prototype that allows users to appreciate data structuring alternatives.*

MOTS-CLÉS : *NoSQL, systèmes orientés document, variabilité, modèle des caractéristiques*

KEYWORDS: *NoSQL, document-oriented systems, variability, feature models*

*. Institute of Engineering Univ. Grenoble Alpes

1. Introduction

La flexibilité des modèles semi-structurés supportés par les systèmes NoSQL orientés documents ouvre la porte à de nombreuses possibilités de représentation des données. Ces possibilités peuvent être nombreuses et des alternatives de structuration potentiellement intéressantes peuvent être écartées car, par exemple, le développeur n'a pas les moyens d'analyser toutes les options, ou parce que, involontairement, un facteur clé n'est pas pris en compte. Ces alternatives sont difficiles à envisager, à comprendre et à gérer à cause du nombre et de l'absence de schéma de base de données dans des systèmes tels que MongoDB. Néanmoins le choix de la structuration de données a un impact important sur la base et les applications. Cela concerne, entre autres, les performances, la facilité de programmation et la facilité à comprendre, maintenir et faire évoluer le système (Gómez *et al.*, 2016).

Nos propositions visent à aider le développeur à mener une phase de conception et d'analyse malgré les nombreuses alternatives possibles et l'utilisation d'un système sans schéma (*Schemaless*). Ainsi, nous explicitons le type des structures de données utilisées dans la base. Pour cela nous proposons le format AJSchéma, simple et lisible, qui sans jouer le rôle de schéma dans la base, permettra de mettre en évidence les types des données et ainsi de raisonner sur la structure.

Nous avons proposé l'approche SCORUS (Gomez, 2018) où le développeur modélise ses données à l'aide d'UML. Ce modèle est traité afin de générer un ensemble d'*alternatives de structuration orientées documents* sous forme d'AJSchémas. Ensuite, une phase d'évaluation, SCORUS évalue automatiquement des métriques pour chaque AJSchéma. Ces métriques constituent des indicateurs objectifs des caractéristiques des structures. Une phase d'analyse complète l'approche en s'appuyant sur les métriques et les préférences des utilisateurs tel que présenté dans (Gómez *et al.*, 2018b ; 2018a). Le prototype ScorusTool implémente cette approche.

Cet article porte sur la phase de génération de variantes de structuration orientées document. Cette génération automatique permet à l'utilisateur d'apprécier facilement de multiples choix de structures dont l'analyse peut être poursuivie ou non. Pour ce générateur nous avons suivi une approche issue du domaine des lignes de produits logiciels. Il s'agit de l'approche par modèles de caractéristiques qui permet d'exprimer la variabilité entre diverses alternatives d'un produit (Kang *et al.*, 2002). Ici, nous produisons diverses alternatives d'AJSchéma. Nous utilisons le modèle UML et les possibilités de structuration orientées documents afin d'identifier les alternatives possibles. Cela nécessite de formaliser les variations et points communs à travers d'un modèle de caractéristiques. La stratégie des modèles de caractéristiques permet de définir la variabilité des structures possibles d'une manière compacte et de contrôler l'explosion des possibilités qui peuvent survenir. Cet article présente l'approche et l'algorithme *AMISS* qui crée un modèle de caractéristiques contenant les variations des structurations. Pour chaque alternative de structuration fournie par ce modèle, nous dérivons les structures correspondantes dans divers formats dont les AJSchémas.

Dans la suite, la section 2 introduit un exemple illustrant la flexibilité et la variabilité des structures. La section 3 présente les principes des lignes de produits et des modèles de caractéristiques. La section 4 détaille les étapes de la ligne de produits basée

sur les modèles de caractéristiques qui permettent d'obtenir a partir d'un modèle UML des alternatives de structuration orientée document. Dans la section 5 nous introduisons l'algorithme *AMISS*. La section 6 introduit le prototype *ScorusTool*. Les travaux connexes sont décrits en section 7. Nos conclusions et perspectives de recherche sont présentées en section 8.

2. Flexibilité et évolution dans la modélisation semi-structurée

Nous sommes intéressés par la flexibilité d'une modélisation des données dans une base orientée document semi-structurée et ayant un grand nombre d'alternatives de structuration possibles. Dans cette section, nous illustrons ces aspects à travers un exemple, dans un contexte de marketing et en considérant un modèle conceptuel UML.

Dans les bases orientées document, les données sont gérées comme un ensemble de collections de documents. Un document est simplement un ensemble de paires `attribut:valeur`. Le type des valeurs peut être atomique ou complexe. Par complexe nous entendons soit un tableau de valeurs de tout type ou un document qui dans ce cas est dit *imbriqué*. Notons que la valeur d'un attribut peut être l'identifiant d'un document ou la valeur d'un attribut d'un document d'une autre collection. Cela permet de *référencer* un ou plusieurs documents. Ce système de types permet beaucoup de flexibilité dans la création des structures.

Nous considérons l'information des agences et des secteurs d'activités introduite par la Figure 1. La classe `Agency` représente une agence qui peut gérer plusieurs secteurs d'activités, représentés par la classe `Business` avec le rôle `bLines`. À son tour, un secteur d'activités est géré par une seule agence (rôle `ag` de l'association).

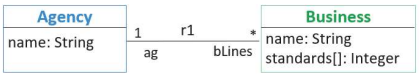

Figure 1. Modèle de classes pour des agences et des secteurs d'activités

Il y a plusieurs façons de modéliser ces données dans une base orientée documents. Chacune des classes peut inspirer la création d'une collection dont les documents contiennent l'ensemble des attributs de sa classe. Dans notre exemple, une collection `Agencies` (respectivement `Business`) correspondra à une collection d'agences dont les documents contiendront les attributs de la classe `Agency` (resp. `Business`).

Concernant l'association, elle peut se matérialiser soit par imbrication soit par référencement des informations de ses classes extrémités en considérant les deux sens de l'association. Cela introduit plusieurs façons de structurer les collections. Il s'agit alors d'avoir un ensemble d'options de manière à disposer de choix intéressants pour le concepteur tout en contrôlant le nombre de possibilités offertes. Nous définissons des **contraintes de structuration** qui portent sur la complétude et certaines formes de liens entre les données qui évitent l'isolement et une complexité inutile qui peut s'avérer coûteuse.

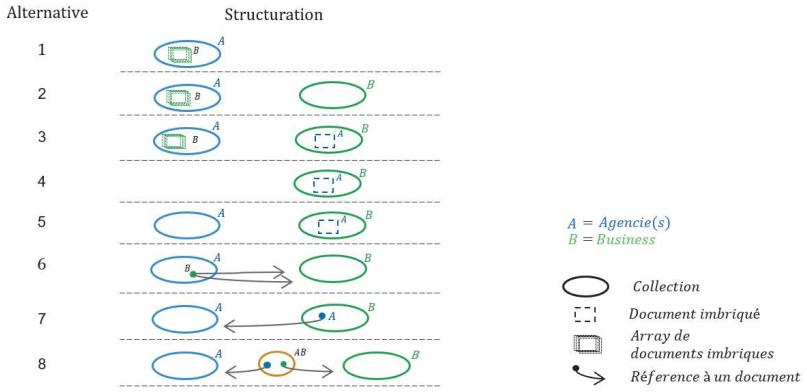

Figure 2. Alternatives de semi-structuration pour le modèle UML de la Figure 1

Existence : pour garantir l'existence de l'information, les alternatives de structuration devront inclure une collection si ses documents sont référencés.

Isolement : empêcher l'isolement potentiel

– d'une collection A qui n'imbrique pas l'information de sa collection partenaire B. A doit être référencée, ou imbriquée dans une autre collection.

– de deux collections. Considérer une *collection-lien* ne contenant que des références aux deux collections si aucune d'elles n'imbrique ni ne référence les informations de sa partenaire.

Références circulaires: pour limiter une forte complexité induite par des références circulaires, nous considérons les alternatives suivantes,

– empêcher les collections d'être référencées entre elles et

– si une collection A imbrique l'information d'une collection B alors, empêcher que cette information inclut une référence vers A.

Reprenons l'exemple de la Figure 1 au vu des contraintes de structuration énoncées. La représentation de l'association r_1 peut être faite selon les huit alternatives illustrées sur la Figure 2. Les documents d'une collection sont homogènes dans leur structure.

Les alternatives 6 et 7 respectent l'existence de la collection référencée. Les alternatives 2 et 5 empêchent l'isolement d'une collection en imbriquant et en dupliquant leur information. L'alternative 8 utilise une *collection-lien* pour les références. Les alternatives qui ne respectent pas les *contraintes de structuration* sont écartées, par exemple l'alternative avec les collections `Agencies` et `Businness` sans imbrication ni référencement entre elles.

Dans le cas où les besoins changent et de nouvelles classes sont ajoutées au modèle UML, la structure de stockage doit changer. De nouvelles alternatives de structuration sont possibles et cela peut impliquer la mise à jour de celles déjà existantes. Le nombre d'alternatives de structuration possibles peut être très grand et le choix difficile. Des contraintes sont nécessaires pour identifier des alternatives de structuration valides tout en limitant la combinatoire.

Notre contribution vise à assister les développeurs dans leurs choix de structuration de données. Pour cela nous proposons un générateur qui produit un ensemble d'alternatives de structuration potentiellement intéressantes. Nous avons analysé les différences et les points communs entre les structures possibles et les représentons à l'aide de modèles de caractéristiques. Cette stratégie permet d'exprimer les variantes entre des produits qui, dans notre cas, sont les alternatives de structuration. Dans la suite nous présentons les principes des lignes de produits logiciels qui sont adaptés pour notre générateur de structures.

3. Variabilité et modèles de caractéristiques

Dans les lignes de produits logiciel, il est nécessaire d'exprimer les différences entre les produits à construire. Il existe de nombreuses stratégies pour représenter cette variabilité. La plus utilisée est celle basée sur des modèles de caractéristiques (Kang *et al.*, 1990), dont l'objectif est de modéliser les combinaisons et les différences possibles entre les produits. On distingue les étapes de modélisation, configuration et dérivation, décrites ci-après.

3.1. Modélisation

La variabilité est représentée par un modèle de caractéristiques désigné par fm. Il est composé d'un *ensemble de caractéristiques* $\mathscr{F}$ et de contraintes qui forcent ou interdisent que deux ou plusieurs caractéristiques soient compatibles ou non.

DÉFINITION 1. — *Un Modèle de caractéristiques est défini comme un quadruplet*

$$fm = (\mathscr{F}, \mathscr{L}, rc, \mathscr{D})$$

où, $\mathscr{F}$ *est l'ensemble de caractéristiques,*
$\mathscr{L}$ *représente les liens qui relient les caractéristiques,*
rc *est la caractéristique racine,* $rc \in \mathscr{F}$*,*
$\mathscr{D}$ *est l'ensemble des contraintes entre les caractéristiques de la forme* $X \implies Y$

Les caractéristiques sont organisées dans une structure arborescente permettant de former des groupes de caractéristiques. La Figure 3 illustre les types de liens qui permettent de désigner les variations entre les caractéristiques :

– Une caractéristique *obligatoire* doit être choisie si le parent est choisi. Les caractéristiques obligatoires sont présentes dans toutes les configurations.
– Une caractéristique *optionnelle* peut être choisie si le parent est choisi. Si ni elle ni un de ses fils n'est choisi, la branche dont elle est racine est enlevée.
– Dans un *groupe OR*, une ou plusieurs caractéristiques du groupe doivent être choisies si le parent du groupe est choisi.
– Dans un *groupe XOR*, une unique caractéristique du groupe doit être choisie si le parent du groupe est choisi.

La Figure 4 développe un modèle de caractéristiques pour une *Liseuse*, nommé $fm_{liseuse}$. La caractéristique racine *Liseuse* a deux sous-caractéristiques obligatoires,

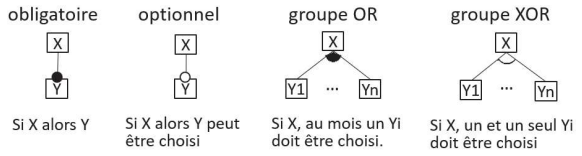

Figure 3. Liens entre les caractéristiques d'un modèle de caractéristiques

Écran et *Dispositif d'entrée*, et une optionnelle, *Son*. À son tour, la caractéristique *Écran*, implique le choix entre un écran *Tactile* ou un écran *Normal*. Par rapport au *Dispositif d'entrée*, il est possible de choisir soit le *Clavier*, soit le *Stylet* soit les deux. Le modèle $fm_{liseuse}$ considère deux contraintes, $\mathscr{D}(fm_{liseuse})$:

- forcer la sélection d'un clavier dans le cas où le son est choisi et,
- empêcher la sélection d'un dispositif d'entrée *Stylet* si un écran normal est choisi.

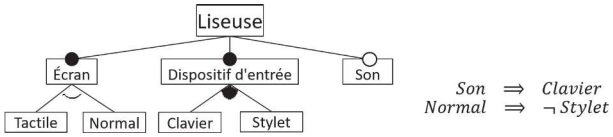

Figure 4. Modèle de caractéristiques $fm_{liseuse}$ pour une Liseuse

3.2. Processus de configuration

Une fois créé le modèle de caractéristiques fm, l'étape de configuration consiste à sélectionner des combinaisons valides entre ses caractéristiques, à savoir la sémantique du modèle notée $\|fm\|$. Une configuration valide est conforme aux contraintes et sera composée d'un sous-ensemble contenant les noms des caractéristiques choisies.

DÉFINITION 2. — *Configuration*
Une configuration C pour un modèle $fm = (\mathscr{F}, \mathscr{L}, rc, \mathscr{D})$ est définie comme un ensemble de caractéristiques sélectionnées parmi l'ensemble des caractéristiques du modèle $\mathscr{F}$. Une configuration est valide si et seulement si elle est conforme aux contraintes $\mathscr{D}$ définies dans le modèle.

$$C(fm) = \{f | f \in \mathscr{F}(fm)\}$$

Dans notre exemple, la sémantique $\|fm_{liseuse}\|$ serait composée de 128 ($2^{7 caractéristiques}$) configurations sans prendre en compte les contraintes. En les considérant, l'ensemble est réduit à 7 configurations valides, parmi lesquelles :

$$C_1(fm_{liseuse}) = \{\text{Liseuse}, \text{Ecran}, \text{Dispositif d'entrée}, \text{Tactile}, \text{Stylet}\}$$

$$C_2(fm_{liseuse}) = \{\text{Liseuse}, \text{Ecran}, \text{Dispositif d'entrée}, \text{Son}, \text{Tactile}, \text{Clavier}\}$$

3.3. Processus de dérivation

Une fois l'ensemble de configurations identifié, chacune est interprétée pour créer une version du produit correspondant aux caractéristiques choisies pour celle-ci.

La figure ci-contre illustre deux des 7 produits finaux du modèle de caractéristiques de la liseuse, noté $\mathscr{P}(fm_{liseuse})$. La dérivation de la configuration C_1 permet de créer une liseuse tactile avec un stylet alors que le produit correspondant à C_2 aura un clavier et une entrée sonore.

4. Génération d'alternatives de structuration orientées document

Nos travaux adoptent l'approche des lignes de produits pour contribuer à la qualité de la structuration des données au sein de systèmes NoSQL orientés documents. La flexibilité et le grand nombre de possibilités offertes par ces systèmes sont aussi bien des atouts que des aspects difficiles à maîtriser. Nous considérons une structure de données ou "schéma" de référence pour la base de documents, comme un produit. La section 4.1 introduit l'approche. Les sections suivantes détaillent les étapes de la ligne de produits basée sur les modèles de caractéristiques qui permettent d'obtenir a partir d'un modèle UML, des alternatives de structuration orientées document.

4.1. Modèles de caractéristiques et structuration orientée document

Nous utilisons un modèle de caractéristiques qui permet de représenter les différences de structuration et de contrôler le nombre de possibilités pour qu'il reste raisonnable. Les différences de structuration seront traitées comme des variantes du produit final, les schémas. Ces variantes seront guidées par des directives de structuration orientées documents telles que l'imbrication, le référencement, la profondeur et la duplication. Les configurations valides résultantes d'une telle modélisation seront donc un ensemble d'alternatives à analyser.

Nous proposons de partir d'un modèle UML donné par l'utilisateur et de créer un modèle de caractéristiques permettant de proposer des alternatives de structuration orienté documents. La Figure 5 illustre le cas de base. Les éléments du modèle UML seront notés comme $mU = (E,R)$ où :

- $E = \{e_1,...,e_n\}$ est l'ensemble des classes
- $R = \{r_1,...,r_n\}$ est l'ensemble d'associations.
- $R(e_i) = \{r_1,...,r_n\}$ est l'ensemble d'associations de la classe e_i
- $E(r_n) = \{e_i,e_j\}$ sont les classes reliées par l'association r_n. $e_i \neq e_j$
- $card(r_n,e_i)$ et $rol(r_n,e_i)$ sont la cardinalité et le rôle de l'association r_n par rapport à la classe e_i.
- $A(e_i) = \{a_1,...,a_n\}$ sont les attributs de la classe e_i. $a_i = \{a_i.name : a_i.type\}$
- $te_i = \{A(e_i),a_{id}\}$ est le type de la classe e_i. Il est composé de l'ensemble d'attributs de la classe et d'un identifiant par défaut [1].

Dans cet article nous nous limitons aux associations binaires sans attributs.

1. Ce type est introduit pour faciliter l'explication de nos propositions.

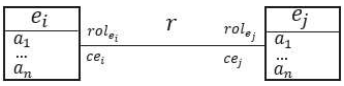

Figure 5. Cas de base : deux classes et une association

4.2. Modélisation de variations entre alternatives

Nous modélisons les variations des alternatives orientées documents avec un modèle de caractéristiques fms permettant d'exprimer les options d'imbrication et référencement. Les principaux choix de modélisation sont les suivants :

1. Une classe e_i peut déclencher la création d'une collection de documents de type te_i.
2. Une association r peut être matérialisée par l'imbrication ou par le référencement de e_i dans e_j ou vice-versa.
3. Une association r peut être matérialisée par une collection-lien.

Pour introduire le modèle de caractéristiques nous allons considérer d'abord le cas simple de deux classes et une association, tel qu'illustré sur la Figure 5. La modélisation de chacune des classes est abordée comme suit.

Modélisation d'une classe
Pour modéliser la variabilité des alternatives de structuration d'une collection pour une classe e_i, nous proposons le modèle de caractéristiques fm_{col}, introduit par la zone grise de la Figure 6a. Ses caractéristiques permettent de traiter l'information propre une classe e_i et une association r :

– La caractéristique "racine", nommée $cole_i$, définit la collection pour la classe e_i.

– Le type des documents de cette collection (au premier niveau) est défini comme une caractéristique fille obligatoire, nommée te_i. Ce type regroupe les attributs de e_i et l'identifiant a_{id}.

La modélisation des associations de la classe introduisent plusieurs variantes. Parmi elles, l'extension du type te_i par ajout d'attributs au niveau 0. Cette possibilité est introduite par la caractéristique optionnelle $te_i\_l_0\_ext$, fille de la caractéristique te_i. Les alternatives de matérialisation de l'association r sont définies par un groupe XOR dépendant de la caractéristique r_role_j. Ce groupe a deux alternatives qui modélisent soit l'imbrication, soit le référencement de documents e_j:

1. La caractéristique $te_i\, EMB^{ce_j}\, te_j$ indique l'*imbrication* dans te_i d'un ou de plusieurs documents du type te_j. Une cardinalité ce_j "plusieurs" indique l'imbrication d'un tableau de documents, une cardinalité 1 indique l'imbrication d'un seul document.

2. La caractéristique $te_i\, REF^{ce_j}\, te_j$ indique le *référencement* depuis un type te_i étendu, d'un ou plusieurs documents du type te_j selon la cardinalité ce_j. Une référence ou un tableau de références. Les références se font en utilisant l'identifiant a_{id}.

Modélisation de deux classes et une association
Basé sur la modélisation de caractéristiques d'une collection, fm_{col}, nous modélisons maintenant la variabilité des alternatives de deux classes et une association. Il s'agit du modèle de caractéristiques fm_{asso} (cf. Figure 6) comme suit :

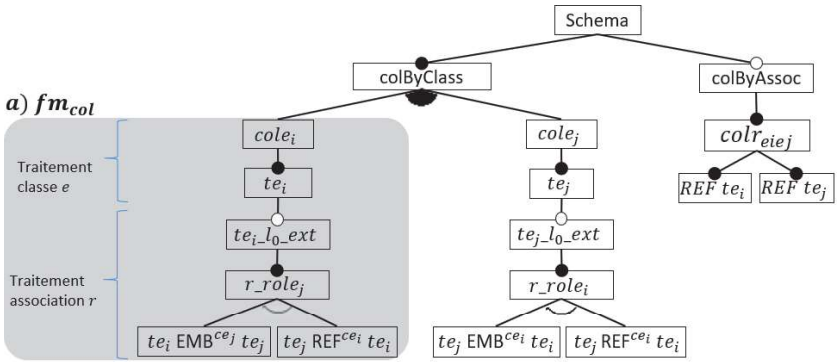

Figure 6. Modèle de caractéristiques fm_{asso} : deux classes et une association

– La caractéristique racine, nommée *Schema*, a deux caractéristiques filles, une obligatoire, nommée *colByClass*, et une optionnelle, nommée *colByAssoc*.

– *colByClass* détermine avec un groupe OR, la ou les collections qui peuvent exister en fonction des classes du modèle UML. Chaque caractéristique fille correspondra à un fm_{col} par classe e_k.

– *colByAssoc* définit une *collection-lien* par association avec une caractéristique nommée $colr_{e_ie_j}$. Celle-la référence des documents de type te_i et te_j.

Modélisation de plusieurs classes et associations
Pour traiter un modèle UML avec plusieurs classes et associations nous proposons l'algorithme *AMISS*, introduit dans la section 5. *AMISS* utilise la modélisation du cas simple (deux classes et une association) que nous venons de présenter. Un parcours du modèle UML crée de manière incrémentale le modèle de caractéristiques complet, nommée fm_s. Ce modèle représente toutes les alternatives de structuration proposées pour une base orientée documents. Le modèle fm_s inclut une collection par classe et par association (pour les éventuelles colletions-lien). Pour chaque collection de classe, le type des éléments inclut les attributs propres à la classe et des possibles attributs supplémentaires pour représenter les associations de la classe.

4.3. Définition des structurations valides

Le modèle de caractéristiques introduit précédemment génère plusieurs alternatives de structuration. Nous avons établi un ensemble de contraintes pour assurer que, par construction, toutes les configurations fournies par le modèle de caractéristiques sont valides. Ces contraintes couvrent les garanties de structuration de la section 2 et servent à diminuer l'explosion combinatoire des possibilités ou des alternatives de configuration. Les contraintes $\mathscr{D}$ sont classées en cinq groupes :

Les contraintes d'isolement $\mathscr{D}^I$, empêchent l'isolement d'une collection. Si une collection de type te_i, au premier niveau, n'a pas d'extension pour son association r_n vers te_j, alors cette association doit être considérée soit depuis une collection *lien* soit depuis un niveau imbriqué appartenant à une autre collection :

$$\mathscr{D}^I = \{ te_i \wedge \neg te_i\_l_0\_ext \implies cole_i e_j \vee r\_role_j \ , \ te_j \wedge \neg te_j\_l_0\_ext \implies cole_j e_i \vee r\_role_i \}$$

Les contraintes d'existence $\mathscr{D}^E$ garantissent l'existence d'une collection référencée. Si un type te_i est référencé, alors une collection du même type doit exister.

$$\mathscr{D}^E = \{ te_i REF^{ce_j} te_j \implies cole_j \ , \ te_j REF^{ce_j} te_i \implies cole_i \}$$

Les contraintes de boucles imbriquées $\mathscr{D}^{BI}$ empêchent les collections d'être référencées entre elles. Si un te_i réfère le type te_j, alors te_j ne peut pas imbriquer te_i.

$$\mathscr{D}^{BI} = \{ te_i REF^{ce_j} te_j \implies \neg te_j EMB^{ce_i} te_i \ , \ te_j REF^{ce_i} te_i \implies \neg te_i EMB^{ce_j} te_j \}$$

Les contraintes de références bouclées $\mathscr{D}^{RB}$ empêchent une collection qui imbrique des documents de type te_i, d'être référencée par une autre collection du même type. Si un te_i réfère le type te_j, alors te_j ne peut pas référencer te_i.

$$\mathscr{D}^{BR} = \{ te_i REF^{ce_j} te_j \implies \neg te_j REF^{ce_i} te_i \}$$

Les contraintes de *liens* d'association $\mathscr{D}^L$ garantissent qu'une *collection-lien* concernant une association r_n ne réfère que des collections dont le type n'est pas étendu par le deuxième type défini par r_n. Si le lien $cole_i e_j$ existe, on peut trouver une collection du type te_i non étendue par te_j et une collection du type te_j non étendue par te_i.

$$\mathscr{D}^L = \{ cole_i e_j \implies (cole_i \wedge \neg r\_role_j) \wedge (cole_j \wedge \neg r\_role_i) \}$$

L'ensemble des contraintes du modèle de caractéristiques fm_{asso} sera donc formé par les contraintes d'isolement, d'existence, de boucles imbriquées, de références bouclées et de lien $\mathscr{D}(fm_{asso}) = \{\mathscr{D}^I, \mathscr{D}^E, \mathscr{D}^{BI}, \mathscr{D}^{BR}, \mathscr{D}^L\}$.

4.4. Configuration des alternatives

Compte tenu des contraintes $\mathscr{D}(fm_{asso})$, le modèle de caractéristiques fm_{asso} fournit huit configurations pour un modèle UML avec deux classes et une association [2]. Ces configurations correspondent aux alternatives de structuration introduites dans la Figure 2. Une configuration contient les noms des caractéristiques choisies. Voici la sémantique $\|fm_{asso}\|$ (configurations) du modèle fm_{asso} :

$$\|fm_{asso}\| = \{C_1, C_2, C_3, C_4, C_5, C_6, C_7, C_8\}$$

$C_1 = \{Schema, colByClass, \mathbf{cole_i}, te_i, te_i\_l_0\_ext, r\_role_j, te_i EMB^{ce_j} te_j\}$

$C_2 = \{Schema, colByClass, \mathbf{cole_i}, te_i, te_i\_l_0\_ext, r\_role_j, te_i EMB^{ce_j} te_j, \mathbf{cole_j}, te_j\}$

$C_3 = \{Schema, colByClass, \mathbf{cole_i}, te_i, te_i\_l_0\_ext, r\_role_j, te_i EMB^{ce_j} te_j,$
$\mathbf{cole_j}, te_j, te_j\_l_0\_ext, r\_role_i, te_j EMB^{ce_i} te_i\}$

$C_4 = \{Schema, colByClass, \mathbf{cole_j}, te_j, te_j\_l_0\_ext, r\_role_i, te_j EMB^{ce_i} te_i\}$

$C_5 = \{Schema, colByClass, \mathbf{cole_i}, te_i, \mathbf{cole_j}, te_j, te_j\_l_0\_ext, r\_role_i, te_j EMB^{ce_i} te_i\}$

$C_6 = \{Schema, colByClass, \mathbf{cole_i}, te_i, te_i\_l_0\_ext, r\_role_j, te_i REF^{ce_j} te_j, \mathbf{cole_j}, te_j\}$

$C_7 = \{Schema, colByClass, \mathbf{cole_i}, te_i, \mathbf{cole_j}, te_j, te_j\_l_0\_ext, r\_role_i, te_j REF^{ce_i} te_i\}$

$C_8 = \{Schema, colByClass, \mathbf{cole_i}, te_i, \mathbf{cole_j}, te_j, colByAssoc, \mathbf{colr_{e_i e_j}}\}$

2. Nous avons utilisé le *SAT solver* S.P.L.O.T. pour calculer le nombre de configurations valides. Cf. http://www.splot-research.org/ et http://52.32.1.180 8080/SPLOT/models/model_20190411_1457420991.xml

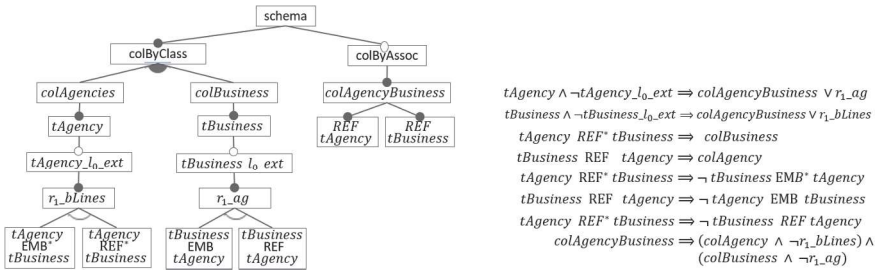

Figure 7. Modèle fm_{asso} correspondant pour le modèle UML de la Figure 1

Les configurations C_1 à C_5 correspondent aux combinaisons avec les choix d'imbrication de documents, alors que C_6 à C_8 correspondent aux choix de référencement. La configuration C_2, par exemple, contient les caractéristiques $cole_i$ et $te_i EMB^{ce_j} te_j$ indiquant l'existence d'une collection $cole_i$ avec imbrication de documents de type te_j. La caractéristique $cole_j$ indique la présence de la collection mais sans étendre son type, en l'absence de caractéristiques du genre EMB ou REF.

Reconsidérons le modèle UML de la Figure 1 avec les classes `Agency` et `Business` et l'association r_1 avec respectivement une cardinalité 1..* et des rôles `ag` et `bLines`. Le modèle de caractéristiques correspondant est introduit dans la Figure 7.

4.5. Dérivation d'une alternative de structuration

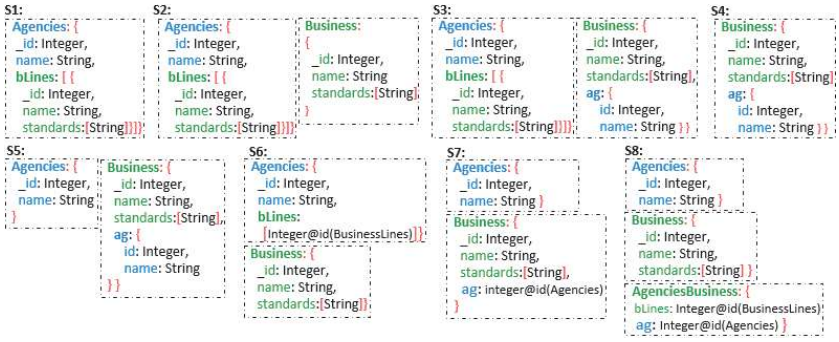

Figure 8. Produits du fm_{asso} de la Figure 7 sous forme d'AJSchema

Afin d'analyser les alternatives de structuration correspondant aux configurations fournies par le modèle de caractéristiques, chaque configuration est dérivée sous forme d'AJSchema. La Figure 8 montre les AJSchemas produits par le modèle de caractéristiques de la Figure 7[3]. Le format AJSchema est assez lisible et met en évidence les types des données. Pour enrichir l'analyse, il est possible d'évaluer automatiquement des métriques structurelles (Gómez *et al.*, 2018b) permettant de quantifier la complexité de la structure pour soit la privilégier, soit l'écarter.

3. Formalisation des structures illustrées par la Figure 2.

Algorithme 1 - AMISS : Algorithme de modélisation de schémas semi-structurés

Input: Modèle UML m, classes avec associations binaires ER
Output: Modèle de caractéristiques fm_s

```
1: fm_s ← null
2: e_i ← random(ER)
3: fm_s ← treatAssociations(fm_s, e_i)
4: fm_s ← treatIsolatedTypes(fm_s, E − ER)

5: function TREATASSOCIATIONS(fm_s, e_i) : fm_s
6:     foreach r_n ∈ R(e_i) do
7:         e_j ← getTarget(r_n, e_i)
8:         fm_asso ← generateFMassociation(e_i, e_j, r_n)
9:         if fm_s is null then
10:            fm_s ← fm_asso
11:        else
12:            fm_s ← FUSIONASSOCIATION(fm_s, fm_asso, r_n, e_i, e_j)
13:        end if
14:        fm_s ← TREATASSOCIATIONS(fm_s, e_j)
15:    end foreach
16:    return fm_s
17: end function
```

5. *AMISS* : Algorithme de ModélIsation de Schémas Semi-structurés

Nous avons vu la modélisation, fm_{asso}, pour deux classes et une association. Nous généralisons maintenant la proposition afin de créer un modèle de caractéristiques, fm_s, pour toutes les classes et associations d'un modèle UML. Pour cela, nous proposons l'*A*lgorithme de *M*odél*I*sation de *S*chémas *S*emi-structurés, *AMISS* (cf. Algorithme 1).

AMISS initie la création du modèle de caractéristiques en partant d'une des classes ayant des associations [4]. Une de ses associations est traitée et donne lieu à un modèle de caractéristiques fm_{asso} (Alg. 1, ligne 8). Ensuite, AMISS parcourt en profondeur les autres classes et associations du modèle UML. Chaque association entraîne la création d'un modèle de caractéristiques fm_{asso} qui est ensuite intégré au modèle complet fm_s (Alg. 1, ligne 12). Cette intégration est faite par fusion des modèles de caractéristiques.

L'algorithme FusionAssociation (cf. Algorithme 2) modifie fm_s pour intégrer la modélisation (représentée dans fm_{asso}) d'une nouvelle association r_n avec ses classes e_i et e_j. La Figure 9 illustre les actions principales (numérotées avec des étoiles) qui enrichissent fm_s.

1. Intégrer la modélisation de r_n dans la collection $cole_i$. Pour cela, ajouter la branche r_role_j du fm_{asso} aux caractéristiques présentes dans fm_s qui imbriquent des documents du type te_i (cf. Lignes 1-9).
2. Ajouter la modélisation de la nouvelle collection créee pour e_j. Le modèle fm_{col} correspondant a la branche $cole_j$ du fm_{asso} est ajouté au groupe OR de la caractéristique *ColByClass* de fm_s (cf. Lignes 10-11).
3. Compléter le type te_i imbriqué dans la nouvelle branche $cole_j$ de fm_s. Pour cela, lui ajouter la modélisation déjà existante des associations autres que r_n. Ces associations

4. Le modèle obtenu est indépendant du choix de la première classe à traiter.

Algorithme 2 - FusionAssociation $(fm_s, fm_{asso}, r_n, e_i, e_j)$: fm_s

▷ Compléter les types te_i à niveau 0 ou imbriqués
1: $fm_{branch} \leftarrow$ extractExtension(fm_{asso}, e_i, r_n) ▷ Extraire branche r_role_j et contraintes
2: $fm_s \leftarrow$ insertBranchCommonClass(fm_s, fm_{branch}, e_i)
3: **foreach** $f \in \mathcal{F}(fm_s) \wedge f = \bigotimes emb^{ce_i} te_i$ **do**
4: **if** f *is leaf* **then**
5: $fm_s \leftarrow$ addFeatureExtension$(fm_s, f, e_i, cse(e_i))$
6: **end if**
7: $fm_{branchV} \leftarrow$ createBranchVersion$(fm_{branch}, e_i.e_j, csr(r_n, e_j))$
8: $fm_s \leftarrow$ embedBranch$(fm_s, fm_{branchV}, f.child)$
9: **end foreach**

▷ Ajouter nouvelle collection $cole_j$
10: $fm_{col} \leftarrow$ extractFMcol(fm_r, e_j, r_n)
11: $fm_s \leftarrow$ insertFMcolNewClass(fm_s, fm_{col}, e_j)

▷ Compléter type te_i imbriqué de $cole_j$
12: $fm_s \leftarrow$ addFeatureExtension$(fm_s, te_j emb^{ce_i} te_i, e_i, cse(e_j))$
13: **foreach** fm_{branch} $filsOf$ $(te_i\_l_0\_ext) \in fm_s \wedge fm_{branch}.root \neq r\_role_j$ **do**
14: $fm_{branchV} \leftarrow$ createBranchVersion$(fm_{branch}, e_i, e_j, csr(r_n, e_i))$
15: $fm_s \leftarrow$ embedBranch$(fm_s, fm_{branchV}, e_j, cse_t)$
16: **end foreach**

▷ Ajouter *collection-lien* pour r_n
17: $fm_s \leftarrow$ insertBranchRelation$(fm_s, fm_{asso}, r_n, e_i, e_j)$

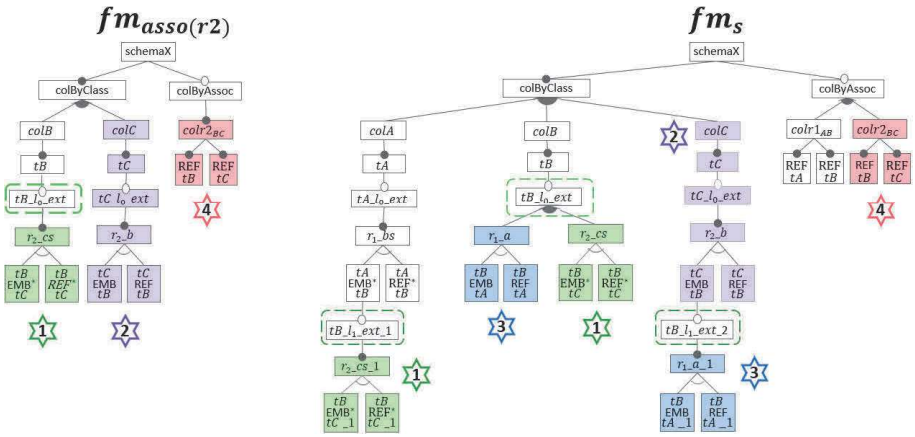

Figure 9. Exemple de fusion: sur le côté gauche modèle fm_{asso} à intégrer, sur la droite modèle fm_s modifié pour intégrer fm_{asso}

se trouvent dans le type te_i de $cole_i$ dans fm_s (cf. Lignes 12-16).
4. Ajouter la *collection-lien* pour r_n. Celle-ci est représentée par la caractéristique $colasso_{e_i e_j}$ de fm_{asso} et doit être insérée dans l'arbre de la caractéristique $colByAssoc$ de fm_s (cf. Ligne 17).

Le modèle fm_s obtenu représente un ensemble de configurations valides où chaque configuration correspond à une alternative de structuration. AMISS crée un arbre de caractéristiques qui peut donner un très grand nombre de configurations valides. Dans le cas de trois classes et deux associations cela correspond à 106 configurations valides. Le nombre de configurations valides croît de manière exponentielle avec le nombre d'associations. Notre préconisation est de restreindre ce nombre en introduisant des contraintes tel qu'illustré dans la section 4.3. Ces contraintes peuvent représenter des préférences de structuration données par l'utilisateur ou être de bonnes pratiques

connues dans le contexte où la base de documents sera utilisée. Par exemple, limiter le nombre d'imbrications de documents, le nombre de références à une collection précise ou le nombre d'attributs complexes par type de document.

6. Mise en œuvre de *ScorusTool*

Le prototype *ScorusTool* implémente la génération de "schémas" mais aussi l'évaluation de métriques structurelles. Ses principaux composants sont illustrés ci-contre.

Le « *Générateur d'alternatives* » met en œuvre les propositions de cet article. Il récupère un modèle UML fourni par l'utilisateur et crée un modèle de caractéristiques afin de fournir un ensemble d'alternatives de structuration orientées document. Le framework *EMF* (Steinberg *et al.*, 2008 ; Eclipse, s. d.) ainsi que la spécification du méta-modèle d'UML (OMG, s. d.) ont été utilisés. Le langage *FAMILIAR* (Acher *et al.*, 2013) a servi pour la création du modèle de caractéristiques. Il permet de manipuler et de raisonner sur les variantes d'un modèle de caractéristiques. L'algorithme *AMISS* a été implémenté à l'aide de *FAMILIAR*. Chacune des configurations, correspondant aux alternatives de structuration, est fournie sous la forme d'un ensemble de noms des caractéristiques sélectionnées (cf. Section 4.4).

Le « *Dérivateur de schémas* » traduit une configuration du modèle fm_s, correspondant à une alternative de structuration de données sous la forme d'AJSchéma (illustré sur la Figure 8) et d'une arborescence, AJTree. Ce composant utilise notamment des bibliothèques de manipulation de graphes en Java. L'AJTree est utilisé pour l'évaluation automatique de métriques structurelles effectué par « *l'évaluateur de métriques* ». Ces métriques ont été présentées dans (Gómez *et al.*, 2018b).

Une interface graphique est proposée à l'utilisateur. Celle-ci montre les configurations fournies par le modèle de caractéristiques fm_s correspondant aux alternatives de structuration. Pour chaque alternative, l'outil montre l'AJTree et l'AJSchéma ainsi que les valeurs des métriques structurelles.

7. Travaux connexes

L'utilisation des modèles de caractéristiques pour générer des alternatives de structuration de données orientées document, s'est avéré pertinente. Cette approche est, à notre connaissance, originale pour la conception de bases de données. Elle permet d'explorer et de gérer un grand nombre d'alternatives de structuration et ouvre des possibilités intéressantes pour la prise en compte de contraintes.

Certains travaux s'intéressent à la modélisation de données pour les bases NoSQL en utilisant d'autres approches. La motivation de la plupart de ces travaux est le choix, presque ad-hoc, d'une structure de données favorable aux performances d'exécution d'un ensemble de requêtes donné. (Zhao *et al.*, 2014) présente un algorithme pour la création systématique d'une base de documents à partir d'un modèle entité-relation. Cet algorithme propose une dé-normalisation de ce modèle avec pré-calcul des join-

tures naturelles par imbrication de documents. La solution engendre en général de la redondance des données.

Des travaux tels que (Mior *et al.*, 2017), (Lombardo *et al.*, 2012) et (Vajk *et al.*, 2013) s'intéressent aux alternatives de "modélisation" des données dans Cassandra (modèle "big table"). Les objectifs des études portent sur le stockage et les performances des requêtes.(Lombardo *et al.*, 2012) propose la création de plusieurs versions des données avec des structures différentes selon les requêtes pré-calculées.

Les approches présentées dans (Abdelhedi *et al.*, 2017 ; Atzeni *et al.*, 2016) sont plus riches que les autres et ciblent plusieurs systèmes avec des modèles de données différents. Ces travaux s'appuient sur un méta-modèle pour représenter les caractéristiques des modèles de données NoSQL et pouvoir créer à partir de la même information, des alternatives de structuration selon le système cible. L'approche de (Abdelhedi *et al.*, 2017) est dirigée par les modèles (Kleppe *et al.*, 2003). Ils proposent de transformer le modèle UML dans un modèle logique générique contenant les concepts communs des trois modèles (Cassandra, Neo4J et MongoDB). Ensuite, une phase de transformation utilise le modèle générique et crée des alternatives de structuration pour chaque système NoSQL. Ils proposent des règles de transformation d'une association selon chaque système. Pour MongoDB, 5 solutions sont proposées. La manière d'interpréter la présence de plusieurs associations dans le modèle n'est pas introduite.

8. Conclusion et perspectives

La flexibilité de la structuration des données offerte par les systèmes NoSQL orientés documents, comme MongoDB, permet de nombreuses options de modélisation, chacune avec ses avantages et ses inconvénients en fonction de plusieurs facteurs.

Tenant compte des enjeux et de la difficulté du choix, nous proposons à l'utilisateur d'étudier plusieurs alternatives semi-structurées à l'aide d'un générateur qui travaille sur un modèle UML des données à gérer. Nous adoptons une approche de lignes de produits logiciel qui permet de créer automatiquement une modélisation de la variabilité des structures susceptibles d'être des "schémas" de référence pour la base de documents. Les algorithmes que nous proposons fonctionnent avec un modèle de caractéristiques où (1) les variantes reflètent la flexibilité du système de types des bases de documents et (2) les contraintes représentent un savoir-faire de modélisation pour obtenir des variantes qui ont du sens. L'explosion du nombre de variantes est ainsi contrôlé mais l'ensemble de contraintes peut évoluer pour intégrer de nouvelles contraintes appropriées au contexte d'utilisation des bases de documents à construire.

Les propositions de cet article sont implantées par le prototype *ScorusTool* qui inclut aussi l'évaluation de métriques structurelles sur les alternatives de structuration. Les perspectives de recherche portent d'abord sur des expérimentations puis sur des extensions pour faciliter l'ajout des contraintes d'un problème particulier afin de réduire le nombre d'alternatives à évaluer.

Remerciements: Nous remercions G. Vega, J. Chavarriaga, JP. Giraudin et les relecteurs anonymes pour leur retours précieux.

Bibliographie

Abdelhedi F., Brahim A. A., Atigui F., Zurfluh G. (2017). MDA-Based approach for NoSQL databases modelling. In *International Conference on Big Data Analytics and Knowledge Discovery*. Springer.

Acher M., Collet P., Lahire P., France R. B. (2013). FAMILIAR: A domain-specific language for large scale management of feature models. *Science of Computer Programming (SCP) , Journal*, vol. 78, n° 6, p. 657-681.

Atzeni P., Bugiotti F., Cabibbo L., Torlone R. (2016). Data modeling in the NoSQL world. *Computer Standards & Interfaces Journal*.

Eclipse. (s. d.). *Eclipse Modeling Framework Project (EMF)*. www.eclipse.org/modeling/emf/. (Accessed: 2018-09-21)

Gomez P. (2018). *Analyse et évaluation de structures orientées document*. Thèse. Université Grenoble Alpes. https://www.theses.fr/2018GREAM076.

Gómez P., Casallas R., Roncancio C. (2016). Data schema does matter, even in NoSQL systems!. In *Research Challenges in Information Science (RCIS) Tenth Intern. Conference on*. IEEE.

Gómez P., Roncancio C., Casallas R. (2018a). Métriques structurelles pour l'analyse de bases orientées documents. In *Actes du xxxvième Congrès INFORSID*.

Gómez P., Roncancio C., Casallas R. (2018b). Towards quality analysis for document oriented bases. In *International Conference on Conceptual Modeling (ER)*. Springer.

Kang K. C., Cohen S. G., Hess J. A., Novak W. E., Peterson A. S. (1990). *Feature-oriented domain analysis (FODA) feasibility study*. Rapport technique. Carnegie-Mellon Univ Pittsburgh Pa Software Engineering Inst.

Kang K. C., Lee J., Donohoe P. (2002). Feature-oriented product line engineering. *IEEE Software Magazine*, vol. 19, n° 4, p. 58–65.

Kleppe A. G., Warmer J., Bast W., Explained M. (2003). *The model driven architecture: practice and promise*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA.

Lombardo S., Nitto E. D., Ardagna D. (2012). Issues in handling complex data structures with NoSQL databases. In *14th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC)*.

Mior M. J., Salem K., Aboulnaga A., Liu R. (2017). Nose: Schema design for NoSQL applications. *IEEE Transactions on Knowledge and Data Engineering*, vol. 29, n° 10, p. 2275–2289.

OMG. (s. d.). *Unified Modeling Language (UML) 2.0*. http://www.omg.org/spec/UML/2.0/. (Accessed: 2018-09-21)

Steinberg D., Budinsky F., Merks E., Paternostro M. (2008). *EMF: eclipse modeling framework*. Livre. Pearson Education.

Vajk T., Feher P., Fekete K., Charaf H. (2013). Denormalizing data into schema-free databases. In *Cognitive Infocommunications (CogInfoCom), 4th International Conference on*. IEEE.

Zhao G., Lin Q., Li L., Li Z. (2014, 11). Schema conversion model of SQL database to NoSQL. In *P2P, Parallel, Grid, Cloud and Internet Computing (3PGCIC), 2014 Ninth International Conference on*.