# SPÉCIFICATION ET UTILISATION D'UN MODÈLE DE DESCRIPTION D'ALGORITHMES D'INDEXATION

Sébastien LABORIE, Dana CODREANU, Florence SÈDES

{laborie, codreanu, sedes}@irit.fr

Université de Toulouse – Université Paul Sabatier – IRIT UMR 5505

118 route de Narbonne – 31062 Toulouse cedex 9

**Mots clefs :**
Algorithme d'indexation, indexation distribuée, indexation à la demande.
**Keywords:**
Indexing algorithm, distributed indexation.
**Palabras clave :**
Indexación algoritmo, indexación distribuida.

## Résumé

Actuellement, de nombreux contenus multimédias sont créés et stockés dans des environnements hétérogènes, éloignées géographiquement, de capacités diverses... Cette masse de données est généralement indexée par des algorithmes qui vont extraire des métadonnées permettant à un utilisateur de rechercher de l'information. De part la diversité et la multitude d'algorithmes d'indexation existants, il n'est pas souhaitable d'exécuter tous ces processus simultanément dans un système d'information. En effet, cela consommerait énormément de ressources et certaines métadonnées pourrait n'être jamais exploitées par le système. De plus, il n'est pas envisageable de déterminer manuellement, avant et pendant l'exécution du système d'information, les algorithmes d'indexation à installer. Dans cet article, nous présentons un modèle de description d'algorithmes d'indexation et montrons comment exploiter ce modèle pour déterminer une liste d'algorithmes d'indexation appropriée par rapport à un ensemble de besoins, de propriétés et de contextes. Également, nous montrons que nos travaux s'intègrent dans le cadre du projet ITEA2 LINDO pour indexer implicitement et explicitement des contenus multimédias.

## Abstract

Nowadays, many multimedia contents are created and stored on remote sites through heterogeneous environments having different capacities. This increasing amount of data is generally indexed by algorithms which extract metadata that will be further queried by a user for retrieving some desired information. Because of the large palette and the diversity of these indexing algorithms, it is not suitable to execute at once every indexing algorithms. Actually, it will overload the information system and will also produce metadata that may not be used inside the system. Moreover, it is cumbersome to determine manually, before and during the execution of the system, the suitable set of indexing algorithms that have to be considered. Consequently, we propose in this paper an indexing algorithm description model and show how to use such model for determining, according specific user needs, properties and contexts, a relevant set of indexing algorithms. Our proposal has been integrated in the ITEA2 LINDO project for indexing implicitly and explicitly multimedia contents.

## 1 Introduction

Ces dernières années de nombreux travaux de recherche ont été menés dans le domaine du traitement du signal et notamment en ce qui concerne les contenus multimédias [10, 8], e.g., l'extraction d'information contenue dans les images et les vidéos [28, 25, 13], la fouille de textes [1], l'analyse de contenus sonores [20], etc. Cela a eu pour conséquence le développement d'une grande palette d'algorithmes d'indexation qui analysent les contenus multimédias pour produire des métadonnées qui seront stockées dans des systèmes d'information et interrogées par les utilisateurs de ces systèmes. Cependant, nous constatons aujourd'hui une grande diversité de ces algorithmes d'indexation en terme de sources de données à traiter en entrée, d'informations extraites en sortie, de contraintes d'exécution, de performances, etc. Dans ce contexte, il n'est pas souhaitable d'exécuter simultanément tous les algorithmes d'indexation possibles dans un système d'information. En effet, cela consommerait énormément de ressources et certaines métadonnées pourrait n'être jamais exploitées par le système. De plus, il n'est pas envisageable de déterminer manuellement, avant et pendant l'exécution du système d'information, les algorithmes d'indexation à installer.

Une question se pose alors : comment déterminer automatiquement une liste d'algorithmes d'indexation la plus appropriée répondant aux besoins des utilisateurs face à des contextes d'exécution spécifiques ?

Pour répondre à cette interrogation, nous proposons, d'une part, de définir un modèle générique de description d'algorithmes d'indexation. En effet, actuellement il n'existe pas un tel modèle mais plusieurs modèles différents propres à chaque algorithme. Puis, d'autre part, nous proposons une méthode flexible de recherche d'algorithmes d'indexation répondant à des besoins ou préférences d'utilisateurs et satisfaisant un contexte d'exécution donné.

Dans un premier temps, nous présentons dans la section 2 cette diversité des algorithmes d'indexation ainsi que des systèmes d'information multimédia qui manipulent différentes variétés d'algorithmes d'indexation. Puis, dans la section 3, nous présentons un modèle de description d'algorithmes d'indexation et montrons comment exploiter ce modèle pour déterminer une liste d'algorithmes d'indexation appropriée par rapport à un ensemble de besoins, de propriétés et de contextes. Enfin, dans la section 4, nous montrons que nos travaux s'intègrent dans le cadre du projet LINDO [1] pour indexer implicitement et explicitement des contenus multimédias. Nous terminons dans la section 5 par une conclusion ainsi que des perspectives.

1. http://www.lindo-itea.eu

# 2 Indexer des contenus multimédias

Dans cette section, nous présentons les travaux de recherche effectués dans le domaine de l'indexation de contenus multimédias en suivant deux axes : les algorithmes d'indexation (§2.1) et les systèmes d'information multimédia gérant plusieurs variétés d'algorithmes d'indexation (§2.2).

## 2.1 Les algorithmes d'indexation

Une définition du processus d'indexation multimédia a notamment été spécifiée dans [3]. Il s'agit de "la modélisation et la représentation de l'information contenue dans les documents sous la forme d'une liste ordonnée de données significatives, appelée index, par un système de gestion de contenus multimédias".

Généralement, un algorithme d'indexation traite en entrée des contenus multimédias, les analyse, en extrait des caractéristiques (e.g., histogrammes de couleurs, statistiques sur le signal, fréquence des mots dans un texte) et produit des informations qui décrivent les médias. Ces informations (ou descripteurs) multimédias sont "des données qui décrivent elles-mêmes le contenu des données" [3], i.e., des métadonnées. Actuellement, de nombreux standards permettent d'encoder des métadonnées multimédias, telle que MPEG-7, Dublin Core, Exif, MXF...

Différents niveaux de description de métadonnées ont été définis dans [3, 4] :

- des descripteurs de bas niveau liés au signal comme, par exemple, les caractéristiques du spectre d'un signal audio, l'histogramme de couleurs d'une image ou des fréquences des mots dans un texte ;
- des descripteurs de niveau moyen qui sont représentés par des concepts comme, par exemple, le nombre d'instruments dans un morceau musicale ou la détection dans une vidéo ou une image de visages, de personnes, de nuages... ;
- des descripteurs de haut niveau qui sont des concepts qui expriment des propriétés très significatives et qui peuvent être exprimés en termes d'autres concepts. Par exemple, le temps, les lieux, les relations entre objets, les différentes actions présentes dans une vidéo, les différents thèmes abordés dans un texte...

Par conséquent, il en résulte que les algorithmes d'indexation sont très divers et très hétérogènes. Nous illustrons leur diversité ci-après selon le type de média traité en entrée.

### 2.1.1 Indexation de contenus textuels

En ce qui concerne les contenus textuels, des approches d'indexation, telles que [15], se sont inspirées des techniques de recherche d'information classique [21] ou de recherche d'information sur le Web, en exploitant les caractéristiques hypertexte, comme les liens entre les pages [2] et les balises HTML [7]. De nombreux travaux de recherche ont également été réalisés afin d'ajouter une couche sémantique à l'indexation textuelle, notamment de faire évoluer l'indexation basée sur les termes à une indexation basée sur les concepts [22] ou sur des modèles et méthodes de représentation des connaissances spécifiques au domaine de l'intelligence artificielle (réseaux neuronaux, réseaux sémantiques, réseaux bayesiens) [19]. Des techniques comme la désambigüisation du sens des mots [18] (par l'étiquetage des mots selon l'appartenance aux catégories lexicales ou syntaxiques) en utilisant des ressources disponibles sur le Web (comme les dictionnaires, les thesaurus ou les ontologies, e.g., la ressource lexicale la plus utilisée est WordNet [2]), l'extraction de concepts selon une ontologie [12] ou la classification des parties du texte selon les thèmes detectés [11] représentent également le sujet de beaucoup de travaux récents.

2. http://wordnet.princeton.edu/

### 2.1.2 Indexation de contenus sonores

Selon [3], les descripteurs audio de bas niveau sont classifiés en fonction du traitement appliqué pour les extraire. Voici quelques exemples (calculés sur un segment d'un échantillon à quelques dixièmes de seconde) :

- Descripteurs temporels : taux de silence [24], corrélation croisée, énergie à court terme [30], volume [17], etc. ;
- Descripteurs d'énergie : énergie globale, énergie harmonique, énergie du bruit, énergie de bandes [23] ;
- Descripteurs spectraux : transformée de Fourier à court terme (STFT) ou transformée en ondelettes (DWT) [26], centroïde spectral [24] , flux spectral [24], point de "roll-off" [24], etc.

En guise d'exemple, [17] propose un algorithme d'indexation de contenus sonores qui extrait et utilise des descripteurs de bas niveau pour réaliser la classification d'émissions de télévision à l'aide d'un classificateur basé sur des réseaux neuronaux. Dans [11], le système d'indexation de contenus multimédias contient un algorithme de segmentation acoustique. Les segments résultés sont analysés par un algorithme qui fait la reconnaissance automatique de la parole et sépare les segments qui contiennent des paroles avec ceux comportant du silence. La transcription des paroles est ensuite fournie à un autre algorithme d'indexation textuelle qui réalise une classification des segments de texte selon plusieurs concepts. Le système Transcriber [3] quant à lui réalise une transcription du discours sous forme textuelle.

### 2.1.3 Indexation d'images et des vidéos

Les travaux de recherche dans le domaine de l'indexation du contenu visuel sont très nombreux. Généralement, les vidéos sont considérées comme une superposition d'une suite d'image et d'un contenu audio (l'analyse de l'audio a été présentée dans la section précédente). Les systèmes utilisent des démultiplexeurs qui séparent le contenu audio du contenu visuel et traitent les contenus séparément, e.g., [9]. Nous nous focalisons dans cette partie sur le contenu visuel.

[27, 14] ont élaboré différentes synthèses des principales techniques utilisées en analyse d'images. Une liste de travaux de recherche concernant l'indexation d'images est également disponible à cette adresse : `http://www.visionbib.com/bibliography/applicat804.html`.

Les contenus visuels peuvent être décrit par des caractéristiques classifiées en trois niveaux [4] :

- un niveau primitif : des statistiques au niveau du signal de l'image comme la couleur, la texture, les formes, la localisation spatiale des éléments, etc. ;
- un niveau sémantique local : des concepts de plus haut niveau comme les objets reconnus ou les visages identifiés dans une image ;
- un niveau sémantique global : des concepts qui décrivent le sujet de l'image ;

En général, les caractéristiques des niveaux supérieurs sont obtenues des caractéristiques de plus bas niveau et non directement du contenu. Parmi les travaux concernant l'extraction de caractéristiques primitives, on retrouve, par exemple, les systèmes suivants : QBIC [5] qui propose des extracteurs et un système de recherche d'images basés sur la couleur, la texture et la forme et PicToSeek [6] qui envisage l'utilisation de la combinaison des invariants de couleurs et de formes afin d'indexer et de rechercher des images.

En ce qui concerne l'extraction des caractéristiques de plus haut niveau, les travaux de recherche se focalisent sur la reconnaissance d'objets, comme dans [16, 29] ou la détection de visages, comme dans A4Vision [4] et OpenCV [5].

---

3. `http://trans.sourceforge.net/en/presentation.php`
4. `http://www.a4vision.com/`
5. `http://sourceforge.net/projects/opencvlibrary/`

## 2.2 Systèmes d'information multimédia gérant plusieurs algorithmes d'indexation

De nombreuses plates-formes d'indexation de contenus multimédias gérant plusieurs algorithmes d'indexation ont été développés. Le tableau 1 compare différents systèmes selon leur généricité, leur architecture, leur capacité à traiter les médias selon différentes modalités et leur flexibilité en terme d'ajout d'indexeur.

TABLE 1: Comparaison des systèmes d'information multimédia.

| Système | Objectif principal | Domaine ciblé | Architecture | | | Indexation | | Nombre d'algos | |
|---|---|---|---|---|---|---|---|---|---|
| | | | Centr. | P2P | SOA | Mono | Cross | Fixé | Variable |
| [11] | indexation automatique du contenu MM | + | + | − | − | + | − | + | − |
| Weblab | plateforme pour l'integration des composantes qui font le traitement du contenu MM | − | + | − | + | − | + | − | + |
| Vitalas | indexation et recherche crossmédia | − | + | − | + | − | + | − | + |
| K-Space | l'inférence sémantique pour l'annotation semi-automatique et la recherche des contenus MM | − | + | − | + | + | − | − | + |
| Candela | analyse du contenu vidéo distribution dans le réseau fonctionnalités de stockage | + | + | − | − | + | − | − | + |
| Muscle | utilisation de l'apprentissage statistique pour la générationu (semi) automatique des métadonnées sémantiques | − | + | − | − | + | − | − | + |
| [9] | intégration sur une plateforme des algorithmes d'indexation et chainage automatique | − | + | − | + | + | − | − | + |
| Video Scout | segmentation et indexation des programmes TV | + | + | − | − | − | + | + | − |
| Lindo | indexation distribuée à grande echelle | − | + | − | − | + | − | − | + |
| Sapir | recherche à grande échelle dans du contenu audio-visuel | − | − | + | − | + | − | + | − |

Suite aux observations faites dans les sections 2.1 et 2.2, nous pouvons conclure :

– qu'il y a une grande diversité d'algorithmes d'indexation, en ce qui concerne leurs entrées, leurs sorties, leurs contextes d'exécution, leurs paramètres et leurs performances. De plus, il n'existe pas de modèle générique de description de ces algorithmes ;

– que même s'ils existent de nombreux systèmes gérant de multiples algorithmes d'indexation, aucun d'entre eux ne définit une méthode automatique déterminant des algorithmes d'indexation appropriés par rapport aux besoins des utilisateurs et des contextes d'exécution.

# 3 Spécifier et utiliser un modèle de description d'algorithmes d'indexation

Dans un premier temps, nous présentons un modèle générique de description d'algorithmes d'indexation (§3.1). Puis, nous proposons un algorithme qui va exploiter ce modèle pour déterminer un ensemble d'algorithmes d'indexation approprié par rapport à un ensemble de besoins, de propriétés et de contextes (§3.2).

## 3.1 Un modèle générique de description d'algorithmes d'indexation

Pour construire un modèle générique de description d'algorithmes d'indexation, nous avons effectué une étude sur une vingtaine d'outils d'indexation utilisés notamment dans le cadre de projets de recherche, tels que KLIMT [6] et LINDO [11]. Nous y trouvons des outils d'analyse et de segmentation de contenus sonores, comme par exemple un outil de segmentation audio en segments de parole, de musique, de bruit et de silence ; un outil d'identification automatique des langues ; un outil de transcription automatique de la parole en texte (français ou anglais) ; un outil qui reconnait les locuteurs d'un discours, etc. Nous trouvons également des outils d'analyse et de segmentation de contenus vidéos, comme, par exemple, des outils de segmentation en plans, de détection de personnes, un outil d'extraction d'images clés de vidéo. En ce qui concerne les images, nous disposons d'un outil de détection de visages, un outil qui extrait les caractéristiques globales de l'image, etc. Pour les documents textuels, nous disposons d'un outil de segmentation selon le thème ainsi que d'un outil de reconnaissance d'entités nommées.

En observant les caractéristiques communes et discriminatoires de tous ces algorithmes d'indexation, nous avons identifié six groupes de caractéristiques suivants :

- **caractéristiques générales** comme, par exemple, le nom, l'auteur (personne ou entreprise), le type de média en entrée, le langage de programmation dans lequel l'algorithme est écrit, l'URL où l'algorithme peut être trouvé, la complexité de l'algorithme... ;
- **les paramètres d'entrée** que l'on connaît a priori (concernant les données d'apprentissage et de tests, et les caractéristiques spécifiques pour chaque type d'entrée) et dont la modification peut engendrer des résultats différents ;
- **les caractéristiques de sortie** c'est-à-dire le type et la description des objets qui sont extraits par l'algorithme ;
- **les mesures de performance**, en effet pour chaque ensemble de données de tests cette catégorie indique quels sont les résultats obtenus par l'algorithme en terme de rappel et de précision ;
- **les contraintes d'exécution** au sujet des conditions qui doivent être satisfaites pour que l'algorithme s'exécute dans les meilleures conditions ;
- **les chaînes d'algorithmes** c'est-à-dire la spécification d'algorithmes pouvant être composés de plusieurs algorithmes.

Nous verrons par la suite que toutes ces caractéristiques peuvent servir à la sélection des algorithmes selon des critères données (e.g., la description de l'objet en sortie, le type de média en entrée, les contraintes de plate-forme), à la comparaison des algorithmes (selon la complexité ou les performances) ainsi qu'à la validation des algorithmes (vérifier les paramètres d'entrée ou les contraintes d'exécution).

La représentation UML du modèle que nous proposons est présentée dans la figure 1. Elle contient les caractéristiques qui ont été mentionnées auparavant structurées dans cinq classes principales : AlgorithmModel, InputParam, OutputObj, Performance et ExecutionConstraints. Les chainages sont représentés par la relation isChained autour de la classe AlgorithmModel.

Des exemples d'instanciation de notre modèle en XML sont également disponibles à l'adresse suivante : `http://www.irit.fr/PERSONNEL/SIG/laborie/LINDO/listingAlgos.php`

6. `http://www.klimt-project.org`

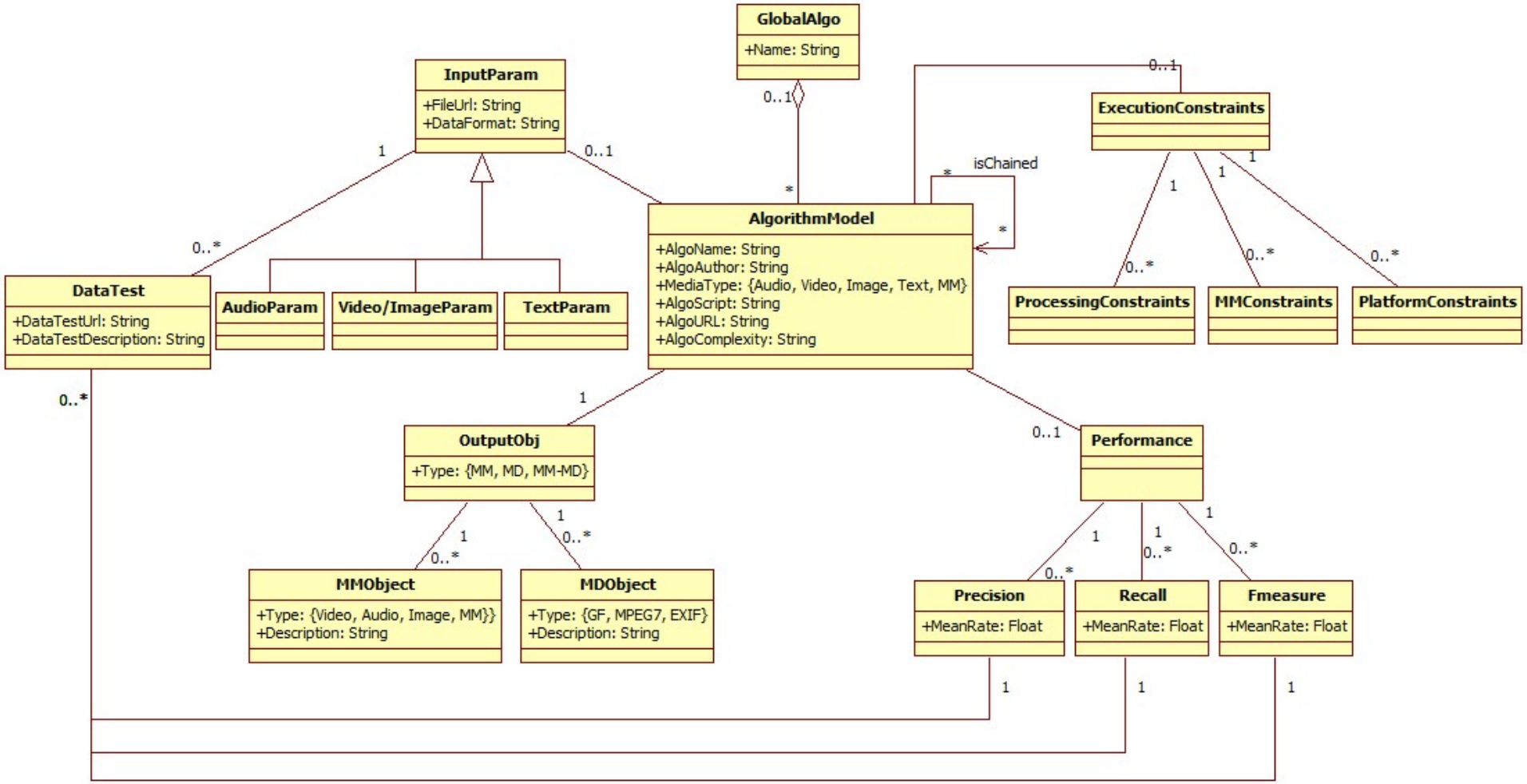


FIGURE 1: Un modèle générique de description d'algorithmes d'indexation.

### 3.2 Procédure de sélection d'algorithmes d'indexation

L'objectif de notre procédure de sélection est de prendre en compte des besoins d'utilisateurs ainsi que des contextes d'exécution pour fournir une liste d'algorithmes d'indexation appropriée permettant d'extraire les informations désirées.

Les éléments qui vont être considérés à l'entrée de notre procédure de sélection sont les suivants :

- la requête (sous forme de termes), i.e., R = $t_1$, $t_2$, ..., $t_m$ ;
- les descriptions d'algorithmes d'indexation correspondant au modèle que nous avons présenté dans la figure 1 ;
- les contextes d'exécution des algorithmes d'indexation, par exemple la description de chaque serveur faisant partie du système d'information multimédia comme leur système d'exploitation, le type et la fréquence de leur processeur, leurs caractéristiques (e.g., s'il est capable de traiter du texte, des images, des vidéos, etc.), leurs performances au niveau du réseau, etc. Dans cet article, nous allons considérer uniquement deux éléments pour le contexte : le système d'exploitation et la fréquence du processeur (C = {OS, CPU}) ;
- des propriétés et des contraintes selon les préférences de l'utilisateur comme le type de média recherché (e.g., des émissions TV, des journaux TV, des discours audio), le format de média recherché, le nombre maximum d'algorithmes à exécuter, des préférences pour les algorithmes qui extraient beaucoup d'éléments ou pour des algorithmes qui extraient moins d'éléments, des préférences pour des chaînes d'algorithmes longues ou plus courtes, des préférences sur la redondance de caractéristiques extraites... Dans cet article, nous considérerons uniquement la contrainte et les propriétés suivantes :
  - C1 : le format de média recherché : Audio, Video, Image, Texte, AudioVisuel, ImageText ;
  - P1 : les algorithmes qui extraient beaucoup d'éléments ou les algorithmes qui extraient peu d'éléments ;
  - P2 : la redondance des caractéristiques ou non ;
  - P3 : les chaines longues ou les chaines courtes ;
  - P4 : le nombre maximum d'algorithmes.

Notre procédure de sélection d'algorithme d'indexation va suivre les étapes suivantes :

**Étape 1 :** Calcul des listes d'algorithmes qui extraient les éléments de la requête

**a)** Pour chaque mot de la requête, une liste d'algorithmes d'indexation qui l'extrait est construite (chaque mot est cherché dans le champ Description de l'OutputObject du modèle) ;

**b)** Les algorithmes qui ne respectent pas les conditions du contexte, comme par exemple l'OS et le CPU (ces valeurs sont comparées avec les champs de la classe PlatformConstraints du modèle) sont éliminés. Une liste d'algorithmes distincts qui extraient les éléments de la requête est créée.

**Étape 2 :** Calcul des listes inverses

**a)** Pour chaque algorithme sélectionné à la fin de l'étape 1, on calcule la liste des éléments de la requête qu'il extrait à partir de la liste construite dans l'étape précédente ;

**b)** Un score est associé à chaque algorithme. Ce score peut dépendre, comme nous l'avons envisagé dans cet article, du nombre d'éléments extraits.

**Étape 3 :** Calcul des listes du résultat

**a)** La liste des algorithmes est triée selon les scores. Par exemple, si l'on souhaite les algorithmes qui extraient le plus d'éléments, le tri sera décroissant. Sinon, le tri sera croissant ;

**b)** En fonction de la liste triée des algorithmes et des scores, une procédure de recherche va identifier des combinaisons d'algorithmes permettant d'identifier tout ou une partie des éléments de la requête.

**Étape 4 :** Calcul des chaînes d'algorithmes

**a)** Pour chaque algorithme du résultat (calculé à l'étape 3), on vérifie que celui-ci peut traiter le type de média donné en entrée. Si cela n'est pas possible et qu'il fait partie d'une chaîne d'algorithmes d'indexation, on vérifie si l'un de ses prédécesseurs peut traiter le média. Si tel n'est pas le cas, le résultat contenant l'algorithme est supprimé, sinon on remplace l'algorithme par la chaîne de traitement. À la fin de cette étape, on peut donc disposer de listes de résultats contenant des chaînes d'algorithmes qui répondent à la requête initiale, e.g., des résultats de la forme [[A1→A2→A3] et A5] et [[A0→A6→A7→A3] et A5] ;

**b)** Pour chaque liste du résultat, on vérifie la propriété du nombre maximum d'algorithmes. Les listes qui contiennent un nombre d'algorithmes plus grand que le nombre précisé par l'utilisateur sont éliminées ;

**c)** Les listes résultats sont ordonnées de façon ascendante selon les tailles des résultats, i.e., le nombre d'algorithmes qui font partie d'une liste du résultat.

### 3.3 Un exemple d'exécution de notre procédure de sélection d'algorithme d'indexation

Nous proposons d'illustrer notre procédure de sélection d'algorithmes d'indexation au travers d'un exemple. Tout d'abord, nous énumérons une partie de la collection de description d'algorithmes. Pour chaque algorithme, nous présentons son nom, le type de média qu'il peut traiter en entrée ainsi que ses contraintes d'exécution. Le graphe de la figure 2 illustre des chaînes d'application d'algorithmes que nous avons pré-définis :

- A1 : Acoustic Segmentation : Audio, {OS=Windows, CPU=2.2GHz} ;
- A2 : Car detection in a parking : Video, {OS=Windows, CPU =2GHz} ;
- A3 : Cars color recognition : Video, {OS=Windows, CPU =2GHz} ;
- A4 : Color recognition : Image, {OS=Windows, CPU =2GHz} ;
- A5 : Demultiplexing : AudioVisual, {OS=Windows, CPU ="2GHz} ;
- A6 : Detecting Persons : Video, {OS=Linux, CPU=2.7GHz} ;
- A7 : Face Recognition : Image, {OS=Linux, CPU=2GHz} ;
- A8 : Feature Extractor : Video, {OS=Windows, CPU=2.5GHz} ;
- A9 : Key Images Extractor : Video, {OS=Windows, CPU=2GHz} ;
- A10 : Language Detection : Text, {OS=Windows, CPU=2GHz} ;
- A11 : Named Entity Recognition : Text, {OS=Windows, CPU=2GHz} ;
- A12 : Pedestrian Recognition : Image, {OS=Windows, CPU=2GHz} ;
- A13 : Person detection : Image, {OS=Windows, CPU=2GHz} ;
- A14 : Person detection in moving cars : Video, {OS=Windows, CPU=2GHz} ;
- A15 : Scenes Resuming Images Extraction : Video, {OS=Windows, CPU=2GHz} ;
- A16 : Shot change detection : Video, {OS=Windows, CPU=2.5GHz} ;
- A17 : Speech Diarization : Audio, {OS=Linux, CPU=2.5GHz} ;
- A18 : Topic Segmentation : Text, {OS=Windows, CPU=2GHz} ;

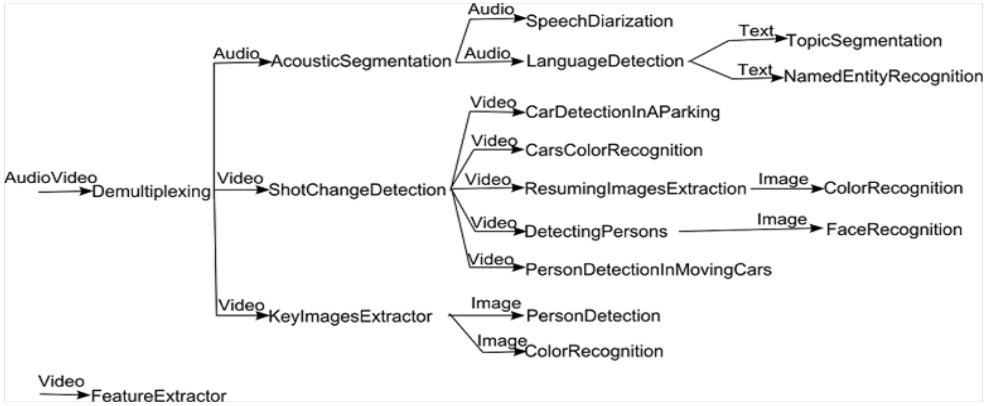


FIGURE 2: Graphe de chaines d'application d'algorithmes d'indexation.

Nous considérons la requête R={person,car,color,parking} ainsi que les préférences suivantes : (1) nous cherchons des algorithmes qui prennent en entrée des vidéos, (2) nous souhaitons exécuter des algorithmes qui extraient beaucoup d'éléments ainsi que (3) des chaînes d'application d'algorithmes courtes, enfin (4) nous ne souhaitons pas exécuter des algorithmes qui extraient des éléments qui sont déjà extraits par d'autres algorithmes de la solution.

**Algorithme 1:** Les trois premières étapes de la section 3.2.

**Entrées** : Une liste de termes ($t_i$) qui constitue la requête $R$, une liste avec les valeurs des éléments du contexte $C$ et les valeurs des propriétés $P1$, $P2$ et $P4$.

**Sorties** : $\mathcal{L}$ est la liste des listes des algorithmes qui extrait les éléments de la requête et satisfait le contexte ainsi que les propriétés données.

1 **pour** *chaque terme* $t_i$ *de* $R$ **faire**
2 | **pour** *chaque algorithme* $A_j$ *de la base de données* **faire**
3 | | **si** $OutputObjetDescription(A_j)$ *contient* $t_i$ **alors**
4 | | | **si** $OS(C) = PlatformConstraintsOS(A_j)$ *et* $CPU(C) >= PlatformConstraintsCPU(A_j)$ **alors**
5 | | | | $ListeAlgos_i \leftarrow ListeAlgos_i \cup \{A_j\}$;
6 | | | | $AlgoSolution \leftarrow AlgoSolution \cup \{A_j\}$;
7 | | | **fin**
8 | | **fin**
9 | **fin**
10 **fin**
11 **pour** *chaque algorithme* $A_i$ *de AlgoSolution* **faire**
12 | $ListeInverse_i = les\ elements\ de\ la\ requete\ qui\ sont\ extraits\ par\ A_i$;
13 | $Score_i = nombre\ d'elements\ de\ la\ requete\ qui\ sont\ extraits\ par\ A_i$;
14 **fin**
15 Construction de la liste des listes AlgoInfo=$\{\{A_i, Score_i, ListeInverse_i\}\}$;
16 **si** *P2=algorithmes qui extraient peu d'elements* **alors**
17 | Trier(AlgoInfo,ascendant);
18 **fin**
19 **sinon**
20 | Trier(AlgoInfo,descendant);
21 **fin**
22 Rechercher(AlgoInfo,$\emptyset$,0) ;

Déroulement de l'algorithme 1 sur notre exemple :

**Étape 1 (lignes 1 à 10) :** Pour chaque terme de la requête, une liste d'algorithmes qui l'extraient est construite (*AlgoList*). La liste des algorithmes différents qui extraient des termes de la requête est *AlgoSolution*.
- $ListeAlgos_1$ = {A6, A13, A14} ;
- $ListeAlgos_2$ = {A2, A3, A4, A14} ;
- $ListeAlgos_3$ = {A3,A4, A8} ;
- $ListeAlgos_4$ = {A2, A3, A4, A14} ;
- $ListeAlgosDifferents$ = {A2, A3, A4, A6, A8, A14}.

**Étape 2 (lignes 11 à 15) :**
- $ListeInverse_{A2}$ = {car,parking} ;
- $ListeInverse_{A3}$ = {car, color, parking} ;
- $ListeInverse_{A4}$ = {color, parking,car} ;
- $ListeInverse_{A6}$ = {person} ;
- $ListeInverse_{A8}$ = {color} ;
- $ListeInverse_{A13}$ = {person} ;
- $ListeInverse_{A14}$ = {person, car, parking} ;
- $Score_{A2}$ = 2 ;
- $Score_{A3}$ = 3 ;
- $Score_{A4}$ = 3 ;
- $Score_{A6}$ = 1 ;
- $Score_{A8}$ = 1 ;
- $Score_{A13}$ = 1 ;
- $Score_{A14}$ = 3.

**Étape 3 (lignes 16 à 21) :**
- $AlgoInfo$={ {A3,$ListeInverse_{A3}$,3}, {A4,$ListeInverse_{A4}$,3}, {A14,$ListeInverse_{A14}$,3}, {A2,$ListeInverse_{A2}$, 2}, {A6,$ListeInverse_{A6}$,1}, {A8,$ListeInverse_{A8}$,1}, {A13,$ListeInverse_{A13}$,1} }.

Dans la suite, nous détaillons la méthode *R*echercher (ligne 22) qui correspond à l'étape 3.b présentée dans la section 3.2.

La méthode *R*echercher prend en entrée trois éléments :

- S1 : un sous-ensemble de AlgoInfo, donc une liste d'algorithmes qui extraient les éléments de la requête. Comme nous avons pu le constater dans l'Algorithme 1, la procédure de recherche prendra en entrée une liste triée selon les scores (en fonction de la préférence P1) ;
- S2 : l'ensemble d'algorithmes courant étant considérés comme de bons candidats ;
- NombreElementsExtraits : nombre d'éléments de la requête qui peuvent être extraits avec S2 ;

Initialement, S1 correspondra à la liste AlgoInfo, S2 sera vide et il n'y aura aucun élement permettant d'extraire des éléments de la requête à partir de S2 (étant donné que S2 est vide).

La méthode *R*echercher manipulera également en interne plusieurs données :

- $\mathcal{L}$ : la liste résultat qui sera initialement vide ;
- $\mathcal{L}$incomplet : une liste de résultat incomplet qui sera initialement vide. Cette liste est utilisée dans le cas où il n'existe pas d'algorithmes permettant d'extraire tous les éléments de la requête. En effet, cette liste contiendra des algorithmes permettant de répondre à une majorité d'éléments de la requête.

La méthode *R*echercher est une fonction récursive, c'est notamment un algorithme de backtrack qui permet de trouver des combinaisons d'algorithmes satisfaisant les besoins des utilisateurs et les contraintes d'exécution des algorithmes. L'algorithme 2 illustre le code de cette méthode.

Cette méthode s'arrêtera lorsque les algorithmes considérés dans S2 extrairont tous les éléments de la requête. Dans ce cas, S2 sera ajoutée dans le résultat $\mathcal{L}$ (lignes 1 à 3). Dans le cas où certains éléments de la requête pourront être extraits mais qu'il n'y aura plus d'algorithme à considérer alors la solution incomplète sera ajoutée à la liste $\mathcal{L}$incomplet (lignes 4 à 6).

À chaque étape de l'algorithme, on considérera uniquement les meilleurs candidats satisfaisant la propriété P1, c'est-à-dire ceux qui ont un score maximum ou minimum (lignes 7 à 15). Chaque meilleur candidat est ajouté à la liste S2 et il est également retiré de la liste S1 (pour ne plus le considérer dans les pas récursifs suivants). Dans le même temps, en fonction des éléments qu'extrait l'algorithme ajouté précédemment dans S2, on recalcule les scores des algorithmes de S1 (lignes 17 à 31). La méthode *R*echercher est une nouvelle fois appelée avec les nouveaux paramètres d'entrée mis à jour et ainsi de suite jusqu'à trouver une combinaison d'algorithme approprié permettant de répondre à la requête (ligne 33).

La méthode *R*echercher est illustrée sur l'exemple présentée dans cette section. Notons que celle-ci peut-être optimisée en utilisant des techniques de backmarking ou de forward checking.

**Algorithme 2:** Rechercher

```
1  if NombreElementsExtraits=cardinalite(R) then
2      ℒ.add(S2) ;
3  end
4  else if S1=Vide then
5      ℒincomplet.add(S2) ;
6  end
7  else
8      copyS1=clone(S1) ;          /* Duplication de S1 et S2 */
9      copyS2=clone(S2) ;
10     if P1=algorithmes qui extraient beaucoup d'éléments then
11         listBestAlgos=les algorithmes A_i pour lesquels Score_i=Score_max ;
12     end
13     else if P1=algorithmes qui extraient moins d'éléments then
14         listBestAlgos=les algorithmes A_i pour lesquels Score_i=Score_min ;
15     end
16     copyS1=clone(S1) ;
17     for chaque algorithme A_i de listBestAlgos do
18         copyS1.remove(A_i) ;
19         copyS2.add(A_i) ;
20         for chaque algorithme A_j de copyS1 do
21             Score(A_j)=Cardinalite(ListeInverse(A_j)-ListeElementsExtraits) ;
22             if Score(A_j)=0 then
23                 if P2=true then
24                     copyS2.add(A_j) ; copyS1.eliminer(A_j) ;
25                 end
26                 else if P2=false then
27                     copyS1.eliminer(A_j) ;
28                 end
29             end
30         end
31         ListeElExtraits ← ListeElExtraits ∪ ListeInverse(A_i);
32         if copyS2.size()<=P4 then
33             Rechercher(copyS1,copyS2, NombreElementsExtraits+Score(A_i)) ;
34         end
35         copyS1=clone(S1) ;
36         copyS2=clone(S2) ;
37     end
38 end
```

La figure 3 présente l'arbre de recherche de la méthode *R*echercher sur l'exemple que nous avons présenté tout au long de cette section.

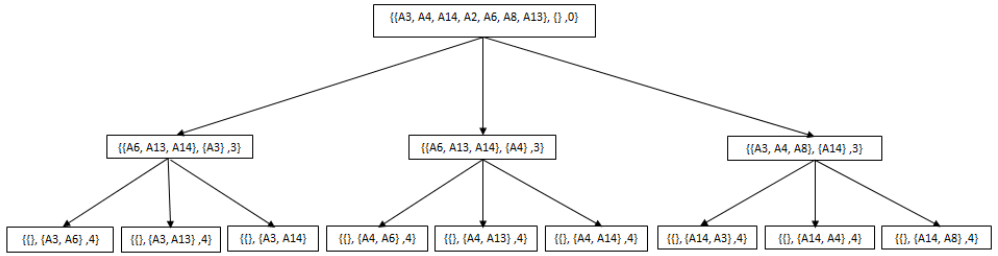


FIGURE 3: Arbre de recherche des solutions.

Après l'exécution de la méthode *R*echercher, nous avons la liste de solutions suivante : {{{A3, A6}, {A3, A13}, {A3, A14}}, {{A4, A6}, {A4, A13}, {A4, A14}}, {{A14, A3}, {A14, A4}, {A14, A8}}}.

Après élimination des doublons (e.g., {A4, A14} et {A14, A4}), nous aurons la liste suivante : {{{A3, A6}, {A3, A13}, {A3, A14}}, {{A4, A6}, {A4, A13}, {A4, A14}}, {A14, A8}}}.

L'extraction des chaînes d'application d'algorithmes d'indexation est réalisée par l'Algorithme 3. Pour ce faire, on parcourt chaque liste d'algorithme. Par exemple, A13 (Person Detection) est un algorithme d'une des listes ci-dessus. Cependant, A13 ne permet pas de traiter des contenus vidéos mais uniquement des images (cf., figure 2). Or, A13 fait partie d'une chaîne d'application d'algorithmes. En effet, l'algorithme A9 (Key Image Extractor) permet de traiter des contenus vidéos pour ainsi fournir à A13 des images à traiter. Exécuter l'algorithme A9 puis A13 est donc une solution possible permettant de répondre à la requête. Cette chaîne sera notée : A9 $\rightarrow$ A13.

Bien évidemment, dans le cas où un algorithme ne peut traiter le média en entrée ou bien que cet algorithme ne fait pas partie d'une chaîne qui permet de traiter le média en entrée, alors la solution contenant cet algorithme sera éliminée du résultat.

**Algorithme 3:** La fonction qui extrait la meilleure chaine

**Entrées** : Le nom de l'algorithme algoName, algorithmsChains=la liste des chaines définies a priori et le format de média désiré par l'utilisateur mediaFormat (définit par l'intermède de la contrainte C1)

**Sorties** : La sous chaine la plus courte qui finit avec algoName et commence avec un algorithme qui prend en entrée le mediaFormat

```
1  si dans algorithmsChains il y a au moins une chaine qui contient algoName
   alors
2      pour chaque chaine C_i qui contient algoName faire
3          indexAlgo=la position de algoName dans la chaine ;
4          pour chaque algorithme A_j de la chaine C_i pour lequel j>indexAlgo
           faire
5              si mediaType(A_j)=mediaFormat alors
6                  extraction(subchain) ;
7                  relevantChains.add(subchain) ;
8              fin
9          fin
10     fin
11     bestChain=minSize(relevantChains) ;
12 fin
```

L'algorithme 3 correspond à l'étape 4 présentée dans la section 3.2. Il considère en entrée les listes qui constituent le résultat, issues de la troisième étape.

Après l'extraction des chaînes et l'ordonnancement croissant par rapport aux nombres d'algorithmes composant la solution, nous aurons les résultats suivants :

1. {Person Detection in moving cars}, {Feature Extractor} ;
2. {Person Detection in moving cars}, {Color recognition} ;
3. {Person Detection in moving cars}, {Cars color recognition} ;
4. {Color recognition}, {Detecting Persons } ;
5. {Cars color recognition}, {Detecting Persons } ;
6. {Color recognition}, {Key Images Extractor - > Person Detection} ;
7. {Cars color recognition}, {Key Images Extractor -> Person Detection}.

Bien sûr, il est possible de choisir automatiquement le premier résultat. Cependant, il peut être judicieux de proposer au développeur du système d'information l'ensemble des solutions appropriées permettant de répondre à la requête et aux besoins des utilisateurs. Egalement, il est possible de raffiner l'ordre de présentation des résultats en ajoutant des priorités d'exécution sur certains algorithmes d'indexation, etc.

## 4 Prototype et Application

Nous avons développé un prototype qui permet de créer et de stocker des descriptions d'algorithmes en se basant sur le modèle que nous avons présenté dans la section 3.1. En se basant sur les descriptions d'algorithmes, notre prototype permet également de définir des chaînes d'application d'algorithmes. Enfin, celui-ci permet la recherche d'un ensemble approprié d'algorithmes d'indexation par rapport à une requête mots clés, un contexte d'exécution d'algorithmes d'indexation, des contraintes et des propriétés définies par l'utilisateur.

Notre prototype comprend donc trois grandes parties :

– Une interface graphique (une applet Java) qui comprend un formulaire permettant de créer des instances de notre modèle (voir section 3.1). Ces instances sont

encodées en XML et stockées dans une base de données Oracle Berkeley DB XML [7]. Le formulaire est accessible à l'adresse suivante : `http://www.irit.fr/PERSONNEL/SIG/laborie/LINDO/launch.html` (Figure 4) ;

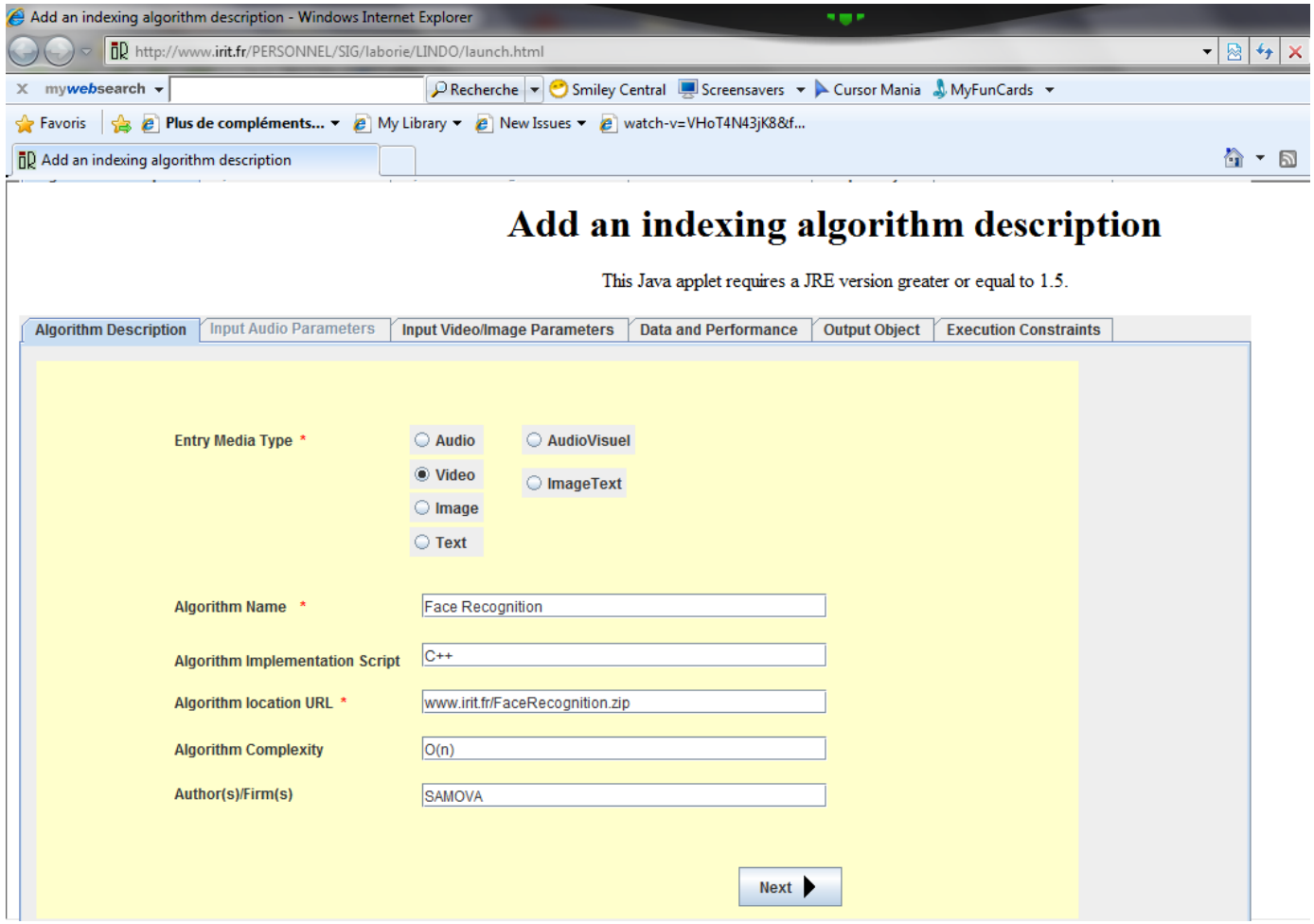


FIGURE 4: Le formulaire qui permet la saisie des descriptions d'algorithmes.

– Une interface qui permet de déclarer des chaînes d'application d'algorithmes relatives aux descriptions stockées dans la collection (Figure 5) ;

7. `http://www.oracle.com/technetwork/database/berkeleydb/overview/index-083851.html`

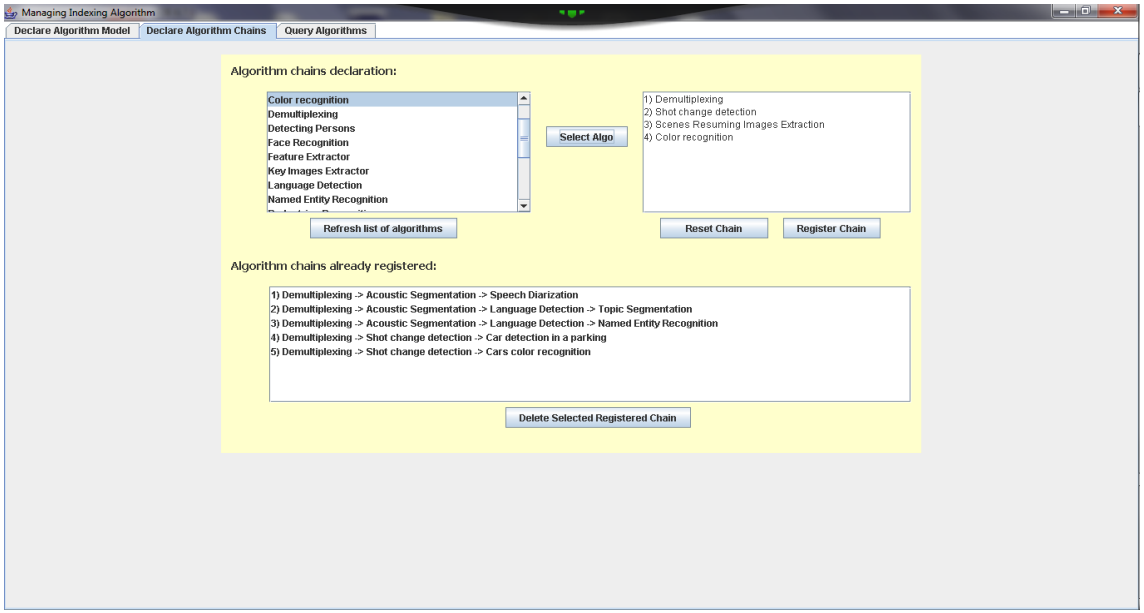


FIGURE 5: L'interface qui permet la définition des chaînes d'application d'algorithmes d'indexation.

– Une interface qui permet de spécifier une requête, d'identifier des contraintes et des propriétés, et sélectionner un contexte d'exécution (Figure 6).

De plus, ce travail s'est intégré dans le cadre du projet européen ITEA2 LINDO (Large Scale Distributed INDexation of multimedia Objects) qui propose une architecture distribuée et générique et qui permet l'indexation de contenus multimédias. Au lieu de déplacer des contenus volumineux vers un serveur central pour traitement, le projet propose d'indexer les contenus multimédias sur les sites distants et de déployer différents algorithmes d'indexation sur ces serveurs distants.

La procédure de décision que nous avons proposé dans la section précédente est utilisée dans le cadre du projet afin de réaliser une indexation implicite (i.e., à partir des besoins, des propriétés souhaitées et des contextes, déterminer automatiquement les algorithmes d'indexation à exécuter à l'arrivée d'un nouveau média) et une indexation explicite (i.e., si un serveur distant ne fournit pas de réponses notre proposition est également utilisée pour déterminer des algorithmes d'indexation à installer à la demande).

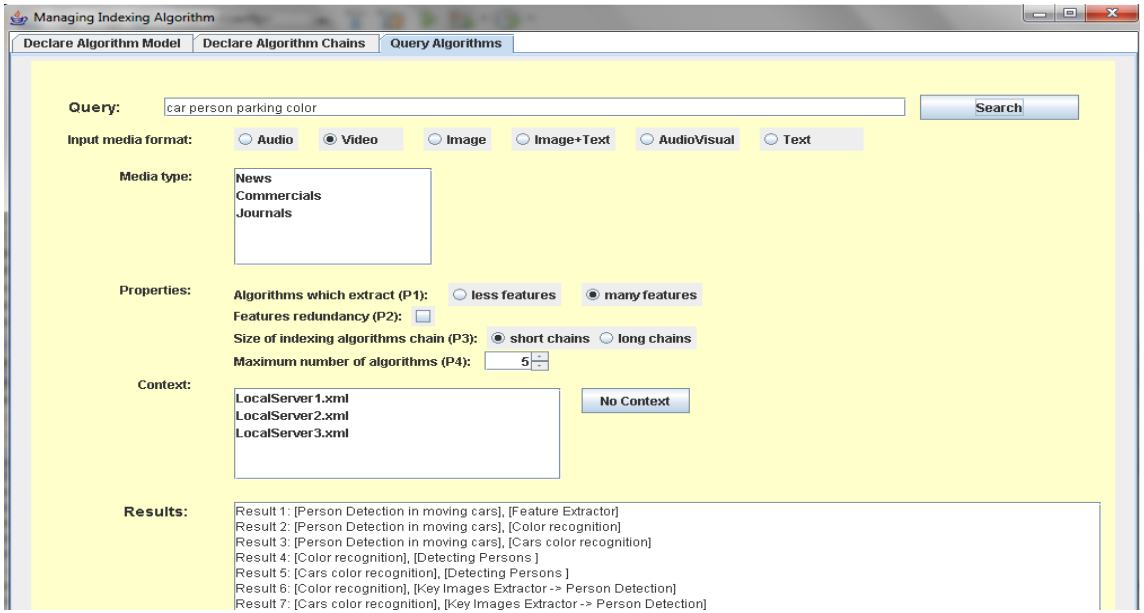


FIGURE 6: L'interface qui permet de spécifier une requête et d'instancier des contraintes et des propriétés.

## 5 Conclusions et perspectives

Nous avons proposé un modèle générique de description d'algorithmes d'indexation ainsi qu'un processus automatique de sélection d'algorithmes d'indexation qui satisfait des contextes donnés, des besoins et des préférences d'utilisateurs de systèmes d'informations. La méthode proposée est utile afin de mettre en place les processus d'indexation implicite et explicite dans le cadre de système d'indexation distribué en temps réel de contenus multimédias.

Nous envisageons comme perspectives, d'utiliser notre modèle de description d'algorithmes d'indexation pour d'autres types d'applications, comme par exemple engendrer à partir de notre modèle des descriptions de type WSMO [8] ou AWSDL [9]. Également, nous souhaiterions déterminer implicitement des chaînes d'applications d'algorithmes d'indexation par rapport aux entrées et aux sorties des algorithmes. En effet, actuellement l'utilisateur définit manuellement tous les chaînages possibles ce qui peut être très fastidieux s'il existe beaucoup d'algorithmes et de possibilités de combinaison.

Enfin, nous souhaiterions évaluer à plus large échelle notre prototype pour mesurer la pertinence des résultats produits, la rapidité de temps de réponse...

8. http://www.wsmo.org
9. http://www.w3.org/TR/sawsdl/

## Références


[1] Michael W. Berry and Malu Castellanos, editors. *Survey of Text Mining II*. Springer London, 2008.

[2] Sergey Brin and Lawrence Page. The anatomy of a large-scale hypertextual web search engine. Technical report, Computer Science Department, Stanford University, 1998.

[3] Bertrand Delezoide. *Modèles d'indexation multimedia pour la description automatique de films de cinéma*. PhD thesis, UNIVERSITÉ PARIS VI PIERRE ET MARIE CURIE, 2006.

[4] John P. Eakins. Towards intelligent image retrieval. *Pattern Recognition*, 35(1) :3–14, 2002.

[5] Myron Flickner, Harpreet Sawhney, Wayne Niblack, Jonathan Ashley, Qian Huang, Byron Dom, Monika Gorkani, Jim Hafner, Denis Lee, Dragutin Petkovic, David Steele, and Peter Yanker. Query by image and video content : The qbic system. *Computer*, 28 :23–32, 1995.

[6] T. Gevers and A. W. M. Smeulders. Pictoseek : Combining color and shape invariant features for image retrieval, 2009.

[7] G. Golovchinsky. *From information retrieval to hypertext and back again : the role of interaction in the information exploration interface*. PhD thesis, University of Toronto, 1997.

[8] Mislav Grgic, Kresimir Delac, and Mohammed Ghanbari, editors. *Recent Advances in Multimedia Signal Processing and Communications*, volume 231 of *Studies in Computational Intelligence*. Springer Berlin – Heidelberg, 2010.

[9] Bassem Haidar. *Services d'Indexation Multimedia Distribues*. PhD thesis, IRIT, 2005.

[10] Aboul-Ella Hassanien, Ajith Abraham, and Janusz Kacprzyk, editors. *Computational Intelligence in Multimedia Processing : Recent Advances*, volume 96 of *Studies in Computational Intelligence*. Springer Berlin – Heidelberg, 2008.

[11] Yoshihiko Hayashi, Katsutoshi Ohtsuki, Katsuji Bessho, Osamu Mizuno, Yoshihiro Matsuo, Shoichi Matsunaga, Minoru Hayashi, Takaaki Hasegawa, and Naruhiro Ikeda. Speech-based and video-supported indexing of multimedia broadcast news. In *SIGIR*, pages 441–442. ACM, 2003.

[12] Jacob Köhler, Stephan Philippi, Michael Specht, and Alexander Rüegg. Ontology based text indexing and querying for the semantic web. *Knowl.-Based Syst.*, 19(8) :744–754, 2006.

[13] Harald Kosch and Paul Maier. Content-based image retrieval systems – reviewing and benchmarking. In *Proc. of the 9th Workshop on Multimedia Metadata (WMM'09)*, pages 1–21, 2009.

[14] Harald Kosch and Paul Maier. Content-Based Image Retrieval Systems - Reviewing and Benchmarking . In *Proceedings of the 9th Workshop on Multimedia Metadata (WMM'09) held in conjunction with the 13th French Multimedia Conference on Compression and Representation of Audiovisual Signals (CORESA 2009)*, pages 1–21, 2009.

[15] P.Y. Lambolez, J.P. Queille, and C. Chrisment. Exrep : A generic rewriting tool for textual information extraction. In *Inginierie des Systemes d'Information 3*, pages 471–487, 1995.

[16] T. Leung and J. Malik. Representing and recognizing the visual appearance of materials using three-dimensional textons. *International Journal of Computer Vision (IJCV)*, 43(1) :29–44, 2001.

[17] Zhu Liu, Yao Wang, and Tsuhan Chen. Audio feature extraction and analysis for scene segmentation and classification. In *Journal of VLSI Signal Processing System*, pages 61–79, 1998.

[18] Christopher D. Manning and Hinrich Schütze. *Foundations of Statistical Natural Language Processing*. The MIT Press, 2002.

[19] Alessandro Micarelli, Filippo Sciarrone, and Mauro Marinilli. Web document modeling. In *Adaptive Web 2007*, pages 155–192, 2007.

[20] D. A. Reynolds and P. Torres-Carrasquillo. Approaches and applications of audio diarization. In *Proc. of IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP '05)*, volume 5, pages 953 – 956, 2005.

[21] Gerard Salton and Michael J. McGill. *Introduction to modern information retrieval*. McGraw-Hill, 1986.

[22] Badrul Sarwar, George Karypis, Joseph Konstan, and John Riedl. Incremental singular value decomposition algorithms for highly scalable recommender systems. In *Fifth International Conference on Computer and Information Science*, pages 27–28, 2002.

[23] E. Scheirer. *Music-Listening Systems*. PhD thesis, MIT Media Laboratory, 2000.

[24] E. Scheirer and M. Slaney. Construction and evaluation of a robust multifeature speech/music discriminator. In *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, 1997.

[25] Cees G.M. Snoek and Marcel Worring. Multimodal video indexing : A review of the state-of-the-art. *Multimedia Tools and Applications*, 25 :5 – 35, 2005.

[26] S. R. Subramanya and Abdou Youssef. Wavelet-based indexing of audio data in audio/multimedia databases. In *in Proceedings of MultiMedia Database Management Systems*, pages 46–53, 1998.

[27] Remco C. Veltkamp and Mirela Tanase. Content-based image retrieval systems : A survey. Technical report, 2000.

[28] Remco C. Veltkamp and Mirela Tanase. Content-based image retrieval systems : A survey. Technical Report UU-CS-2000-34, Department of Computing Science, Utrecht University, 2001. `http://www.aa-lab.cs.uu.nl/cbirsurvey/cbir-survey/`.

[29] Dongqing Zhang and Shih-Fu Chang. A generative-discriminative hybrid method for multi-view object detection. In *IEEE CVPR*, New York City, New York, June 2006.

[30] Tong Zhang and C.-C. Jay Kuo. Content-based classification and retrieval of audio. In *in SPIE's 43rd Annual Meeting - Conference on Advanced Signal Processing Algorithms, Architectures, and Implementations VIII*, pages 432–443, 1998.