



Document :	<i>Conquer, gérer l'hétérogénéité</i>
Sous-tâches :	3
Numéro des Livrables :	D3.2D3.3
Date	6/06/2012
Rédacteurs :	<i>Fabrice Jouanot (LIG), Marie-Christine Rousset(LIG), Guillaume Morin (LIG), Ammar Johar (LIG), Anis Benyelloul (LIG)</i>
Coordinateurs :	Fabrice Jouanot

SOMMAIRE

1.1	RAPPEL DES OBJECTIFS DE LA TACHE 3	4
1.2	STRUCTURE DU DOCUMENT	4
2.1	RAPPELS SUR CONQUER	7
2.2	CONQUER DANS L'INFRASTRUCTURE CONTINUUM	8
2.2.1	PRESENTATION GENERALE	8
2.2.2	INTERACTION AVEC LA BDC	9
2.2.3	ILLUSTRATION DE CONQUER SUR LE SCENARIO INDUSTRIEL	9
2.2.3.1	Exemple de moment "Dans la voiture"	9
2.2.3.2	Exemple de moment "Recherche 2 ^{ème} vanne"	12
2.2.3.3	Exemple de moment "Gestion de la clé de vanne"	15
3.1	INTRODUCTION	17
3.2	LES METHODES DE CLUSTERING	17
3.2.1	MEURES DE SIMILARITE ET DONNEES	18
3.2.2	ALGORITHMES	19
3.2.2.1	Clustering hiérarchique	19
3.2.2.2	L'algorithme ROCK	21
3.2.3	DIFFICULTE DE L'EVALUATION	23
3.2.3.1	La qualité interne	23
3.2.3.2	La qualité externe	24
3.2.4	CONCLUSION	25
3.3	UNE METHODE DE CLUSTERING HIERARCHIQUE POUR LES DONNEES DESCRIPTIVES	25
3.3.1	DESCRIPTION DES DONNEES D'ENTREES	26
3.3.2	REPRESENTATION, TRANSFORMATION ET ENRICHISSEMENTS DES DONNEES	27
	<i>Reconstruction des liens entre les documents</i>	28
3.3.3	FONCTIONS DE SIMILARITE	30
3.3.3.1	Jaccard	30
3.3.3.2	Cosinus et Tfidf	31
3.3.3.3	SoftJaccard	32
3.3.4	L'ALGORITHME AVERAGE-LINK	34
3.4	EVALUATIONS ET INTERPRETATIONS	35
3.4.1	VALIDATION D'UNE HIERARCHIE	35
	<i>Evaluation de la qualité intra/extra cluster</i>	35
	<i>Evaluation externe avec jeu de données artificielles</i>	36
3.4.2	INTERPRETATION	37
	<i>Résultats de la qualité interne Q_1 et Q_2</i>	37
	<i>Résultats de la qualité externe Q_3</i>	38
3.5	EVALUATIONS ET INTERPRETATIONS DU CLUSTERING HIERARCHIQUE	39
3.6	PROBLEME DU NOMMAGE DES CLUSTERS	39
3.6.1	METHODES DE LABELISATION D'UN CLUSTER HIERARCHIQUE	40

3.6.1.1	Méthodologie générale	40
3.6.1.2	Approche Jensen-Shannon Divergence	41
3.6.1.3	Approche basée sur une variante de TF-IDF	43
3.6.2	EXPERIMENTATIONS ET LIMITES DES APPROCHES	ERREUR ! SIGNET NON DEFINI.
3.7	CONCLUSION.....	45
4.1	METHODE DE RELAXATION DE REQUETES SPARQL.....	ERREUR ! SIGNET NON DEFINI.
4.2	APPLICATION DE LA RELAXATION A CONQUER	ERREUR ! SIGNET NON DEFINI.
4.3	EXPERIMENTATIONS ET VALIDATION.....	ERREUR ! SIGNET NON DEFINI.
4.4	CONCLUSION.....	ERREUR ! SIGNET NON DEFINI.
	<i>Contributions.....</i>	<i>46</i>
	<i>Perspectives.....</i>	<i>46</i>

1 Introduction

1.1 Rappel des objectifs de la tâche 3

Cette tâche a pour objectif de prendre en compte l'hétérogénéité de la description du contexte ainsi que des composants et de leurs fonctionnalités pour le déploiement d'applications interactives dans lesquelles les dispositifs utilisables se découvrent, se connectent et interagissent à la volée et de façon décentralisée.

Dans ce projet, on distingue l'hétérogénéité de l'interopérabilité que l'on suppose acquise grâce à l'utilisation de formats et de protocoles permettant de faire communiquer et interagir des dispositifs physiques ou des composants logiciels.

Nous voulons permettre une description à haut niveau d'abstraction du contexte, des composants et de leurs fonctionnalités :

- suffisamment compacte pour que chaque dispositif puisse stocker sa description et télécharger les descriptions d'autres dispositifs, ainsi que des informations contextuelles,
- suffisamment riche pour qu'on puisse raisonner sur ces informations contextuelles et fonctionnelles afin par exemple d'inférer des propriétés d'alignement en vue de pouvoir qualifier différents assemblages de composants, en fonction de la demande de l'utilisateur et/ou du contexte.

Cette tâche s'organise en 3 sous-tâches:

La tâche sera organisée autour des quatre sous-tâches suivantes :

- T3.1 : la définition d'un langage logique de description contextuelle de composants et de services qui est présenté dans le livrable 3.1
- T3.2 : la découverte d'assemblages et d'alignements et T3.3 : le raisonnement sur les propriétés et la continuité des services qui sont présentés dans ce livrable T3.3
- T3.4 : la réalisation d'un démonstrateur

1.2 Structure du document

Ce document est structuré comme suit :

- La section 2 replace la Base de connaissance dans le contexte du projet Continuum en évoquant la problématique requérant une ontologie pour modéliser l'espace intelligent, l'architecture de l'infrastructure Continuum et une illustration de l'utilisation de la base de connaissance dans le cadre du scénario industriel.
- La section 3 présente une méthode de constructions semi-automatique d'une ontologie comme bootstrap de base de connaissance. La définition d'une ontologie est une étape particulièrement critique que se propose de résoudre notre étude.
- La conclusion fait la synthèse sur résultats obtenus et évoquent des pistes pour aller plus loin dans l'usage d'outils sémantique pour résoudre les problèmes de continuité de service en informatique ambiante.

2 Rappels et problématiques

L'augmentation des supports mobiles et des dispositifs disséminés dans notre environnement change considérablement l'approche dans la gestion des ressources logicielles. Ce type d'environnement est souvent identifié dans la littérature comme l'informatique ubiquitaire ou pervasive. A l'usage, ces environnements nécessitent de résoudre les problématiques de déploiement, de sécurité, d'hétérogénéité, de découverte ou de continuité de service et un nombre conséquent de propositions dans la littérature démontre l'engouement pour ce type d'architecture. UPnP (Universal Plug and Play) est une solution s'alignant sur l'approche plate-forme de service, visant principalement les infrastructures de petites tailles (bureaux, petites entreprises, etc.). Il propose à travers une spécification de définir comment les dispositifs se connectent et interagissent entre eux.

Des travaux récents [BJR] ont proposé d'organiser les éléments UPnP participant au contexte sous une hiérarchie (taxonomie) de classes, dans le but d'utiliser les standards du Web Sémantique [ARR+11]. L'objectif est d'avoir une manipulation déclarative et facilement extensible de la connaissance permettant le raisonnement sur ces éléments. Ce modèle, décrit avec le langage RDFS, permet d'effectuer des requêtes à différents niveaux d'abstraction de l'arborescence.

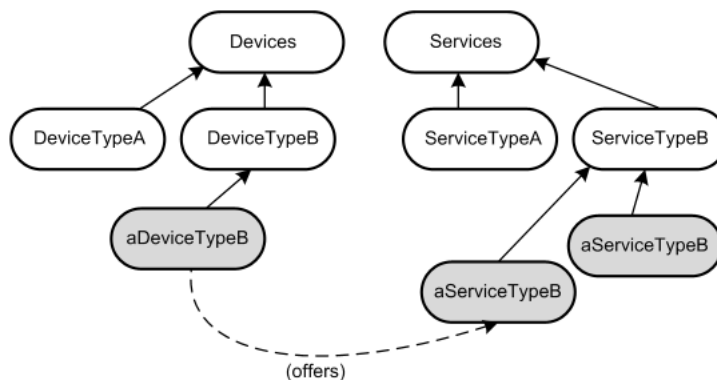


Figure 1: Exemple simplifié d'une taxonomie de classes

La Figure 1 représente une vision simplifiée d'un découpage des éléments d'une architecture cible (ici inspiré d' UPnP) par un ensemble de types regroupés sur deux classes, les devices et les services ainsi que les instances {aDeviceType, aServiceType}. La figure 1 montre aussi comment il est possible de mettre en relation des classes par des liens sémantiques qui s'expriment sous la forme de prédicats en logique du premier ordre. Dans la figure 1 nous définissons que aDeviceTypeB offre un service aServiceTypeB soit le fait suivant :

offers(aDeviceTypeB, aServiceTypeB)

L'objectif de ce document est d'étudier la manière de construire automatiquement une telle ontologie et d'exploiter les caractéristiques inhérentes à cette construction pour améliorer l'interrogation de la base connaissance.

2.1 Rappels sur Conquer

Nous proposons une approche déclarative pour la modélisation des données du contexte basée sur les standards du web sémantique (RDFS pour la représentation, et SPARQL pour l'expression de requêtes).

L'approche se base sur une taxonomie de classes et de propriétés qui vont permettre de catégoriser les données du contexte, et d'exprimer des requêtes sur ces données.

Après un bref résumé de l'approche, les sections suivantes illustrent quelques moments clés du scénario industriel de Continuum et du scénario prospectif.

Les informations sur le contexte sont stockées en interne sous forme de *triplets* RDFS. Chaque triplet est composé d'un sujet, prédicat et d'un objet. Dans la notation graphique, le sujet et l'objet d'un triplet correspondent au nœud du graphe, et les prédicats aux arrêtes. La figure 1 montre un exemple de données RDFS représentées sous forme graphique ainsi que la représentation en triplets correspondante.

Indépendamment de la notation choisie, les triplets ont une sémantique formelle décrite en logique du premier ordre: Un triplet de la forme (*sujet*, *prédicat*, *objet*) est interprété en logique du premier ordre comme un fait *predicat(sujet,objet)* : *predicat* étant prédicat et *sujet* et *objet* des constante dans le langage du premier ordre.

RDFS définit certains prédicats standards impliquant une sémantique particulière. Par exemple, un triplet de la forme (*aScreen*, *rdf:type*, *Display*) est interprété en logique du premier ordre par le fait *Display(aScreen)*, et le triplet (*Display*, *rdfs:subclassof*, *Service*) est interprété par le fait que toutes les instances de *Display* sont des instances de *Service*:

$$\sum x Display(x) \Rightarrow Service(x)$$

On peut donc en déduire le triplet (*aScreen*, *rdf:type*, *Display*).

SPARQL est le langage de requête recommandé par le W3C pour l'interrogation de base de données RDF. Il existe différentes syntaxes pour exprimer des requêtes SPARQL. Par exemple la requête suivante recherche les services qu'offre l'instance *aTv*:

```
SELECT ?s WHERE {
    ?s rdf:type Service
    aTv offers ?s
}
```

L'expression correspondante en logique du premier ordre peut s'énoncer ainsi:

$$\exists x Service(s) \wedge Offers(aTv, s)$$

La plupart des implémentations de SPARQL existantes ne fournissent pas de support pour la sémantique RDFS évoquée plus haut. Pour pallier à ce problème et permettre l'utilisation d'implémentations SPARQL existantes, une phase de pré-traitement, appelée saturation est nécessaire. L'étape de

saturation consiste à parcourir la base de triplets RDFS en appliquant les différentes règles de déductions, et en ajoutant les triplets résultants à la base.

2.2 Conquer dans l'infrastructure CONTINUUM

Conquer se trouve au cœur de l'infrastructure Continuum et différents niveau d'interaction avec la base de connaissance existent. Les sections suivantes rappellent l'architecture générale de Continuum et y replace la base de connaissance Conquer.

2.2.1 Présentation générale

La figure 1 donne une représentation architecturale globale de l'infrastructure CONTINUUM :

- Les dispositifs communicants de l'environnement ambiant sont encapsulés dans des services UPnP.
- L'application, sous forme d'un tissage de composants, est encapsulé dans un service WCOMP. Ce service permet de faire abstraction des services et dispositifs utilisés pour réaliser la tâche induite par l'application à un moment donné.
- L'aspect dynamique de l'architecture est piloté par le composant AA Designer qui est en charge de reconfigurer le service WCOMP représentant l'application. Ce service est complété par un UPnP designer qui sert essentiellement de proxy entre les dispositifs et leur représentation en service UPnP.
- Une base des composants regroupe des composants décrit dans via une onotlogie de référence, et qui sont disponibles pour une utilisation par le AA Designer.

Trois composants clés complètent l'architecture de Continuum : la Base de Connaissances (BdC), le Gestionnaire de Contexte (GC) et les méta-IHM tactique-GC, tactique-BdC et réflexe:

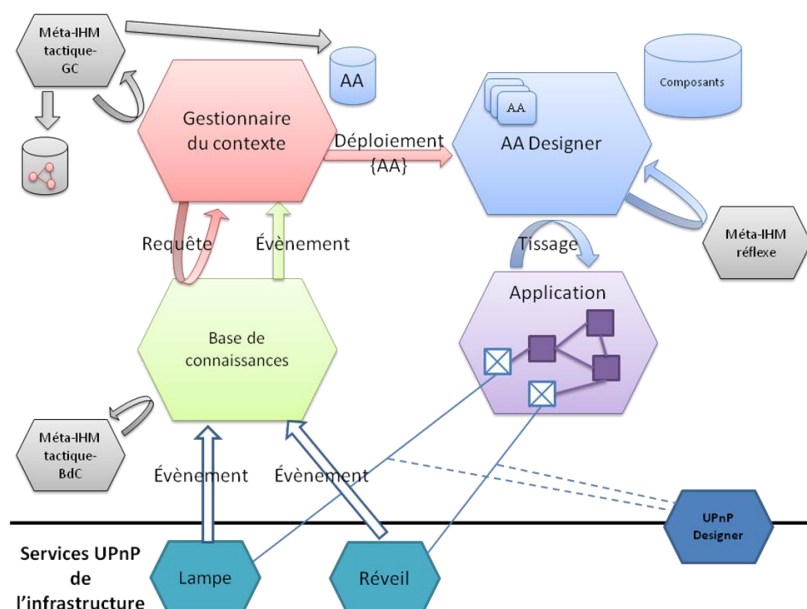


Figure 1. Architecture Continuum

la Base de Connaissances (BdC), le Gestionnaire de Contexte (GC) et les méta-IHM tactique-GC, tactique-BdC et réflexe: La Base de Connaissances CONTINUUM (BdC)

- La Base de Connaissances (BdC) maintient une connaissance de l'environnement. Elle repose sur les standards du Web Sémantique et offre un modèle basé sur RDFS et une extension de SPARQL pour modéliser et interroger l'espace intelligent.
- Le Gestionnaire de Contexte CONTINUUM (GC) s'appuie sur les concepts introduits par [Rey 05 ; Coutaz 05] : un contexte est un graphe de situations qui partagent les mêmes ensembles de prédicats et d'entités. Dans CONTINUUM, ces entités sont les classes et instances de classes modélisées dans la BdC (Personne, Dispositif, etc.) et les prédicats sont des prédicats exprimables dans les termes de la BdC.
- Les Méta_IHM sont présentes au niveau réflexe (méta_IHM réflexe), au niveau tactique pour mettre au point les règles pilotant l'adaptation (KISS), et au niveau stratégique en utilisant des techniques d'apprentissage (non implémenté actuellement).

2.2.2 Interaction avec la BdC

Comme illustrée sur la figure 1, la BdC reçoit des événements UPnP en provenance des dispositifs UPnP de l'environnement:

- La base de faits de la BdC est mise à jour en fonction des fluctuations de l'environnement (apparition/disparition de dispositifs, mise à jour des propriétés de dispositifs)
- Transfert de ces événements au gestionnaire de contexte. Le GC peut recevoir soit les événements bas niveau non modifiés, soit des événements haut niveau après traitement par la BdC.

Le GC peut aussi interroger la BdC (requête SPARQL) sur réception de notification de changement d'état en provenance de la BdC notamment afin de compléter ses connaissances sur une situation. À chaque situation correspond des AAs mémorisés dans une base d'AAs qui seront déployés par le GC. Cette base d'AA évolue via KISS dans son rôle de méta-IHM tactique. Cette *méta-IHM* doit permettre à l'utilisateur final de programmer des AAs pour des situations prévisibles. La construction de l'interface d'interaction avec l'utilisateur est définie dynamiquement par interrogation au contenu de la BdC.

2.2.3 Illustration de Conquer sur le scénario industriel

Les sections suivantes présentent l'instanciation de l'approche sur le scénario industriel de Continuum. Le scénario décrit une intervention d'un fontainier pour effectuer un arrêt d'eau, et détaille les différents moments de l'intervention.

2.2.3.1 Exemple de moment "Dans la voiture"

Maurice finit la préparation de l'arrêt d'eau depuis chez lui. Les informations résultantes de la préparation suivent Maurice alors qu'il entre dans son véhicule pour se rendre sur le lieu de l'intervention.

Les dispositifs disponibles dans le véhicule sont:

- Vision tête haute (HeadUpDisplay)
- PDA
- L'ordinateur embarqué du véhicule (EmbeddedComputer)
- Le GPS du véhicule (CarGPS) (PortableGPS)

La figure 2 montre la modélisation des dispositifs présents dans le véhicule.

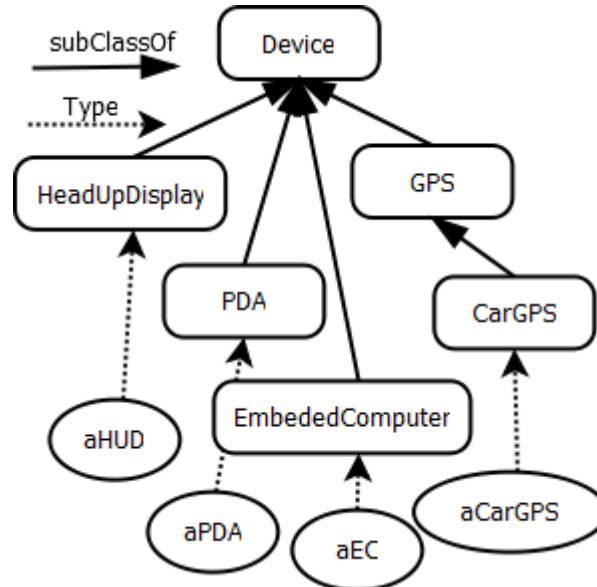


Figure 2. Moment 1 (Voiture): Les dispositifs disponibles

La figure 3 montre les différents services disponibles dans le véhicule: On voit quatre catégories de services de premier niveau:

- SIG: Services d'accès au Service d'informations Géographique;
- Display: Services d'affichage;
- Command: Services de contrôle;
- GPSLocalization: Services de localisation GPS.

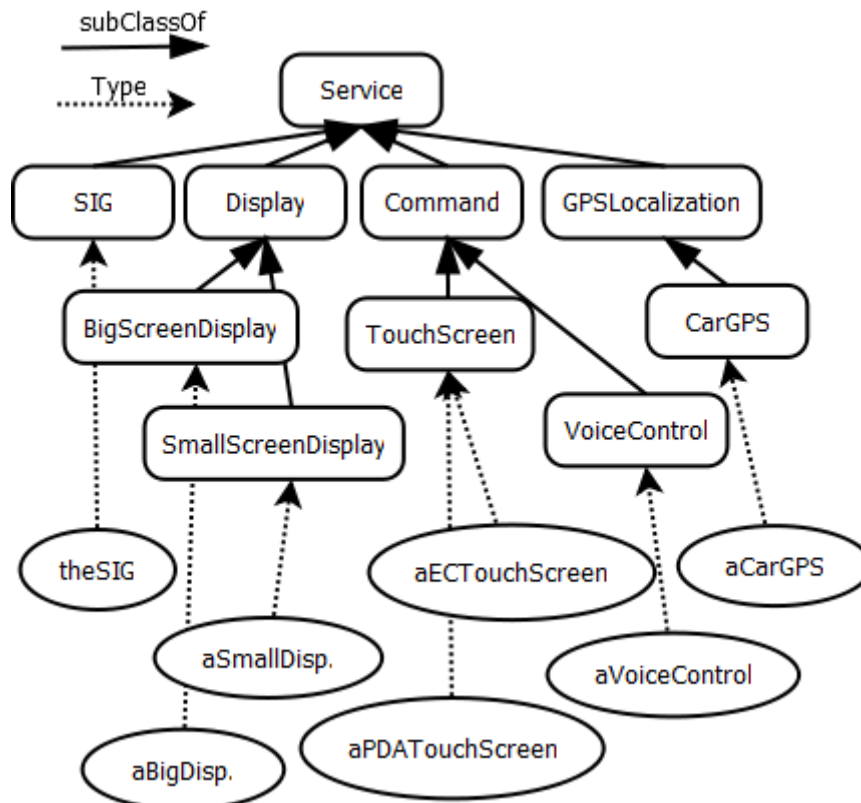


Figure 3. Moment 1 (Voiture): Services Disponibles

Les catégories de premier niveau se déclinent à leur tour en classes plus spécifiques: `BigScreenDisplay` et `SmallScreenDisplay` pour les services d'affichage sur grand et petit (i.e portable) écran, respectivement. La classe `Command` se déclinent à son tour en `TouchScreen` pour les services de commande par écran tactile et `VoiceControl` pour les services de commande vocale.

Chaque dispositif va donc proposer un service d'un des types cités, ce qui est modélisé par une relation `Offers` entre les instances de dispositifs et les instances de services:

- La vision tête haute (type `HeadUpDisplay`) offre un service d'affichage sur petit écran (type `SmallScreenDisplay`);
- Le PDA permet d'accéder au SIG (i.e offre un service du type `SIG`), permet de contrôler le système grâce à son écran tactile (i.e offre un service de type `TouchScreen`), et permet l'affichage sur petit écran (service du type `SmallScreenDisplay`);
- L'ordinateur de bord offre un service d'affichage sur grand écran (type `BigScreenDisplay`) ainsi qu'un service de commande vocale (type `VoiceControl`);
- Le GPS du véhicule fournit un service de localisation (type `GPSLocalization`).

Pendant son trajet jusqu'au lieu d'intervention, le système doit accomplir deux tâches:

- Guider Maurice jusqu'à destination: Tâche du type `Arrival`
- Continuer la préparation de l'intervention dans le véhicule (si nécessaire): tâche du type `Preparation`.

Pour la première tâche, un service de type `GPSLocalization` est nécessaire. La requête correspondante dans ce cas va porter sur les dispositifs offrant des services de type `GPSLocalization`. La formalisation en SPARQL peut s'énoncer ainsi:

```
SELECT ?device WHERE {
    ?device offers ?service
    ?service type GPSLocalization
}
```

Il n'y a qu'un seul résultat possible pour cette requête c'est le GPS de la voiture qui est le seul dispositif à offrir un service du type `GPSLocalization`.

Pour la tâche de préparation: un service d'accès au SIG, un service de contrôle (type `Command`) et un service d'affichage (type `Display`) pour afficher les informations sur le bief. Les requêtes correspondantes peuvent s'énoncer ainsi:

<pre>SELECT ?device, WHERE { ?device offers ?service ?service type SIG }</pre> (a)	<pre>SELECT ?device, WHERE { ?device offers ?service ?service type Command }</pre> (b)	<pre>SELECT ?device, WHERE { ?device offers ?service ?service type Display }</pre> (c)
---	---	---

Pour la requête (a) le seul dispositif offrant accès au SIG est le PDA.

La requête (b) donne les résultats suivants:

- Le PDA, qui offre un service de type `TouchScreen`, sous-type de `Command`;
- L'ordinateur embarqué, qui offre aussi un service du type `TouchScreen` ainsi qu'un service `VoiceControl` (i.e il permet un contrôle vocal), tout deux sous-type de `Command`.

La requête (c) donne les résultats suivants:

- Le PDA, qui offre un service de type SmallScreenDisplay, sous-type de Display;
- L'ordinateur embarqué, qui offre un service de type BigScreenDisplay, sous-type de Display;
- La vision tête haute de la voiture, qui offre un service de type SmallScreenDisplay, sous-type de Display.

Les 4 montre une situation où la tâche de préparation est réalisée par le service de type SIG offert par le PDA, ainsi que les services de type VoiceControl et BigScreenDisplay de l'ordinateur de Bord.

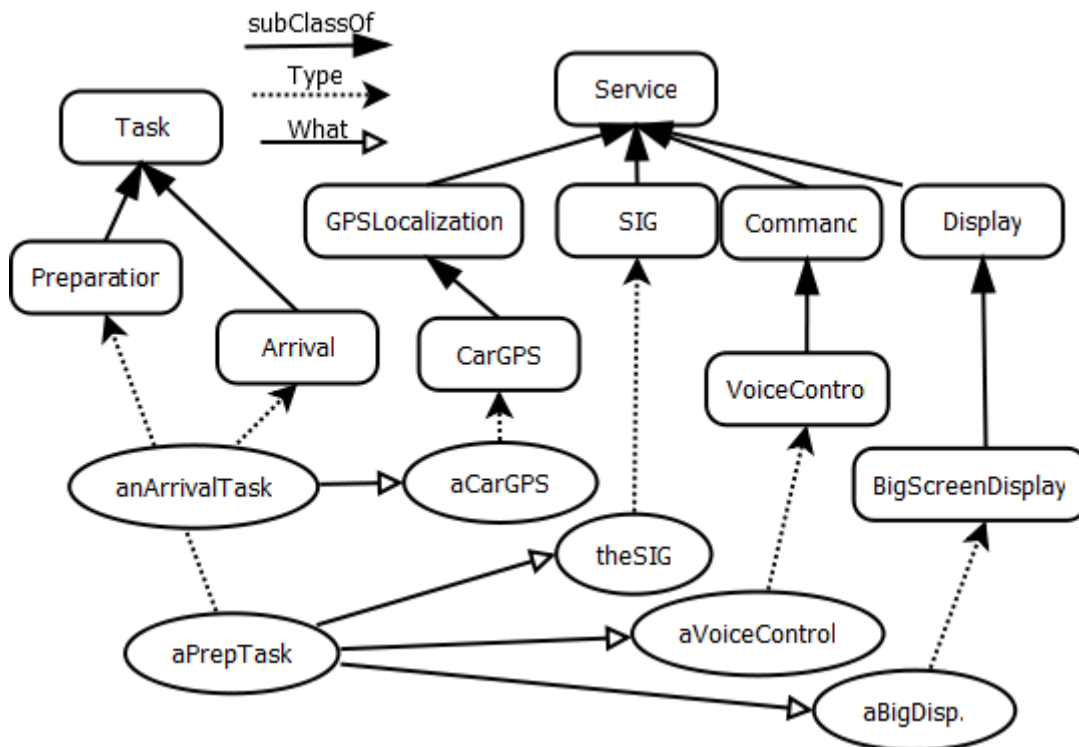


Figure 4. Moment 1 (Voiture): Tâches

2.2.3.2 Exemple de moment "Recherche 2^{ème} vanne"

Maurice arrive sur les lieux de l'intervention et sort de son véhicule.

Les dispositifs qu'il emporte avec lui sont (voir figure 5):

- Son PDA,
- Son casque/visière (HeadSet)
- Son GPS portable (PortableGPS)
- Sa Poêle

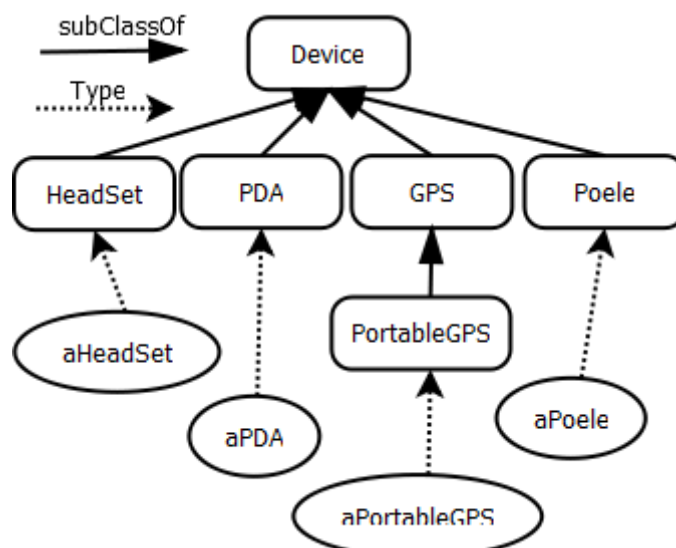


Figure 5. Moment 2 (2ieme Vanne): Dispositifs

On retrouve la plupart des types de services disponible pendant le moment 1 (c.f. Figure 4). Les classes de services de premier niveau restent les mêmes (SIG, Display, Command, GPSLocalization) avec l'ajout d'une classe pour les services de détection de bouche à clef (Detection).

On retrouve les classes SmallScreenDisplay, TouchScreen, VoiceControl, pour les services d'affichage sur petit écran, la commande par écran tactile et la commande vocale.

Le PDA fournit toujours un service d'affichage sur petit écran (type SmallScreenDisplay), un service de commande par écran tactile (type TouchScreen), ainsi qu'un accès au SIG. Le casque visière (nouveau dispositif) offre un service d'affichage sur petit écran (SmallScreenDisplay) et un service de commande vocale (type VoiceControl).

Une fois sur le terrain la tâche consiste à repérer la vanne, à fermer (tâche du type Location) et à continuer la tâche de préparation d'intervention commencée chez lui: il demande des informations supplémentaires aux SIG, demande des photos cotée à fouille ouverte du site, etc.

La tâche de repérage nécessite un service du type GPSLocalization (pour le guider jusqu'à la bouche à clefs), un service d'affichage (type Display) pour afficher les informations concernant son avancement, et un service du type Détection pour pouvoir localiser la vanne une fois sur place. Les requêtes correspondantes peuvent s'énoncer ainsi:

<pre> SELECT ?device WHERE { ?device offers ?service ?service type GPSLocalization } </pre> (a)	<pre> SELECT ?device WHERE { ?device offers ?service ?service type Display } </pre> (b)	<pre> SELECT ?device WHERE { ?device offers ?service ?service type Detection } </pre> (c)
--	--	--

La requête (a) renvoi pour résultat le dispositif GPS portable que Maurice emmène avec lui sur le terrain.

La requête (b) retourne comme résultat le PDA et le casque, qui offrent tous deux un service de type SmallScreenDisplay, sous-type de Display.

La requête (c) renvoi pour résultat le détecteur de métaux que Maurice utilise pour trouver les bouches a clefs.

La figure 7 montre une situation ou la tâche de repérage est réalisée en exploitant le service de détection fournit par la poêle, le service de repérage GPS fournit par le GPS portable, et le service d'affichage de la visière.

La tâche de préparation nécessite (comme vu précédemment) un service d'accès au SIG, un service de type Command, et un service d'affichage (type Display). La requête est identique a celle du moment 1. Dans le moment 2, ces services sont fourni respectivement par le PDA, le Casque (pour le service de Command et le service Display).

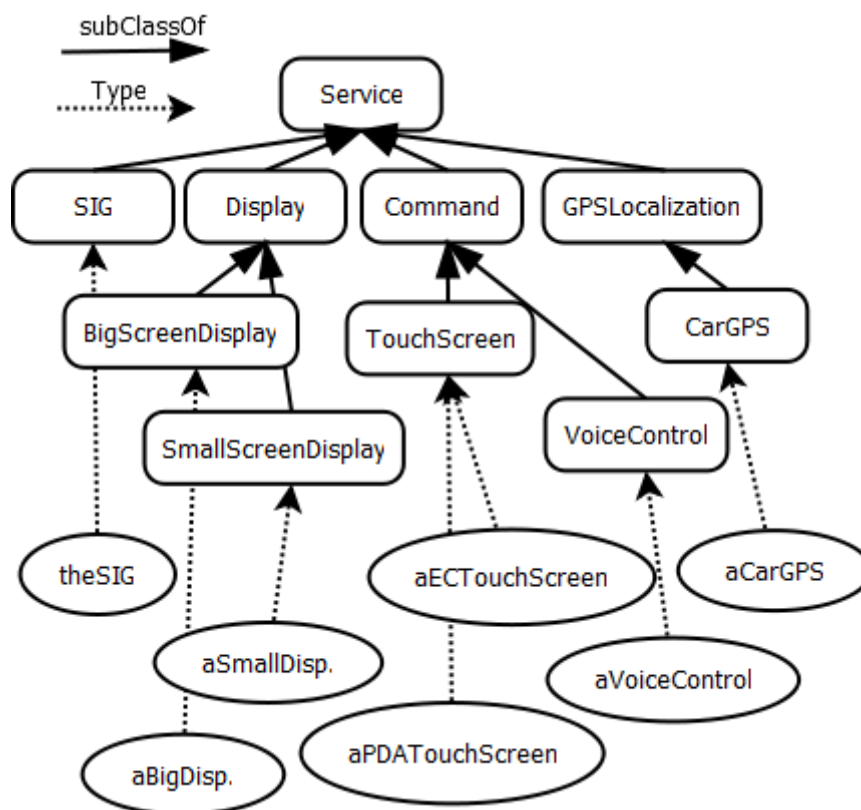


Figure 6.Moment 2 (2ieme Vanne): Services

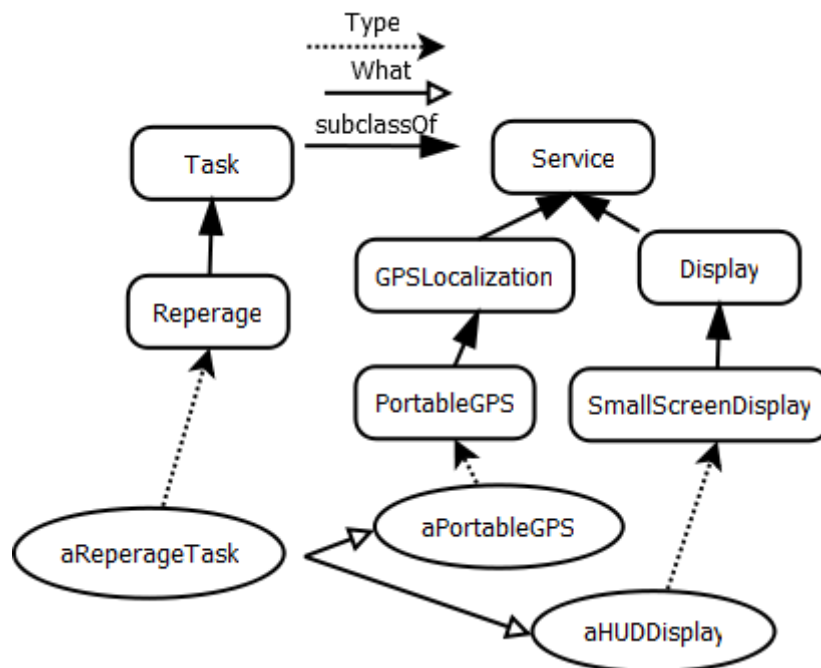


Figure 7. Moment 2 (2ieme Vanne): Tâches

2.2.3.3 Exemple de moment "Gestion de la clé de vanne"

`Maurice abaisse la visière de son casque => l'affichage de l'écran du PDA est maintenant sur la visière du casque". Cela se traduit par la substitution du service d'affichage fournit par le PDA, par le service d'affichage de la visière (voir figure 8).

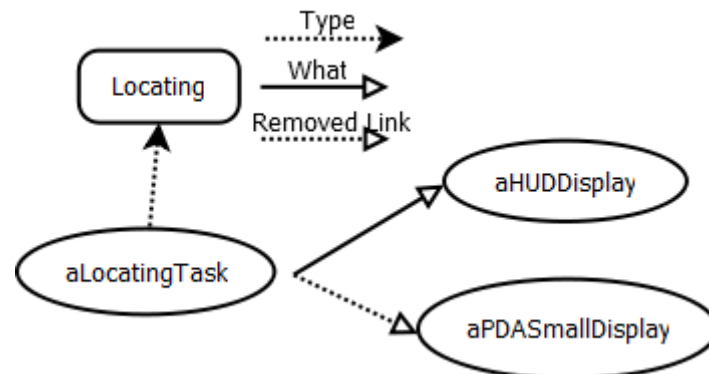


Figure 8. Moment 2 (2ieme Vanne): L'affichage de l'écran du PDA passe sur la visière du casque

``Maurice a les mains occupées et il veut passer en mode commande vocale pour mettre à jour les informations sur la nouvelle position de vanne dans le SIG". Cela se traduit la substitution du service de commande fournit par le PDA (de type TouchScreen) par le service de commande du casque/visière (du type VoiceControl) (voir figure 9).

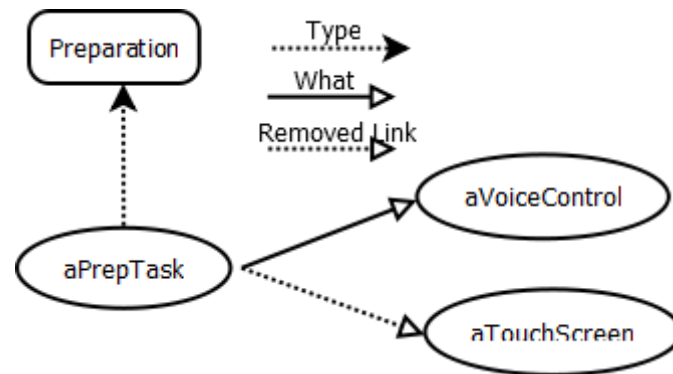


Figure 9: Moment 2 (2ieme Vanne): Passage de l'écran tactile du PDA a la commande vocale pour le contrôle

3 Construction automatique d'ontologie pour l'informatique ambiante

3.1 Introduction

Une *taxonomie*, avec des prédicats associés, forme une *ontologie légère*. La construction d'ontologie *même légère*, est un problème difficile qui a donné lieu à de nombreux travaux avec deux grandes familles d'approche. La première regroupe les travaux sur les méthodologies pour modéliser la connaissance afin de guider la construction manuelle, produite par un expert, d'ontologie souvent riche et complexe. La seconde visant la construction automatique d'ontologie.

A travers cette étude, nous souhaitons expérimenter *l'automatisation* de la construction d'une telle hiérarchie en utilisant une méthode de *clustering hiérarchique*. Pour produire cette taxonomie, nous utilisons un ensemble de documents XML fourni par le *Forum UPnP*. Chaque document représente un *patron* à partir duquel il est possible de définir des *instances*. Nous considérons ici ces patrons comme des types de l'architecture UPnP. Ainsi, la méthode de construction de la hiérarchie devra être assez générique pour organiser des regroupements à partir de données descriptives proches du domaine, en d'autres termes, de pouvoir généraliser l'utilisation aux plates-formes de services avec le minimum de paramétrage. Sans la connaissance du domaine, il est important que la solution puisse fournir *des critères de validation*, qu'ils soient apportés par une mesure objective ou avec une mesure subjective adaptée à un référentiel.

Ce chapitre est organisé en quatre chapitres :

- Un état de l'art des différentes techniques de clustering.

Dans cette partie, nous examinons les particularités et les composants du clustering hiérarchique en illustrant par quelques exemples qui font référence dans le domaine.

- La présentation de notre approche

Dans cette partie, nous décrivons les étapes de construction d'une taxonomie à l'aide d'une méthode de clustering hiérarchique. Nous précisons les caractéristiques de l'algorithme utilisé et nous abordons les mesures de similarité que nous avons expérimentés

- Une partie interprétation et validation où nous interprétons les résultats obtenus suivant certaines mesures de qualité.

- Une partie traitant du nommage des clusters afin de construire une taxonomie utilisable

- Une partie d'évaluation de la méthode dans un cadre général et dans celui plus spécifique de Continuum

3.2 Les méthodes de clustering

L'accroissement du volume de données dans notre environnement et leurs rôles de plus en plus significatifs, nous poussent à utiliser davantage les méthodes de traitements automatisés. Cette partie fournit un état de l'art d'une branche spécifique, souvent mentionnée sous le nom de *clustering* qui s'attache à étudier les problématiques liées au classement de ces données. Ici, nous définissons par *classement* le moyen de lier une donnée à un ou plusieurs groupes naturels (appelés

parfois cluster) dont les membres partagent un ensemble de propriétés. Il est fait souvent mention de *similarités entre les membres d'un même cluster*. L'utilisation du *clustering* est vaste et varie selon le besoin. Mais l'objectif général reste identique : révéler par le *regroupement ou la segmentation* des données une connaissance qui peut être exploitée ultérieurement.

L'utilisation d'une méthode de clustering n'est pas définie comme une approche monolithique mais plutôt comme une série d'étapes. Ainsi, [JD88, And04] propose une liste définissant ces étapes. Nous proposons une liste similaire mais réduite et plus adaptée à la taille de notre problème.

- Extraction : extraire les données et en distinguer des objets spécifiques (individus). C'est ici que les données sont préparées, nettoyées.
- Représentation des données : obtenir une représentation des données adaptée à l'algorithme, choisir une fonction de similarité suivant la nature de cette représentation, appréhender les différents critères sur ces données.
- Stratégie de classement : choisir l'algorithme suivant les exigences (performance, représentation des résultats, type de données) et fixer les éventuels paramètres.
- Validation : estimer la qualité de la construction produite (par un expert ou en utilisant une métrique adaptée).
- Interprétation et utilisation : juger si la classification répond bien aux besoins exprimés en utilisant la classification avec les données réelles si cela n'a pas déjà été fait auparavant.

3.2.1 Meures de similarité et données

La plupart des algorithmes de clustering utilise une fonction de similarité pour distinguer deux individus. D'une manière générale une similarité est une mesure normalisée (*i.e.* $\in [0,1]$) employée sur une paire individus qui respecte les expressions suivantes, avec $Sim(x_i, x_j)$ une mesure de similarité, $Sim : A \times A \rightarrow [0,1]$ alors :

$$(\text{maximalité}) \forall a \in A, Sim(a, a) = 1$$

$$(\text{symétrie}) \forall a, a' \in A Sim(a', a) = Sim(a, a');$$

La littérature fait aussi souvent référence à des distances comme évaluation de la similarité entre deux individus. Ces fonctions sont analogues, et donnent la similarité par $1 - Sim$. Deux individus complètement similaires ont une distance de 0. Ces mesures sont applicables sur plusieurs types de représentation des données. Pour un ensemble de données qualitatives on utilise souvent des mesures comme Jaccard, Dice ou Tversky. Par exemple les définitions suivantes pour deux ensembles x_i et x_j

l'indice de Jaccard :

$$Jaccard(x_i, x_j) = \frac{|x_i \cap x_j|}{|x_i \cup x_j|}$$

Dice :

$$Dice(x_i, x_j) = \frac{2|x_i \cap x_j|}{|x_i| + |x_j|}$$

Alors que pour les données quantitatives on utilise plutôt Cosine. Soit deux vecteurs de dimension k , $\vec{x}_i = (x_{i1}, \dots, x_{ik})$ et $\vec{x}_j = (x_{j1}, \dots, x_{jk})$ alors :

$$\text{Cosine}(\vec{x}_i, \vec{x}_j) = \frac{\vec{x}_i \cdot \vec{x}_j}{\|\vec{x}_i\| \cdot \|\vec{x}_j\|}$$

De nombreuses fonctions de similarité existent et sont largement étudiées [CRF03, XI05] dans la littérature. Dans la suite de cette étude nous détaillons plus précisément Jaccard et Cosine.

3.2.2 Algorithmes

D'une manière générale, il existe deux grandes catégories de méthodes de clustering, le clustering par partition et le clustering hiérarchique :

- Le clustering par partition produit une *partition* d'un ensemble d'individus de départ où chaque individu appartient à un cluster unique.
- Le clustering hiérarchique produit une *hiérarchie* de clusters liés par une relation de généralisation (si nous remontons progressivement dans la hiérarchie), spécialisation (si nous descendons dans la hiérarchie).

De même, ces méthodes peuvent être subdivisées par l'utilisation de caractéristiques particulières. Nous pouvons citer les approches basées sur les grilles, sur la densité, les graphes, les statistiques, etc. La littérature fournit déjà un ensemble de catalogues et de comparatifs [Can07, XI05] important sur les méthodes de clustering. C'est pourquoi nous proposons plutôt de présenter les approches permettant de produire des *hiérarchies*.

3.2.2.1 Clustering hiérarchique

Les approches hiérarchiques se divisent principalement en deux groupes :

- Les algorithmes de clustering hiérarchique ascendant, construisant la hiérarchie de bas en haut. Ils commencent avec autant de clusters que d'individus pour ensuite fusionner, à chaque étape, deux à deux les clusters jusqu'à obtenir un seul cluster racine. On parle de construction par agglomération.
- Les algorithmes de clustering hiérarchique descendant, construisant la hiérarchie de haut en bas, en commençant avec un cluster racine contenant l'ensemble des individus, qu'il sépare progressivement. On parle de construction par division.

Même si l'on trouve généralement beaucoup plus de références sur les méthodes ascendantes, nous avons orienté notre étude sur une approche ascendante *surtout* parce qu'il nous paraît plus intuitif de fusionner les objets dans le cas de notre projet.

Une série de méthodes hiérarchiques sont souvent citées et utilisées : *Average-link*, *Simple-link* et *Complete-link*.

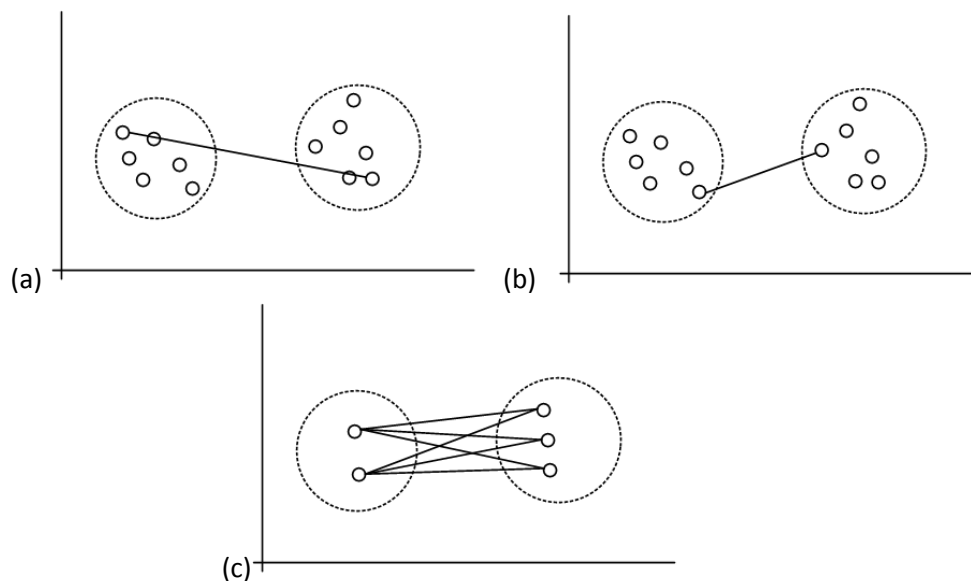


Figure 6: (a) Complete-link (b) Simple-link (c) Average-link

La Figure 6 montre comment ces méthodes sélectionnent chaque cluster suivant différentes approches :

- Simple-link cherche à fusionner les deux clusters qui possèdent la distance minimale entre les paires d'individus.
- Complete-link cherche à fusionner les deux clusters qui possèdent la distance maximale entre les paires d'individus.
- Average-link cherche à fusionner les deux clusters qui possèdent la distance moyenne minimale.

Très appréciées pour leurs facilités d'implémentation, elles permettent de produire rapidement des résultats. Cependant, le passage à l'échelle reste difficile avec une complexité de $O(N^2)$.

Simple-link est connu pour former rapidement des chaînes. Une *chaîne* est produite lorsque deux clusters sont fusionnés de manière prématurée ou non pertinente à cause d'une chaîne d'individus les reliant entre eux. Complete-link est lui très sensible aux valeurs isolées ou aberrantes (*outlier*) et a tendance à fabriquer des grappes. Average-link est souvent présenté comme un compromis entre ces deux approches.

Il peut être pertinent de fournir une représentation efficace pour l'utilisateur. Ainsi, le *dendrogramme* est souvent utilisé pour produire une image interprétable de la construction hiérarchique. Cette représentation matérialise deux informations pertinentes, les regroupements des différents clusters et les mesures de similarité entre deux regroupements :

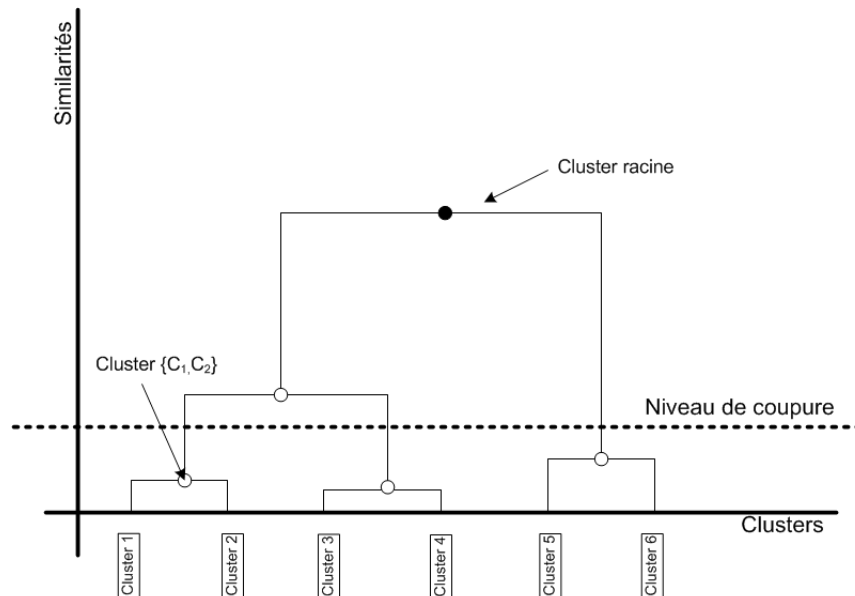


Figure 7: Dendrogramme simple

La Figure 7 illustre ce que pourrait être la représentation d'une hiérarchie obtenue avec six individus (les clusters en bas de l'image sont définis comme des singletons). Outre le fait que l'utilisateur peut interpréter plus facilement l'arborescence, le dendrogramme est souvent utilisé pour définir de manière empirique un *niveau de coupe*. Le niveau de coupe permet de ne garder qu'une partie des $2n - 1$ clusters de l'arborescence à partir d'une *partition* choisie comme pertinente. Chaque niveau de coupe représente une partition de l'ensemble de départ (ici $S = \{c_1, c_2, c_3, c_4, c_5, c_6\}$ ensemble des clusters de départ et S_p une partition produite après plusieurs regroupements donnée dans l'exemple par la ligne de coupe, soit $S_p = \{\{i_1, i_2\}, \{i_3, i_4\}, \{i_5, i_6\}\}$). De la même manière, pour obtenir un nombre plus réduit de clusters, l'utilisateur peut choisir une limite maximale de clusters présents dans la hiérarchie. L'avantage de ces algorithmes est qu'ils s'adaptent simplement aux fonctions de similarité de plusieurs types.

3.2.2.2 L'algorithme ROCK

Comme il est difficile de définir des similarités avec des mesures euclidiennes sur des individus composés de valeurs qualitatives, ROCK (**RO**bust, **Cl**ustering for **C**ategorical **D**ata) [GRS00], propose une approche qui repose sur un concept de *liens* entre les individus. On parle alors d'individus voisins. Pour cela nous devons définir :

- Soit, x_i et x_j deux individus, $Sim(x_i, x_j)$ une fonction de similarité, θ un seuil fourni par l'utilisateur, alors x_i et x_j sont voisins si $Sim(x_i, x_j) > \theta$.
- Une fonction $link(x_i, x_j)$ qui représente les voisins que x_i, x_j ont en commun.
- Une fonction dite d'*interconnectivité* entre deux clusters C_i, C_j donnée par :

$$link[C_i, C_j] = \sum_{x_i, x_j \in C_i} \sum_{x_i, x_j \in C_j} link(x_i, x_j)$$

Ainsi, dans ce cas le critère de regroupement n'utilise pas les similarités. Il est défini par le nombre de liens que partagent les individus des deux clusters. ce critère est donné par :

$$g[C_i, C_j] = \frac{\text{link}[C_i, C_j]}{(n_i + n_j)^{1+2f(\theta)} - n_i^{1+2f(\theta)} - n_j^{1+2f(\theta)}}$$

Avec $n_i^{1+2f(\theta)}$ le nombre de voisins (liens) qui doivent se retrouver dans le cluster C_r et $f(\theta)$ proposé par les auteurs sous la forme :

$$f(\theta) = \frac{1-\theta}{1+\theta}$$

Ainsi ROCK commence par sélectionner un échantillon d'individus puis agrège jusqu'à la racine s'il ne possède pas de clause d'arrêt pour limiter le nombre de clusters. ROCK possède une complexité de $O(n^2 + nm_m m_a + n^2 \log n)$ avec m_m nombre maximum de voisins d'un individu et m_a le nombre moyen de voisins d'un individu. ROCK permet de prendre en compte les valeurs isolées.

Nous pouvons aussi évoquer *Chamelon* [KHK99], un algorithme basé sur un graphe des *k-plus* proches voisins et *CURE* [GRS01], plus adapté à des espaces numériques de grandes tailles. Pour limiter notre sélection, nous retenons essentiellement ROCK, et les clustering hiérarchiques ascendants. ROCK pour sa capacité à manipuler des données qualitatives. Les clustering hiérarchiques ascendants pour leur flexibilité à utiliser tout type de fonctions similarité, et le peu de paramètre à fournir. Le Tableau 1 rappelle les caractéristiques importantes de ces algorithmes :

Tableau 1: Récapitulatif des exemples des algorithmes ROCK, Simple-link, Complet-link, Average-link

Méthodes	Particularités remarquables
Simple-link	- Basé sur la similarité minimum entre deux individus appartenant à deux clusters différents - Tendance à former des chaînes - Utilise les distances pour discriminer les clusters - S'adapte simplement aux diverses fonctions de similarité - Traitements coûteux
Complete-link	- Adapté à la formation de grappe - Utilise les distances pour discriminer les clusters - S'adapte simplement aux diverses fonctions de similarité - Traitements coûteux
Average-link	- Utilise les distances pour discriminer les clusters - Traitements coûteux - Compromis avec Simple-link, Complet-link - S'adapte simplement aux diverses fonctions de similarité
Rock	- Adapté pour des données catégorielles - Est basé sur un concept de lien pour regrouper les individus - Résistant aux valeurs isolées

3.2.3 Difficulté de l'évaluation

La *qualité* d'une classification produite est difficile à estimer en ne disposant pas une définition précise de ce qu'est un bon cluster pour les données dont nous disposons. D'une manière générale, il existe deux approches. La première est de considérer une connaissance sur le domaine (un expert, un jeu de donnée étiqueté, etc.), soit, un référentiel permettant d'évaluer la divergence avec une classification produite automatiquement. Nous parlons alors de mesure externe. La seconde, appelée mesure interne, se base sur des critères objectifs de la construction qui prend en compte les mesures (distances, similarités) entre les individus ainsi qu'entre les clusters formés.

3.2.3.1 La qualité interne

La qualité interne d'une classification se base principalement sur deux concepts : (I) produire des clusters avec la plus grande cohésion possible, en d'autres termes, maximiser la similarité de l'ensemble des membres d'un même cluster, (II) obtenir des clusters les plus distincts, ce qui revient à avoir l'écart le plus important entre chacun des clusters.

L'une des approches possible est de mesurer la cohésion *intra-cluster*. Par exemple, l'erreur quadratique pour chaque cluster produit dans la classification C avec $K = |C|$ et $dist$ une distance que l'on cherche à minimiser :

$$E^2(C) = \sum_{k=0}^K \sum_{x_i \in C_k} dist(\vec{x}_i, \vec{\rho}_k)$$

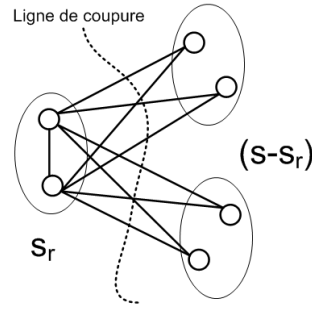
Avec le centroïde $\vec{\rho}_k$ donné par :

$$\vec{\rho}_k = \frac{1}{|C_k|} \sum_{i \in C_k} \vec{x}_i$$

D'autres approches basées sur l'étude de la cohésion d'un cluster sont souvent employées [Can07] comme la radius que l'on cherche à minimiser :

$$Radius(C_k) = \frac{1}{|C_k|} \sum_{i=1}^K dist(\vec{x}_i, \vec{\rho}_k)$$

Une autre approche est de considérer la cohésion intra-cluster avec l'écart extra-cluster simultanément. Par exemple, [ZK04] évalue une mesure basée sur un graphe G_s obtenu en modélisant l'ensemble des individus de S par un sommet dont chaque arrête représente la similarité entre chaque individu. Dans la figure 4 $cut(S_r, S - S_r)$ est défini comme ligne de coupure entre l'ensemble des sommets S_r délimité par le cluster r et le reste des sommets du graphe $(S - S_r)$.

Figure 8: Graphe G_s

Afin de prendre en compte la cohésion *intra-cluster*, nous utilisons la valeur des arrêtes entre chaque sommet appartenant à S_r afin de mettre à l'échelle la première mesure. Nous définissons $Q_{MinMax}(r)$ comme la qualité d'un cluster r calculée avec le graphe G_s :

$$Q_{MinMax}(r) = \frac{\sum_{x_i \in S_r, x_j \in S-S_r} Sim_{\delta}(x_i, x_j)}{\sum_{x_i, x_j \in S_r} Sim_{\delta}(x_i, x_j)}$$

Il faut rappeler que cette mesure est applicable que sur une *partition de la classification*, soit dans le cas d'une hiérarchie à un niveau de coupure. De plus, les mesures internes évaluent la qualité de la formation des clusters, elles n'assurent pas que les clusters soient pertinents. Souvent, ce type de mesure est croisé par une vision plus subjective telle que la mesure de qualité externe ou la vision d'un expert.

3.2.3.2 La qualité externe

La qualité externe est plus complexe à produire car elle demande d'avoir une connaissance au préalable sur la construction que l'on souhaite obtenir. Deux mesures sont utilisées couramment pour estimer la concordance entre un ensemble de regroupement que l'on sait pertinent (que nous nommerons *classe*) et les clusters produits dans l'arborescence. Ces mesures de qualité utilisent des critères connus que sont *Recall* et *Precision*. Avec un cluster C_k quelconque de la hiérarchie et une classe i .

$$Recall(C_k, i) = \frac{|C_k \cap i|}{|i|}$$

Le rappel (Recall dans l'exemple) évalue si le cluster a une part importante du groupement défini par la classe.

$$Precision(C_k, i) = \frac{|C_k \cap i|}{|C_k|}$$

La précision (Precision dans l'exemple) évalue le bruit qu'il y a entre les individus de la classe et l'ensemble des individus que possède le cluster.

Ainsi le F-score d'un cluster C_k et d'une classe i sera donné par :

$$Fscore(C_k, i) = \frac{2 \times Recall(C_k, i) \times Precision(C_k, i)}{Recall(C_k, i) + Precision(C_k, i)}$$

Un cluster C_k correspondant parfaitement à la classe i aura un *Fscore* de 1. Ainsi avec H une hiérarchie et I un ensemble de classe:

$$Fscore_{clustering}(H, I) = \sum_{i \in I} \frac{|i|}{N} \max_{C_k \in H} Fscore(C_k, i)$$

Pour prendre en compte la taille de la classe i le Fscore est très souvent pondéré par le rapport de la taille de la classe i avec le nombre d'individus N . De la même manière, la mesure de l'Entropie peut s'appliquer sur un ensemble de clusters et de classes :

$$Entropy(C_k) = - \sum_{i \in I} Precision(C_k, i) \times \log(Precision(C_k, i))$$

$$Entropy(H, I)_{clustering} = \sum_{C_k \in H} \frac{|C_k| \times Entropy(C_k, i)}{N}$$

L'entropie mesure la dispersion de l'information, alors que le Fscore cherche plutôt à mesurer la relative exactitude des résultats.

3.2.4 Conclusion

La littérature dispose déjà de nombreuses méthodes pour classier des données de manière hiérarchique. Elle apporte des solutions adaptées suivant la nature des besoins (dimension de l'espace à traiter, la nature des données, le nombre de paramètres que doit fournir l'utilisateur, la représentation de l'arborescence, etc.). Ainsi, le choix d'une approche se fait souvent à partir d'un compromis dans le choix de plusieurs composantes.

Pour cette étude, nous souhaitons récupérer les similarités utilisées pour classier les clusters afin de les exploiter dans la construction d'une ontologie. Nous souhaitons limiter les paramètres à fournir. Enfin, nous soulignons que le nombre de patrons à classier est très réduit et même si celui-ci peut se développer, il restera dans le même ordre de grandeur. Pour notre étude, l'algorithme *ROCK* pourrait être adapté, mais il n'utilise pas les similarités pour fusionner les clusters. Or, les similarités seront exploitées pour construire et enrichir l'ontologie. L'algorithme *Chameleon* reste sensible aux données isolées et demande plus de paramètres. Ainsi, il nous a paru pertinent d'orienter l'étude sur l'algorithme *Average-link*. Sa flexibilité permet d'utiliser différentes sortes de fonctions de similarité et ne requiert qu'un nombre de clusters final.

3.3 Une méthode de clustering hiérarchique pour les données descriptives

Cette section présente le contexte d'étude pour la fabrication d'une hiérarchie de données déclaratives basées sur des patrons XML. La section 1 détaille les données brutes qui sont extraites. Ensuite, nous proposons d'évaluer l'impact de la qualité des résultats obtenus en utilisant l'algorithme de classification *Average-link* suivant plusieurs critères :

- Section 3.3.2, différentes représentations des données : un encodage de base adapté à l'algorithme, et une série de transformations appliquées sur ce codage de base pour améliorer la classification.

- Section 4.3.3, la description de trois fonctions de similarité que nous étudions. Deux fonctions classiques (*Jaccard* et *Cosine*) et une troisième (*SoftJaccard_w*) que nous proposons.

Enfin, dans la section 3.3.4, nous détaillons l'algorithme *Average-link*.

3.3.1 Description des données d'entrées

L'*UPnP forum* fournit un ensemble de *spécifications* de *devices* et *services* de base respectant la convention UPnP. Dans le jargon UPnP, un *device* est un dispositif offrant des *services*. Par exemple, un *device* peut être un serveur de média, un scanner, une caméra de surveillance, etc. offrant respectivement la possibilité de récupérer un flux media, numériser des documents ou de contrôler une caméra.

L'ensemble des *devices* et *services* est défini au travers de descripteurs XML qui permettent, entre autre, de décrire les informations nécessaires à leurs utilisations (localisation, état du dispositif, la version, etc.) ainsi que des informations non fonctionnelles comme le nom du fabricant ou la description du modèle etc.

Dans notre étude, nous avons extrait les *patrons exemple* fournis par UPnP Forum pour en exploiter le contenu et former notre échantillon d'analyse. Il est important de noter que le nombre de patrons exemple récupérés, après extraction, est inférieur à 100, ce qui est réduit pour le type de données considérées et peut représenter un handicap.

Dans les deux exemples fournis (Annexes 1 et 2), nous pouvons observer les deux sortes de patrons extraits se déclinant en plusieurs types. Les patrons des *devices* comportent des champs pour des informations générales comme la version, le numéro du modèle ou le nom du fabricant, mais également, le type du *device* et les références aux *services* qu'il offre avec les balises *<serviceType>*.

```
<deviceType>urn:schemas-upnp-org:device:RemoteUIClientDevice:1</deviceType>
....
<serviceList>
  <service>
    <serviceType>urn:schemas-upnp-org:service:RemoteUIClient:1</serviceType>
    <serviceId>urn:upnp-org:serviceId:RemoteUIClient1</serviceId>
    <SCPDURL>URL to service description</SCPDURL>
    <controlURL>URL for control</controlURL>
    <eventSubURL>URL for eventing</eventSubURL>
  </service>
</serviceList>
```

Ici, le patron du device *RemoteUIdevice* offre un service *RemoteUIClient*. Les balises *SCPDURL* *controlURL* et *eventSubURL* représentent des localisations pour récupérer une description et interagir avec le service. Ces derniers champs ne sont renseignés que sur des instances de ce device.

Le patron des *services* apparaît comme étant une structure différente surtout composée d'actions (dans le jargon UPnP une action est une méthode du service), d'arguments ou de variables d'état.

```

<action>
  <name>GetCurrentConnections</name>
  <argumentList>
    <argument>
      <name>CurrentConnectionsList</name>
      <direction>out</direction>
      <relatedStateVariable>CurrentConnections</relatedStateVariable>
    </argument>
  </argumentList>
</action>
<action>
  .....

<stateVariable sendEvents="no">
  <name>CurrentConnections</name>
  <dataType>string</dataType>
</stateVariable>

```

Ici, nous observons comment chaque argument d'une action est explicitement lié à une variable d'état. Nous noterons que les patrons des *services* sont composés essentiellement de données techniques ne possédant pas la définition de leur type dans leurs descripteurs.

3.3.2 Représentation, transformation et enrichissements des données

Données

Nous souhaitons choisir une représentation facilement adaptable aux diverses fonctions de similarité utilisées. Nous avons sélectionné une approche similaire à [VFLD06] qui transforme les structures XML en des ensembles d'attributs adaptés aux hypothèses suivantes :

- Les données sont sous la forme de document semi-structuré (format XML) et les valeurs sont pour une très grande partie constituées de variables catégorielles (i.e. une URL, le nom d'une action, d'un état ou d'un argument).
- La profondeur des arbres est peu élevée et les arbres se répètent souvent, ce qui nous fait considérer la structure comme peu complexe.

Tableau 2: (a) Une représentation hiérarchique (XML) et (b) sa correspondance sous forme d'ensemble d'attributs

Diagram (a) illustrates a hierarchical tree structure. The root node is 'a'. Node 'a' has two children: 'b' and 'c'. Node 'b' has two children: 'd' and 'd'. Node 'c' has one child: 'e'. Node 'e' has two children: 'f' and 'g'. Node 'f' has two children: 'h' and 'i'. The values associated with each node are shown in brackets below the node: 'd' is '[d_{1_value}]', 'd' is '[d_{2_value}]', 'g' is '[g_value]', 'h' is '[]', and 'i' is '[i_value]'. Nodes 'a', 'b', 'c', 'e', and 'f' are white circles, while nodes 'd', 'd', 'g', 'h', and 'i' are gray circles.

Ensemble d'attributs
<i>a.b.d</i> @ <i>[d_{1_value}]</i>
<i>a.b.d</i> @ <i>[d_{2_value}]</i>
<i>a.c.e.g</i> @ <i>[g_value]</i>
<i>a.c.e.f.h</i> @ <i>[]</i>
<i>a.c.e.f.i</i> @ <i>[i_value]</i>

Le Tableau 2 illustre la transformation d'une arborescence XML en un ensemble d'attributs. Un attribut représente le chemin complet de la racine à la feuille. Si la feuille possède une valeur, celle-ci est différenciée et ajoutée à la fin de l'attribut. Avec cette représentation, les listes possédant les mêmes valeurs seront réduites en un seul attribut, dans notre exemple l'attribut avec le chemin $a.b.d$ sera unique si $d_2_value = d_1_value$.

Nous définissons aussi un ensemble de transformation des données de départ visant à améliorer le résultat de la classification. En effet, nous pensons que certaines modifications, spécifiques au modèle (UPnP dans notre cas) améliorent le regroupement des individus.

Reconstruction des liens entre les documents

Chacun des *devices* contient une référence sur les services qu'ils offrent. Les *services* n'ont pas de référence sur les *devices* qui les hébergent. Rajouter ce lien paraît pertinent car nous considérons que deux services partageant les mêmes *devices* est un critère de rapprochement.

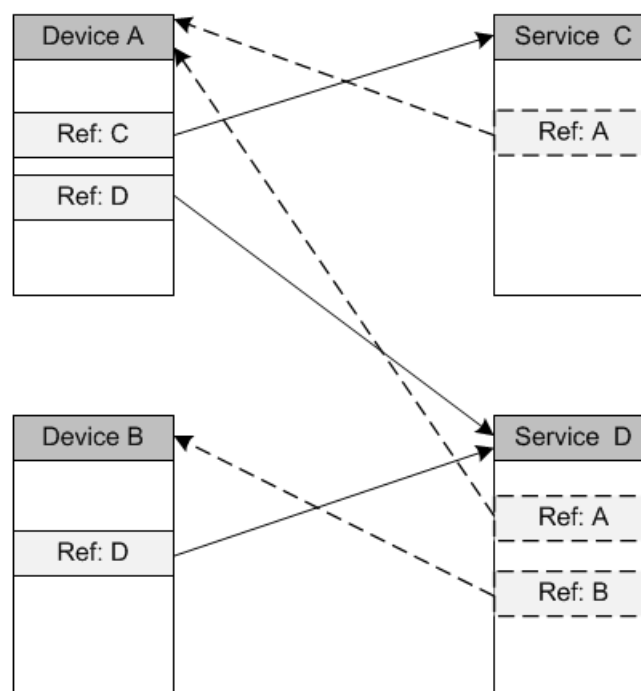


Figure 9: Ajout des références device et service

La Figure 9 illustre des *individus services* et des *individus devices* partageant ce type de référence.

Concrètement, dans le cas du device *DeviceA* nous ajoutons l'attribut :

root.device.deviceType@[urn:schemas-upnp-org:device:DeviceA]

dans le *ServiceC* et le *ServiceD*. De même, les individus services ne possèdent pas le nom de leur type, nous rajoutons cet attribut pour que l'individu soit auto-descriptif.

Sérialisation Contextuelle

Aplatir la structure arborescente XML génère des attributs que nous interprétons un par un. Cependant, certains attributs ne sont pertinents que dans un contexte donné, en d'autres termes, ils n'ont de sens que dans une structure. Par exemple, la définition d'une action est composée d'un nom, de 0 à n arguments et d'un type d'argument. Il est préférable de conserver le contexte formé par cette arborescence et de le représenter par un seul attribut. Pour construire cet attribut nous fournissons à la fonction qui aplatit la source XML, les paramètres de contextualisation pour certains sous-arbres qui nous intéressent. Concrètement pour chaque sous-arbre t de la représentation XML que nous souhaitons contextualiser nous fournissons $\langle C_t, E_t \rangle$. Avec, C_t un contexte représenté par un chemin $a.b.c.d$ à partir de la racine jusqu'au nœud où l'on commence la contextualisation. E_t un ensemble ordonné de chemins de ce sous-arbre. Ainsi nous formons un attribut unique composé de C_t concaténé avec les valeurs des derniers noeuds des chemins de E_t . Comme il y a plusieurs chemins dans le sous-arbre nous récupérerons ces valeurs suivant l'ordre d'apparition dans l'arbre XML. Dans les figures 3 et 6 nous fournissons C_t comme chemin $Root.ActionList.Action$ et pour E_t ($Action.Name$, $Action.Arg.Name$) (Figure 10).

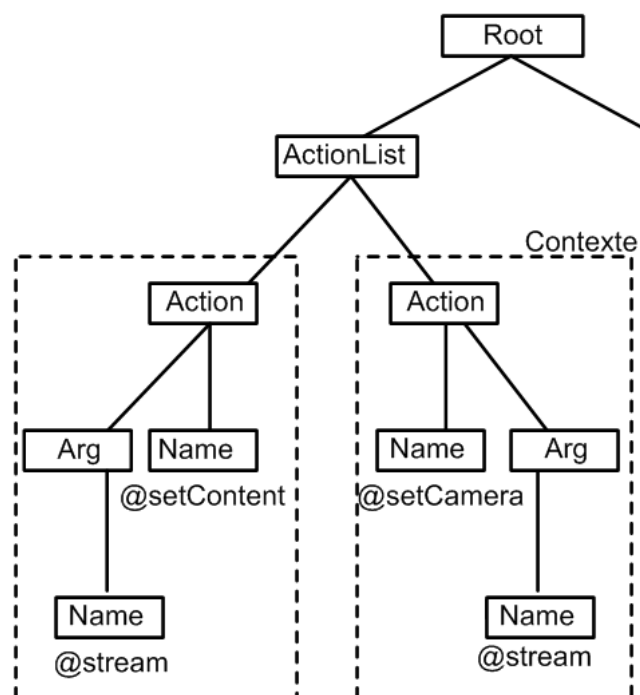


Figure 10: Contextualisation des sous-arbres

Tableau 3:(a) et (b) exemples simplifiés d'une contextualisation d'un ensemble d'attributs

(a)	(b)
Sans contexte	Avec contexte
Root.ActionList.Action.Name@[setCamera]	Root.ActionList.Action@[setCamera.stream]
Root.ActionList.Action.Arg.Name@[stream]	Root.ActionList.Action@[setContent.stream]
Root.ActionList.Action.Name@[setContent]	
Root.ActionList.Action.Arg.Name@[stream]	

Dans l'échantillon réalisé avec les données UPnP, nous nous limitons à contextualiser seulement les sous-arbres décrivant les actions et les variables d'état.

Ainsi nous éviterons de rapprocher des individus sur des critères trop abstraits comme le type d'un argument. Le nœud où l'on décide de contextualiser représente la profondeur.

Ce processus permet aussi de réduire le nombre d'attributs, ce qui améliore grandement le temps de traitement.

Nettoyage des attributs

Nous effectuons aussi un nettoyage des données présentes dans le contenu qui pourraient bruyé les mesures entre les attributs. Par exemple, les arguments des actions sont tous préfixés avec la chaîne **A_ARG_TYPE** ou la déclaration des types de service avec **urn:schemas-upnp-org:service:..**. Dans ce cas nous les supprimons.

3.3.3 Fonctions de similarité

De nombreuses fonctions de similarité existent dans la littérature. Nous avons retenu deux fonctions : Jaccard pour sa faculté à comparer deux ensembles de valeurs couramment utilisée avec des données catégorielles, et Cosine utilisée avec une approche par pondération. Enfin, nous proposons une fonction de similarité *SoftJaccard* permettant de relâcher les contraintes dans la comparaison des attributs.

3.3.3.1 Jaccard

L'indice de Jaccard est une métrique largement utilisée pour comparer la similarité entre deux ensembles. Dans ce cas, nous définissons un ensemble par un individu et ses attributs.

$$indiceJaccard(x_i, x_j) = \frac{|x_i \cap x_j|}{|x_i \cup x_j|}$$

Avec notre codage, la similarité des deux individus est évaluée par le rapport entre l'intersection et l'union des attributs. Nous obtenons une valeur normalisée (i.e. valeur $\in [0,1]$) avec une similarité de 1 indiquant des individus identiques et 0 des individus complètement différents.

Avec la mesure de similarité Jaccard, les attributs sont évalués comme autant de valeurs booléennes. Chaque attribut possède ainsi le même poids.

3.3.3.2 Cosinus et Tfidf

Cela reste assez compliqué de proposer un poids pour chaque attribut de manière subjective, c'est pourquoi nous proposons d'utiliser *Cosine*, une fonction de similarité qui est souvent employée conjointement avec le Tfidf dans la littérature pour fournir des poids sur les individus d'un corpus.

Cosine est une fonction qui mesure la similarité entre deux vecteurs en évaluant l'angle formé entre eux. Plus formellement:

Nous définissons deux individus comme deux vecteurs de dimension k soit $\vec{x}_i = (x_{i1}, \dots, x_{ik})$ et $\vec{x}_j = (x_{j1}, \dots, x_{jk})$ alors :

$$\text{Cosine}(\vec{x}_i, \vec{x}_j) = \frac{\vec{x}_i \cdot \vec{x}_j}{\|\vec{x}_i\| \cdot \|\vec{x}_j\|} = \frac{\sum_{n=1}^k x_{in} \cdot x_{jn}}{\sqrt{\sum_{n=1}^k (x_{in})^2} \cdot \sqrt{\sum_{n=1}^k (x_{jn})^2}}$$

Les valeurs d'un vecteur représentant le Tfidf de l'attribut n . De la même manière que *Jaccard*, *Cosine* fournit une valeur normalisée.

Le *Tfidf* (term frequency-inverse document frequency) évalue le poids d'un attribut par sa fréquence dans l'individu ainsi que sa fréquence dans le corpus. De manière générale, le Tfidf d'un attribut sera plus significatif s'il est souvent présent dans l'individu et rare dans le corpus. Nous calculons, en amont de la classification, pour chaque individu, le Tfidf de chaque attribut Soit avec l'individu x_j :

$$tf_{i,j} = \frac{n_{i,j}}{\sum_k n_{k,j}}$$

Avec :

$n_{i,j}$ fréquence de l'attribut t_i pour l'individu x_j

$\sum_k n_{k,j}$ somme de l'ensemble des attributs de l'individu x_j

$$idf_i = \log \left(\frac{|D|}{|\{x_j : t_i \in x_j\}|} \right)$$

$|D|$ nombre d'individus du corpus

$|\{x_j : t_i \in x_j\}|$ nombre d'individus x_j possédant l'attribut t_i

Soit, le *Tfidf* de l'attribut t_i pour l'individu x_j

$$tfidf_{i,j} = tf_{i,j} \cdot idf_i$$

Un individu est maintenant représenté par des attributs ayant chacun un poids :

Tableau 4: Chaque attribut possède un poids

Individus x_j	
Attributs	TFIDF
$a.b.d@[d_1_value]$	$tfidf_{1,j}$
$a.b.d@[d_2_value]$	$tfidf_{2,j}$

$a.c.e.g@[g_value]$	$tfIdf_{3,j}$
$a.c.e.f.h@[]$	$tfIdf_{4,j}$
$a.c.e.f.i@[i_value]$	$tfIdf_{5,j}$

3.3.3.3 SoftJaccard

Dans certains cas il semble intéressant de rapprocher des individus partageant des attributs qui ne sont pas strictement identiques.

Tableau 5: (a) et (b) deux simplifications d'individus avec des attributs partageant de l'information

(a)

houseService
<i>getTemperature()</i>
<i>getNumberSensor()</i>
<i>sensorNumber</i>
<i>TemperatureRoomState</i>

(b)

sensorService
<i>stopSensorTemperature(sensorName)</i>
<i>startSensorTemperature(sensorName)</i>
<i>TemperatureState</i>

Dans le Tableau 5 nous observons deux individus simplifiés avec des attributs tous différents au sens *strict*. En appliquant une des mesures de similarité vue plus haut, les individus seraient considérés comme dissemblables alors qu'ils partagent visiblement de l'information.

Il existe peu d'exemple de fonctions de similarité *applicable* sur un ensemble de valeurs permettant de relâcher les contraintes sur les attributs. [Sai07] utilise une version modifiée de Jaccard. Cependant cette mesure n'utilise pas de pondération sur les attributs. Nous proposons une fonction similaire qui prend en compte les poids sur les attributs qui seront fournis par TfIDF.

Nous proposons $softJaccard_{\omega}$ pour rapprocher les individus avec de l'information bruitée:

Avec x_i et x_j deux individus :

$$softJaccard_{\omega}(x_i, x_j) = \frac{\sum_{t \in CLOSE(x_i, x_j, \theta)} W(t)}{\sum_{t \in x_i \cup x_j} weight_{x_i}(t) + weight_{x_j}(t)}$$

$CLOSE(x_i, x_j, \theta)$ représente l'ensemble des attributs $t \in x_i$ $t' \in x_j$ dont $Sim_{\alpha}(t, t') > \theta$. Avec $Sim_{\alpha}(t, t')$ une deuxième fonction de similarité adaptée aux chaîne courtes (ici nous prenons Jaro-Winkler). $W(t)$ une fonction définissant le poids de l'attribut t avec :

$$W(t) = \begin{cases} weight_{x_i}(t) \cdot \frac{\max_{t' \in x_j} (Sim_{\alpha}(t, t')) - \theta}{1 - \theta} & \text{si } t \notin x_j \\ weight_{x_j}(t) \cdot \frac{\max_{t' \in x_i} (Sim_{\alpha}(t, t')) - \theta}{1 - \theta} & \text{si } t \notin x_i \\ weight_{x_i}(t) + weight_{x_j}(t) & \text{si } t \in x_i, x_j \end{cases}$$

Le poids d'un attribut est défini par $weight_{x_i}(t)$ pour x_i et par $weight_{x_j}(t)$ pour x_j . Ce poids est égal à zéro si l'attribut n'existe pas dans l'individu. Nous recadrons aussi le poids des attributs qui n'existent pas dans les deux individus dans $CLOSE(x_i, x_j, \theta)$ avec :

$$\frac{\max_{t' \in x_j} (Sim_{\alpha}(t, t')) - \theta}{1 - \theta}$$

Puisqu'il peut y avoir plusieurs similarités au-dessus du seuil, nous prenons la similarité maximale.

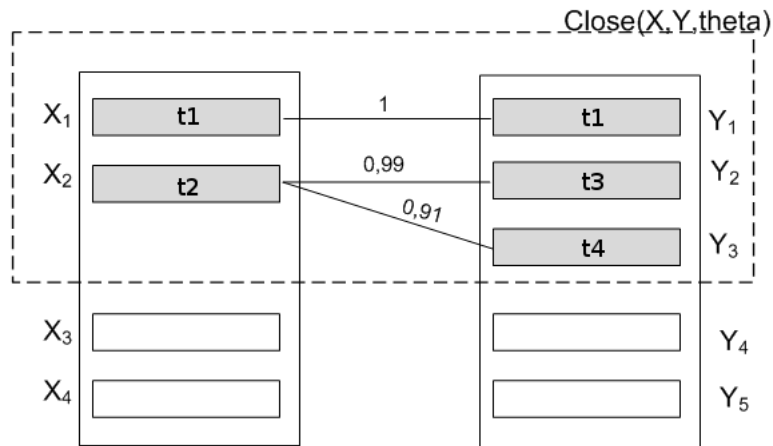


Figure 11: Exemple de rapprochement avec $softJaccard_{\omega}$

La Figure 11 illustre le rapprochement de l'individu X et Y . $CLOSE(x_i, x_j, \theta)$ est représenté par les attributs grisés, et les similarités au-dessus du seuil θ par les liens entre X et Y . Ainsi par exemple avec un seuil $\theta = 0,9$:

$$W(t4) = weight_Y(Y_3) \times 0,10$$

et pour $W(t4)$, $t4$ ne partage qu'une similarité avec l'attribut $t2$ qui est de 0,91.

$$W(t2) = weight_X(X_2) \times 0,90$$

Puisque $t2$ a une similarité maximum avec l'attribut $t3$ de 0,99.

$$\{a.b.c.d.e.f.g.h.i.j@[start(arg1)], a.b.c.d.e.f.g.h.i.j@[stop(arg2)]\}$$

Tableau 6: Estimation d'un attribut avec et sans son chemin (Jaro-Winkler)

Estimation avec $Sim_{\alpha} = \text{Jaro-Winkler}$	
Chemin + valeur	Valeur
.96674067	0.78493506

Afin de rendre la comparaison plus robuste nous appliquons Sim_{α} seulement sur une partie de l'attribut (matérialisé en gras sur le tableau 6) qui est la concaténation de la valeur trouvée dans l'arborescence XML. Comme l'illustre le Tableau 6 si nous définissons le seuil à 0,90, en considérant le chemin et sa valeur, nous rapprochons des attributs qui peuvent être éloignés. Dans ce cas, nous

effectuons des rapprochements plus importants sur la structure. Avec un ensemble de patrons où la structure varie peu, cela ne se révèle pas pertinent.

3.3.4 L'algorithme Average-Link

L'approche que nous avons expérimentée se base sur *Average-link* qui est un algorithme de la famille des *Clustering Hiérarchique Ascendant*. Cet algorithme est souvent utilisé pour sa simplicité tout en offrant des résultats satisfaisants. Les facteurs limitant comme le passage à l'échelle et l'impossibilité d'ajouter des individus de manière incrémentale restent acceptables puisque le nombre d'individus est très réduit. Ainsi, dans le cas d'un ajout, nous pouvons reconstruire intégralement la hiérarchie. Cependant, sa relative sensibilité à l'effet de chaîne reste un point important que nous devons intégrer à notre évaluation. L'algorithme AVGLink (présenté ci-dessous) prend en entrée un ensemble de clusters C dont chacun possède une référence sur un individu de la base (singleton). Il commence par calculer la matrice des similarités avec $build_similarity_matrix(C)$ entre chaque individu. Lorsque nous utilisons *Cosine* ou *softJaccard* _{ω} , $build_similarity_matrix(C)$ doit calculer le Tfidf des individus avant de les comparer. A chaque étape un nouveau cluster est créé avec une référence sur les deux clusters à fusionner et celui-ci est ajouté dans la liste globale alors que les références des clusters qui viennent d'être fusionnés sont supprimées. Puisque nous avons $n-1$ cluster à chaque étape nous arrêtons l'algorithme lorsqu'il ne reste plus qu'un cluster racine. L'algorithme retourne une hiérarchie de clusters.

Algorithm 1 AVGLink**Require:** C {Ensemble des clusters avec $\forall i \in S \exists c \in C$ }

```

1:  $Sim_C \leftarrow \text{build\_similarity\_matrix}(C)$ 
2: while  $|C| > 1$  do
3:    $c_{candidate}^i \leftarrow \emptyset$ 
4:    $c_{candidate}^j \leftarrow \emptyset$ 
5:    $similarity_{max} \leftarrow 0$ 
6:   for all  $c_i \in C$  do
7:     for all  $c_j \in C$  do
8:        $similarity_{current} \leftarrow \text{getSimilarityCluster}(c_i, c_j, Sim_C)$  { La
        similarité entre deux clusters avec l'average-link s'obtient avec
         $\left(\frac{1}{n} \sum_{x_i \in c_i, x_j \in c_j} Sim_\delta(x_i, x_j)\right)$  avec  $n = |c_i| \cdot |c_j|$  }
9:       if  $similarity_{current} \geq similarity_{max}$  then
10:         $c_{candidate}^i \leftarrow c_i$ 
11:         $c_{candidate}^j \leftarrow c_j$ 
12:         $similarity_{max} \leftarrow similarity_{current}$ 
13:       end if
14:     end for
15:   end for
16:    $c_{new} \leftarrow \text{merge}(c_{candidate}^i, c_{candidate}^j)$ 
17:    $\text{delete}(c_{candidate}^i, c_{candidate}^j, C)$ 
18:    $C \leftarrow c_{new}$ 
19: end while
20: return  $C$  {retourne les clusters hiérarchisés }

```

3.4 Evaluations et interprétations

Ce chapitre est composé de deux parties. La première fournit notre protocole de validation et la seconde propose une interprétation des résultats.

3.4.1 Validation d'une hiérarchie

Cette section décrit l'évaluation des hiérarchies obtenues. Nous commençons par définir les différentes méthodes de validation.

Evaluation de la qualité intra/extra cluster

Pour évaluer la qualité interne, nous avons utilisé l'approche similaire au MinMaxCut qui prend en compte l'aspect intra-cluster et extra-cluster simultanément. Cette mesure de qualité ne pouvant s'appliquer que sur une partition, nous proposons de mesurer la qualité **globale** de notre cluster avec Q_1 soit :

$$Q_1 = \sum_{i=0}^n \left(\frac{1}{|p_i|} \sum_{j \in p_i} Q(r_j) \right)$$

Avec n le nombre de partitions formées par l'algorithme, $Q(r_j)$ et le MinMaxCut du cluster r_j , enfin, p_i la i^{ieme} partition.

Nous souhaitons aussi estimer la variation de cette qualité sur chaque partition. Pour cela, nous évaluons l'*Écart type* de la qualité de chaque partition, ainsi Q_2 nous donne une estimation de la variation globale :

$$Q_2 = \sum_{i=0}^n \sqrt{\frac{1}{|p_i|} \sum_{j=0}^{|p_i|} (Q(r_j) - Q_m(p_i))^2}$$

Avec $Q_m(p_i)$ la qualité moyenne d'une partition, ici donnée par $\frac{1}{|p_i|} \sum_{j \in p_i} Q(r_j)$. Plus le cluster a une qualité homogène plus la valeur sera faible. Ainsi, plus nous obtenons des valeurs faibles, meilleure est la qualité pour Q_1 et Q_2

Evaluation externe avec jeu de données artificielles

Pour cette étude, nous ne possédons pas d'un ensemble de données se rapprochant suffisamment du modèle UPnP. Afin de mesurer la capacité des fonctions de similarité à sélectionner des regroupements pertinents, nous avons généré artificiellement des individus puis nous les avons liés à un ensemble de classes spécifiques. En appliquant le clustering avec différentes mesures de similarité sur ces individus artificiels, nous estimons la corrélation entre les clusters produits et l'ensemble des classes prédéfinies.

Pour produire notre jeu de données nous avons généré chaque individu avec un ensemble d'attribut dont le nombre est estimé aléatoirement dans un intervalle (dans notre cas, nous souhaitons nous rapprocher du nombre d'attribut moyen de nos données à savoir [10,20]). Chaque attribut est défini par $a.b.c.d@[random]$ avec *random* une chaîne de caractères aléatoires (en pratique nous utiliserons une chaîne de 20 caractères pour 500 individus pour avoir une très forte probabilité d'unicité). Nous avons défini un ensemble de classes I représentées chacune par un label. Pour lier un individu à une classe, nous rajoutons à l'individu un attribut $a.b.c.d@[label_i]$ avec $label_i$ le label de la classe i . Ainsi, chaque individu appartient à n classes Figure 12 .

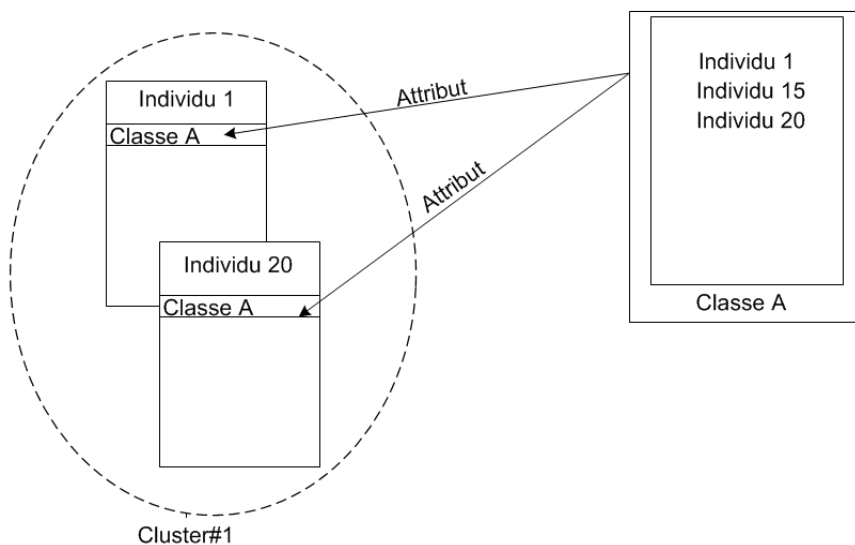


Figure 12: Le cluster contient deux individus liés à la classe A

Le nombre de classes pour chaque individu dépend de la distribution choisie, dans notre cas nous effectuons une distribution dichotomique simple.

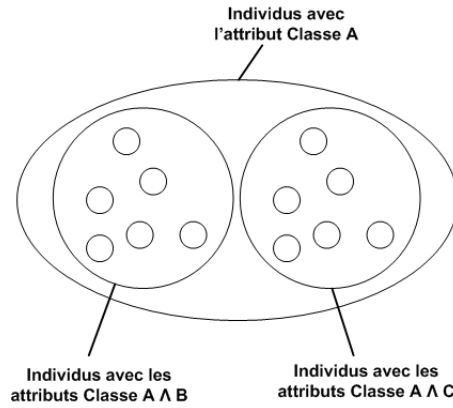


Figure 13: Exemple de séparation en 3 classes distinctes

La Figure 13 illustre la séparation effectuée pour lier chaque individu à une classe. Dans l'exemple 9, les 12 individus possèdent un attribut avec le label de la classe A, 6 possèdent l'attribut de la classe B et 6 possèdent l'attribut de la classe C.

Dans cette expérimentation, nous souhaitons observer plus particulièrement le comportement de $SoftJaccard_{\omega}$ face aux deux autres fonctions de similarité. Ainsi, nous bruitons les attributs pour simuler grossièrement des données non strictement identiques. Insérer du bruit revient à injecter des caractères aléatoires autour du $label_i$. Ainsi, en utilisant le $FScore_{clustering}(H, I)$ avec H l'ensemble des clusters de la hiérarchie et I l'ensemble des classes (section 3.2 page **Erreur ! Signet non défini.**), nous calculons la concordance entre les classes et les clusters produits. Puisque le FScore maximum entre un cluster et une classe est de 1, il est facile de calculer le FScore maximum pour toutes les classes et S l'ensemble des individus.

$$FScore_{Max}(I, S) = \sum_{i \in I} \frac{|i|}{|S|}$$

Les résultats sont présentés par Q_3 sous la forme

$$Q_3 = \frac{FScore_{clustering}(H, I)}{FScore_{Max}(I, S)}$$

Comme nous utilisons certaines données aléatoires (nombre d'attributs et bruit) nous prenons la moyenne de 5 estimations pour chaque fonction.

3.4.2 Interprétation

Résultats de la qualité interne Q_1 et Q_2

Le tableau 7 compare les résultats des mesures Q_1 et Q_2 obtenus lors de la construction. Ces résultats sont récupérés avec et sans la correction des données pour Q_1 et avec la correction des

données pour Q_2 . Nous entendons par correction des données les transformations (contextualisation des attributs ou la reconstruction des liens) que nous effectuons sur la représentation de base.

Cette validation s'est effectuée sur un ensemble 88 patrons. L'échantillon étant très petit les temps de traitement sont inférieures à 5 secondes pour Jaccard et Cosine. Par contre *SoftJaccard_ω* est bien plus lent puisqu'il exécute le traitement en 15 secondes. Il est fort probable que l'utilisation d'une seconde mesure de similarité diminue considérablement le temps de traitement.

Tableau 7: Qualité évaluée avec (avec et sans correction des données) et

Qualité	Q_1 avec corr.	Q_1 sans corr.	Q_2
Jaccard	262.66562	294.94135	170.36343
Cosine	66.12294	85.80193	95.90717
<i>SoftJaccard_ω</i> ($\theta = 0.9$)	121.5997	210.10025	174.49806

avec corr.: avec phase de traitement des données

sans corr : sans phase de traitement des données

Le *Cosine* forme des clusters avec des similarités beaucoup plus distinctes et les résultats semblent plus homogènes face à *SoftJaccard_ω* et *Jaccard*. Bien que cela n'assure pas la construction de clusters pertinents, cela peut être un critère important dans notre cas puisque nous considérons que les mesures peuvent être réutilisées comme une information dans la construction de l'ontologie. Nous observons aussi que la représentation des données a un impact très important pour *SoftJaccard_ω*. Cela semble dû à sa capacité à intégrer des ressemblances moins marquées par le relâchement des contraintes strictes sur les attributs et produire des clusters plus dilatés.

Résultats de la qualité externe Q_3

Tableau 8: Résultats avec différents ratios de bruit

Nombre d'individus bruités en %	0	25	50	75	100
Jaccard	0,93	0,83	0,69	0,58	0,54
Cosine	1	0,88	0,72	0,59	0,54
<i>SoftJaccard_ω</i> ($\theta = 0.9$)	1	0,82	0,81	0,80	0,83

Le tableau 8 présente le FScore normalisé avec différents ratios d'individus bruités. Nous observons que Jaccard n'arrive jamais à retrouver une concordance parfaite entre les classes et les clusters produits. C'est un mauvais résultat si l'on considère la faible complexité des données. Si nous ne bruitons pas les individus, *Cosine* et *SoftJaccard_ω* produisent pour chaque classe i un cluster parfaitement correspondant. Nous observons aussi qu'avec l'augmentation du bruit, *Cosine* perd sa capacité à produire des bons clusters alors que l'utilisation de *SoftJaccard_ω* permet de mieux résister au bruit.

Bien que les résultats soient significatifs, il faut considérer ces données comme un cas extrême. Cependant *SoftJaccard_ω* peut être une bonne alternative dans le cas où nous avons un lot de données partageant peu d'information mais pouvant être regroupé (par exemple, le cas d'un service ayant la même méthode avec l'ajout d'un argument)

3.5 Evaluations et interprétations du clustering hiérarchique

Des travaux récents [BJR] ont proposé de modéliser les objets intervenant dans le contexte sous une hiérarchie de classe. Cette représentation hiérarchique (taxonomie) est complexe à construire et demande une connaissance au préalable dès la construction ou à chaque enrichissement.

Nous avons proposé de construire **automatiquement** une telle hiérarchie en utilisant une méthode de clustering hiérarchique. Pour cette expérimentation, nous nous sommes basés sur l'environnement UPnP (Universal Plug and Play) adapté aux exigences des aspects ubiquitaires. Plus précisément, nous avons utilisé le contenu de patrons diffusés par UPnP Forum spécifiant un ensemble de types de dispositifs et de services couramment utilisés.

Dans cette étude, nous avons évalué la qualité de la hiérarchie produite à travers plusieurs critères que sont la représentation des données ou l'utilisation de différentes fonctions de similarités. Ainsi, nous avons évalué 3 mesures. *Cosine* et *Jaccard*, provenant de la littérature et *Softjaccard_ω*, une mesure de similarité que nous proposons afin d'améliorer la classification de données bruitées. Pour utiliser les patrons XML, nous avons proposé un mode de représentation simple et adapté aux spécificités de ces données déclaratives. Enfin, nous proposons une approche pour estimer la qualité globale de la hiérarchie obtenue avec cette technique.

3.6 Problème du nommage des clusters

Il y a eu de nombreux travaux sur les algorithmes de clustering hiérarchique et ces applications dans l'extraction d'information et la recherche en fouille de données. Cependant l'intérêt pour la définition de descripteurs ou labels des différents clusters est bien moindre [MU10][PU06]. La création automatique de labels pour des clusters ne fournit le plus souvent pas des descripteurs compréhensibles ou pertinents. Ces labels consistent généralement en une liste de termes sur laquelle une personne doit inférer manuellement une description générale.

The algorithm is based on the hypothesis that by comparing the word distribution from different parts of the hierarchy, it should be possible to assign appropriate labels to each cluster in the hierarchy.

Cette section propose de compléter notre méthode de clustering hiérarchique par un algorithme simple qui va automatiquement assigner un label concis mais suffisamment descriptif pour expliciter la sémantique de chaque cluster. L'algorithme combine des informations statistiques, la hiérarchie en présence, et le corpus à la base de la construction des clusters. Cet algorithme est bien adapté aux dispositifs UPnP, en particulier les dispositifs standards dont les classes de dispositifs possèdent beaucoup de propriétés communes. Une variante est également présentée et donne d'excellents résultats dans le cas de classes de dispositif sans recouvrement.

Cette section est organisée comme suit:

- une description des méthodes de labélisation étudiées
- la présentation des résultats expérimentaux et de la méthode de validation
- une synthèse sur la contribution et le lien avec la relaxation de requête (interrogation étendue de la BdC)

3.6.1 Méthodes de labélisation d'un cluster hiérarchique

Cette section présente d'abord la méthode générale de labellisation d'un cluster hiérarchique, puis introduit deux méthodes de labélisation qui peuvent chacune donner de bons résultats en fonction du domaine.

3.6.1.1 Méthodologie générale

Le processus de construction des descripteurs ou labels d'une hiérarchie de clusters se décompose en cinq étapes:

- La construction d'une hiérarchie augmentée avec les documents du corpus relatifs à chaque cluster.
- L'extraction et le traitement du texte des documents du corpus afin de ne conserver que des termes utiles pour les étapes suivantes
- La transformation des documents en ensemble de phrases avec des données statistiques (fréquence)
- Un score est associé à chaque phrase est calculé afin de sortir des phrases candidates à la labélisation de chaque cluster
- Les résultats sont filtrés et classés.

La Construction de la hiérarchie augmentée consiste à associer à chaque cluster de la hiérarchie les documents pertinents du corpus à partir desquels seront extrait les labels potentiels pour ce cluster.

En entrée de ce traitement :

- la hiérarchie de cluster obtenue à l'étape précédente. Chaque possède son propre identifiant numérique.
- un ensemble de fichiers décrivant les dispositifs et services relatifs à chaque cluster (nous considérons ici des documents dans le format de ressource UPnP standard).

En sortie: Une hiérarchie identique mais augmentée par des relations avec les différents fichiers ressources.

L'Extraction et traitement du texte consiste à reformater et filtrer le contenu des fichiers ressources afin de préparer le traitement d'identification des labels.

En entrée: La hiérarchie augmentée générée dans le traitement précédent.

En sortie: La même hiérarchie mais fichiers ressources ont été transformés afin de rendre le contenu exploitable dans une recherche de descripteur de cluster.

Les actions associées à cette étape sont au nombre de trois:

- Les documents sont transformés dans un format UTF-8.
- Les mots de liaisons inutiles à la compréhension et générant beaucoup de bruit (stop-words) sont retirés des documents ressources. Pour cela une liste préexistante est utilisée comme référence, cette liste est spécifique à chaque langue et/ou domaine d'application.
- Les mots composés ou expressions complexes sont découpés en mots en considérant le séparateur “_” et la présence de lettre capitales à l'intérieur d'un mot (nommage de méthodes et services).

La Transformation du texte en phrase est une étape clé du traitement. Elle transforme les documents en ensemble de mots et associe des statistiques utiles pour identifier les labels candidats.

En entrée: la hiérarchie avec les fichiers correctement reformatés.

En sortie: la même hiérarchie avec des ensemble de mots associés à chaque cluster.

Ce traitement génère des phrases à partir du contenu des fichiers ressources. Ces phrases peuvent être composées de un, deux ou trois mots (unigram, bi-gram, tri-gram). A chaque phrase est associée sa fréquence dans le document.

Le Calcul du score des phrases candidates est l'étape d'identification des labels candidats.

En entrée: la hiérarchie avec ses ensemble de phrases.

En sortie: la hiérarchie avec une liste de label candidats pour chaque cluster.

Le processus de nommage des clusters est assez simple. Il consiste à considérer toutes les phrases d'un cluster comme des candidats potentiels. Un score est calculé pour chaque phrase afin de définir un classement des meilleures phrases.

Le Filtrage des résultats est nécessaire pour conserver les meilleurs labels et évitant les recouvrements de descripteur entre cluster dont la sémantique est voisine.

Une valeur de coupure permet généralement de ne conserver que les meilleurs labels, mais ceci peut être insuffisant pour obtenir une taxonomie correcte:

$$\text{Cutoff_threshold} = \frac{\sum_{i=1 \rightarrow k} \text{score}(i)}{2k}$$

3.6.1.2 Approche Jensen-Shannon Divergence

La fonction de Jensen-Shannon Divergence (JSD) est calculée pour chaque phrase dans l'ensemble Dc des phrases d'un cluster Dc, en considérant le corpus de référence des phrases extraites de tous les documents ressources Dref. Dref est l'union de tous les documents associés aux clusters frères, sachant que la méthode de clustering hiérarchique retenue donne comme résultat un arbre binaire.

JSD(Dref, Dc) forme un vecteur à d dimensions:

$$\text{JSD}(\text{Dref}(t), \text{Dc}(t)) = P(t_{dc}) \log \frac{P(t_{dc})}{M(t)} + P(t_{dref}) \log \frac{P(t_{dref})}{M(t)}$$

$$M(t) = \frac{1}{2(P(t_{dc}) + P(t_{dref}))}$$

P(t) = nombre d'occurrences de t divisé par le nombre total d'occurrence de tous les termes.

Les k phrases avec le meilleur score JSD sont retenues comme labels candidats.

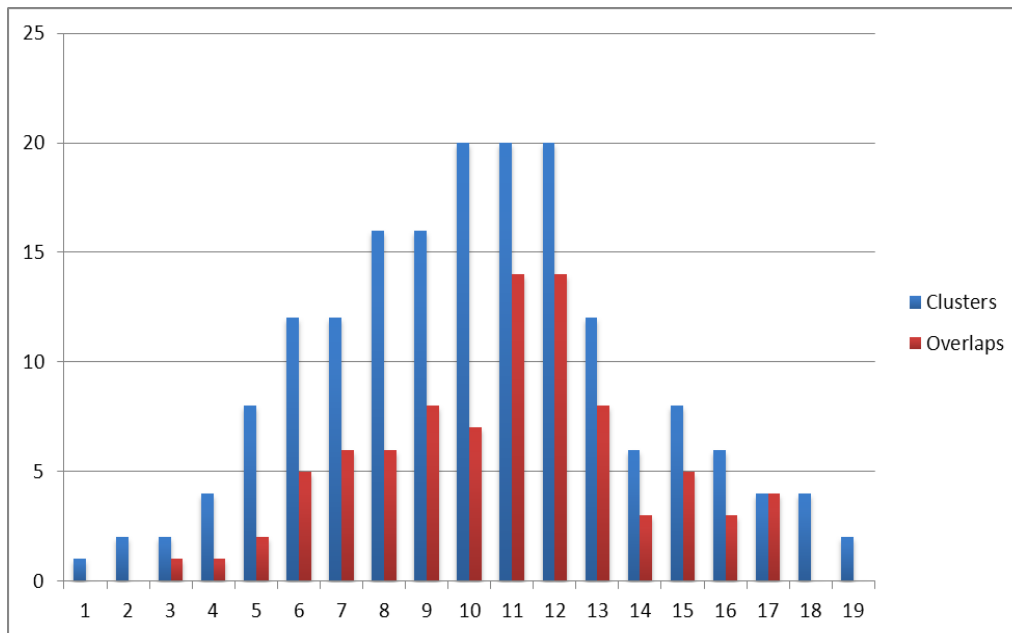
Les résultats obtenus à partir de l'ensemble des descriptions de dispositifs UPnP standard disponibles à l'UPnP forum donne le résultat suivant:

Nombre de cluster après hiérarchisation : 175

Nombre de dispositifs à l'origine: 88

Profondeur de l'arbre: 19

Largeur de l'arbre: 20



La figure ci-dessus montre la répartition des dispositifs sur les 19 niveaux de la taxonomie. Elle met surtout en évidence un recouvrement très important des labels pour les clusters d'un même niveau. Cela est principalement dû à une séparation assez floue de certains clusters. Certains clusters différents parfois par une seule méthode suffisamment discriminante pour l'algorithme de clustering automatique.

Pour résoudre en partie ce problème qui rend la taxonomie inutilisable, nous proposons un traitement a posteriori dans une étape supplémentaire prenant en entrée notre taxonomie avec ses descripteurs candidats.

Cette phase de post-traitement se décompose en deux étapes:

- Les labels qui sont en commun entre clusters frères sont supprimés des fils et ajoutés dans le cluster parent:

$$L_{Parent} = L_{Parent} \cup (L_{Self} \cap L_{Sibling})$$

$$L_{Self} = L_{Self} / (L_{Self} \cap L_{Sibling})$$

- Les labels qui sont en commun entre un fils et son père sont supprimés du parent:

$$L_{Parent} = L_{Parent} / ((L_{Self} \cap L_{Parent}) / L_{Sibling})$$

Ce traitement est appliqué sur les k meilleurs labels de chaque cluster.

Un problème de clusters sans label peut apparaître après ce premier post-traitement pour une valeur de k petite. A l'opposé les labels d'un cluster peuvent être vide de sens lorsque que k est trop grand. Pour résoudre ce problème nous proposons de limiter les clusters à traiter à un sous-ensemble de cluster dans le cas où des labels sans signification apparaissent (mesure du score):

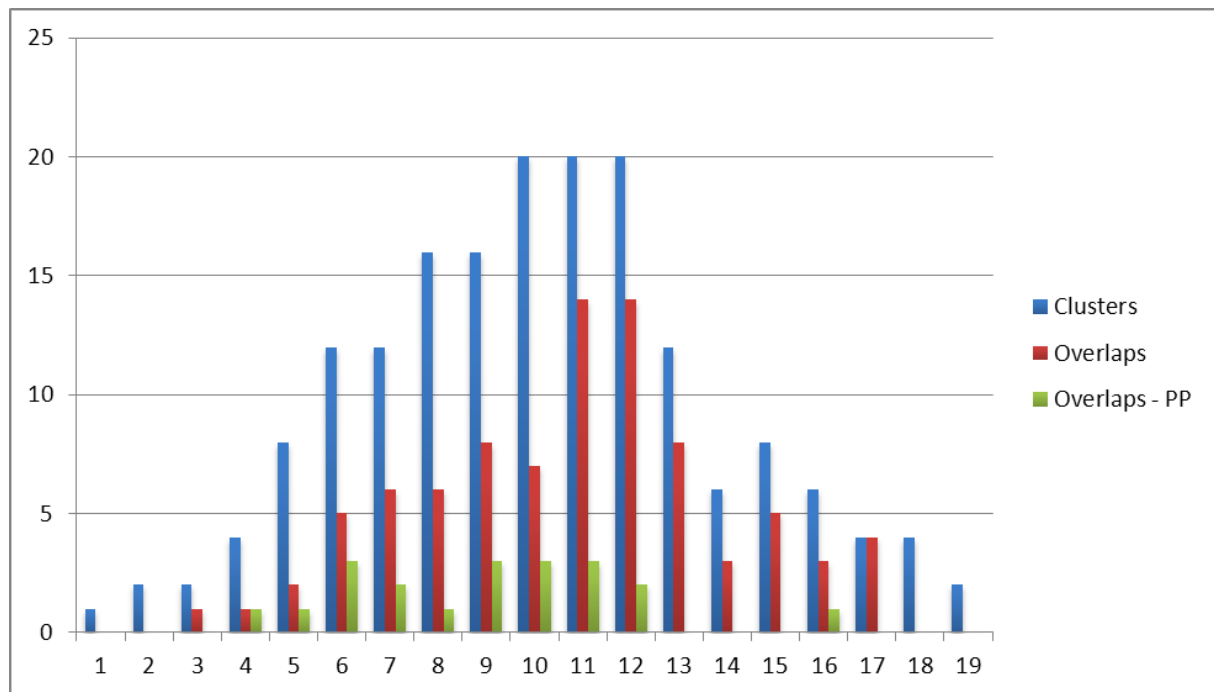
$$Lm_{Cluster} = Sublist(L_{Cluster}, K)$$

Ainsi le post traitement devient:

$$L_{Parent} = L_{Parent} \cup (Lm_{Self} \cap Lm_{Sibling})$$

$$L_{Self} = L_{Self} / (Lm_{Self} \cap Lm_{Sibling})$$

$$L_{Parent} = L_{Parent} / ((Lm_{Self} \cap Lm_{Parent}) / Lm_{Sibling})$$



Le graphique ci-dessous montre une diminution sensible des recouvrements de labels en tre clusters, cependant une mesure externe de la qualité de ces labels (par un simple constat humain) laisse apparaître un niveau sémantique assez bas, rendant la taxonomie peut utilisable.

Nous verrons, dans la section 3.6.2, que ce résultat dépend en fait beaucoup des données en entrée.

3.6.1.3 Approche basée sur une variante de TF-IDF

Le problem inherent au scoring issu de la fonction JSD est relative aux documents ressources eux-mêmes qui contiennent souvent les mêmes phrases. Il n'y a pas suffisamment de séparation entre les corpus de chaque cluster. Les termes les plus discriminants et portant aussi le plus de sémantique sont les termes techniques décrivant les dispositifs et les services. L'utilisation des noms de méthodes et le nom des variables d'états semble une manière pertinente d'améliorer la séparation entre les clusters.

Nous proposons donc une nouvelle approche qui reprend la méthode générale de labélisation d'un cluster hiérarchique mais en modifiant l'étape de construction du corpus. La construction de la hiérarchie augmentée est obtenue en récupérant tous les noms de fonctions et de variables d'états des dispositifs, puis en les associant aux différents clusters.

Le scoring de chaque phrase utilise une variante de TF-IDF en lieu et place de JSD:

- Discrimination des frères (SD): cette mesure favorise les phrase/termes qui ont une fréquence assez basse dans le cluster frère

$$SD = \text{freq}(L, \text{self}) - \text{freq}(L, \text{sibling})$$

- Discrimination du parent (PD): cette mesure favorise les phrases qui représentent bien les clusters fils

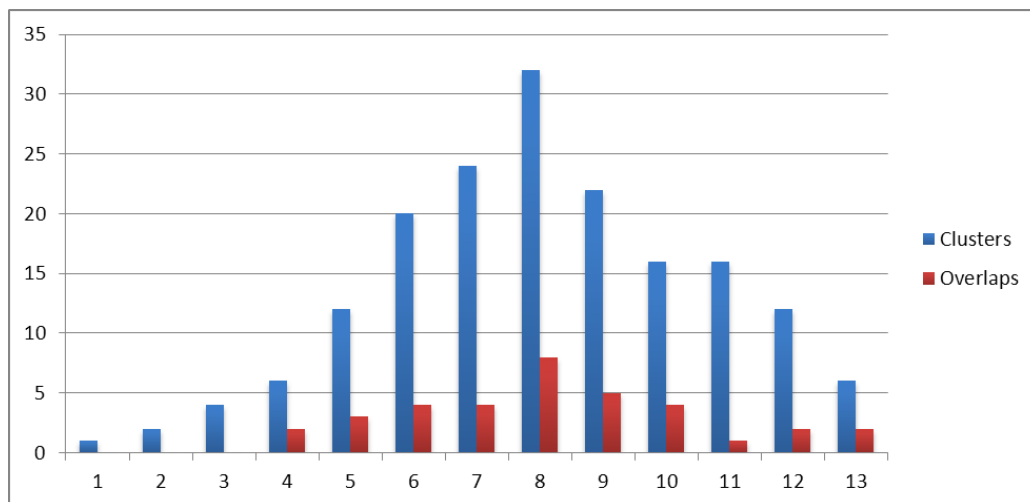
$$PD = \text{freq}(L, \text{child1}) + \text{freq}(L, \text{child2}) - |\text{freq}(L, \text{child1}) - \text{freq}(L, \text{child2})|$$

Le score finale est la somme de ces deux mesures:

$$\text{Score}(L) = SD + PD$$

Les phrases sont encore triées en fonction de leur score afin d'obtenir les k meilleurs labels.

Cette approche fait apparaître un gain très appréciable en qualité de labélisation des clusters: la sémantique des labels fait sens et les recouvrements de labels entre clusters disparaissent Presque, mais pas complètement. Une phase de post-traitement peut résoudre définitivement ces problèmes résiduels sans rendre la taxonomie inutilisable.



La Figure 14 donne un extrait de la taxonomie produite à partir de la description des dispositifs standards du Forum UPnP en suivant notre approche.

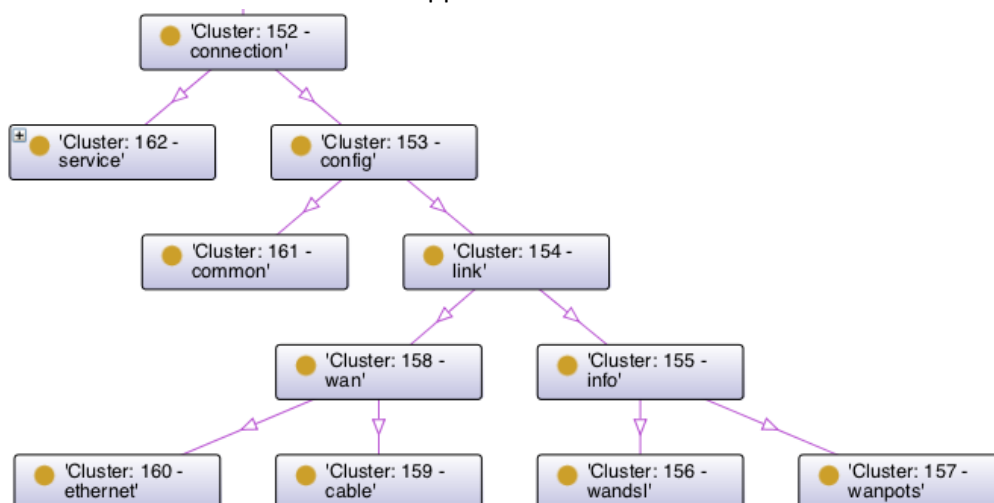


Figure 14: Extrait de la taxonomie produite

3.7 Conclusion

Nous avons apporté dans ce chapitre une réponse à un problème clé à l'utilisation d'une base de connaissance: la construction semi-automatique d'une ontologie de base. La qualité de l'ontologie de départ est un facteur important dans la résolution des problèmes d'hétérogénéité, dans l'expressivité des requêtes d'interrogation, et dans les possibilités d'évolution de cette ontologie. Notre approche repose sur une technique de clusterin hiérarchique compléter par une méthode de labélisation pour construire une taxonomie à partir de la description UPnP des dispositifs du domaine d'applications. Les résultats obtenus ont permis de valider plusieurs ontologies bootstrap dans Continuum.

4 Conclusion

Contributions

La contribution majeure consiste en une méthode de construction automatique d'ontologie, non supervisée. Les différentes infrastructures existantes proposent toutes un modèle statique ou très peu évolutif qui implique une étape complexe pour spécifier l'ontologie. Au mieux une ontologie métier est imposée en se limitant à un domaine d'application.

Notre approche s'attache à proposer un modèle d'ontologie déclaratif et facilement extensible. Cependant afin de simplifier l'enrichissement d'une ontologie, nous avons besoin d'une ontologie de bootstrap bien construite. Notre méthode permet de construire une ontologie à partir des spécifications des dispositifs et services types de l'application visée. Une technique de classification basée sur le clustering hiérarchique, couplée à une méthode de nommage des clusters générés, permet d'obtenir une ontologie correcte.

Nous avons, dans la mesure du possible, évalué la qualité de l'ontologie produite, et expérimenté sur quelques scénarios définis dans le projet Continuum.

Perspectives

L'axe principal pour améliorer la gestion de l'hétérogénéité consiste à dégrader les résultats d'une requête à la base de connaissance. Une requête peut ne pas obtenir de résultats dans un certain contexte d'exécution, mais des résultats proche mais non exacts pourrait tout à fait être suffisants pour la réalisation d'une tâche. Il s'agit donc d'attaquer le problème de relaxation de requêtes. L'effet inverse est également à considérer lorsqu'une requête retourne trop de réponses.

Dans cette perspective la mesure de la qualité des résultats devient cruciale. L'approche retenue pour construire une ontologie (clustering hiérarchique) fournit de manière intrinsèque une information de distance entre clusters particulièrement utile pour consolider cette évaluation de la qualité d'un résultat (dispositifs ou services découverts).

5 Références bibliographiques

- [And04] Periklis Andritsos. Scalable Clustering of Categorical Data and Applications. PhD thesis, University of Toronto, 2004.
- [ARR+11] Serge Abiteboul, Marie-Christine Rousset, Philippe Rigaux, Pierre Senellart, and oana Manolescu. Web Data Management. Cambridge University Press, 2011.
- [BJR] Anis Benyelloul, Fabrice Jouanot, and Marie-Christine Rousset. Conquer: an rdfs-based model for context querying.
- [Can07] Laurent Candillier. Contextualisation, Visualisation et évaluation en Apprentissage Non Supervisé. PhD thesis, 2007.
- [CRF03] William W. Cohen, Pradeep D. Ravikumar, and Stephen E. Fienberg. A comparison of string distance metrics for name-matching tasks. In *IIWeb*, pages 73–78, 2003.
- [GRS00] Sudipto Guha, Rajeev Rastogi, and Kyuseok Shim. Rock: A robust clustering algorithm for categorical attributes. *Inf. Syst.*, 25(5):345–366, 2000.
- [GRS01] Sudipto Guha, Rajeev Rastogi, and Kyuseok Shim. Cure: An efficient clustering algorithm for large databases. *Inf. Syst.*, 26(1):35–58, 2001.
- [HU08] Carlos A. Hurtado and Alexandra Poulouvasilis and Peter T. Wood, Query Relaxation in RDF. *Journal on Data Semantics, LNCS 4900*, pages 31–61, 2008.
- [JD88] Anil K. Jain and Richard C. Dubes. Algorithms for Clustering Data. Prentice-Hall, 1988.
- [KHK99] George Karypis, Eui-Hong Han, and Vipin Kumar. Chameleon: Hierarchical clustering using dynamic modeling. *IEEE Computer*, 32(8):68–75, 1999.
- [MU10] Markus Muhr and Roman kern and Michael Granitzer, Analysis of Structural Relationships for Hierarchical Cluster Labeling. *SIGIR'10 Proceedings of the 33rd international ACM SIGIR conference on Research and development in information retrieval*, pages 178-185, 2010.
- [PU06] Pucktada Treeratpituk and Jamie Callan, Automatically Labeling Hierarchical Clusters. *Proccedings of dgo'06*, 2006
- [Sai07] Fatiha Sais. Intégration Sémantique de Données guidée par une Ontologie. PhD thesis, Universitéw Paris-Sud, 2007.
- [VFLD06] Anne-Marie Vercoastre, Mounir Fegas, Yves Lechevallier, and Thierry Despeyroux. Classification de documents xml à partir d'une représentation linéaire des arbres de ces documents. In *EGC*, pages 433–444, 2006.
- [XI05] Rui Xu and Donald C. Wunsch II. Survey of clustering algorithms. 2005.
- [ZK04] Ying Zhao and George Karypis. Empirical and theoretical comparisons of selected criterion functions for document clustering. *Machine Learning*, 55(3):311–331, 2004.