Apprendre des actions utilisateur pour guider le processus de configuration dans les lignes de produits logiciels

François Chastel, Mireille Blay-Fornarino, Célia da Costa Pereira, Simon Urli

LABORATOIRE

INFORMATIQUE, SIGNAUX ET SYSTÈMES DE SOPHIA ANTIPOLIS

UMR7271

Apprendre des actions utilisateur pour guider le processus de configuration dans les lignes de produits logiciels

François Chastel, Mireille Blay-Fornarino, Célia da Costa Pereira, Simon Urli

EQUIPE **SPARKS**

Rapport de Recherche

octobre-2015

Laboratoire d'Informatique, Signaux et Systèmes de Sophia-Antipolis (I3S) - UMR7271 - UNS CNRS
2000, route des Lucioles - Les Algorithmes - bât. Euclide B 06900 Sophia Antipolis – France
http://www.i3s.unice.fr

Membre de UNIVERSITÉ CÔTE D'AZUR

Learning from User Actions for Driving the Configuration Process in Software Product Lines

François Chastel, Mireille Blay-Fornarino, Célia da Costa Pereira, Simon Urli

Equipe SPARKS

octobre-2015- 26 pages

Abstract: In software product lines, the choices the ''user'' has to make to configure a product may be very numerous. According to the user's knowledge of the line as well as her own vision of the target product, some choices may be difficult and may turn out to be wrong. However, the configuration processes usually show similarities in the sequence of actions, at least over sub-parts of the process. In the context of a product line for the production of ''information diffusion systems'', configuration traces are available in terms of user actions and we intend to learn from these traces to help the user in her choices. This report presents the first results achieved.

Key-words: learning, sofware product line, user actions

Apprendre des actions utilisateur pour guider le processus de configuration dans les lignes de produits logiciels

Résumé : Dans les lignes de produits logiciels les choix que doit faire l' ''utilisateur'' pour configurer un produit peuvent être très nombreux. En fonction de la connaissance de l'utilisateur dans la ligne et également de sa propre vision du produit visé, certains choix peuvent être difficiles et s'avérer erronés. Cependant, les processus de configuration présentent *a priori* des similarités dans l'enchaînement des actions, au moins sur des sous-parties de processus. Dans le contexte d'une ligne de produits pour la production de ''systèmes de diffusion d'informations'', nous disposons des traces de configurations en terme d'actions utilisateur et nous ambitionnons d'apprendre de ces traces pour aider l'utilisateur dans ses choix. Ce rapport présente les premiers résultats obtenus.

Mots-clefs : apprentissage, ligne de produits logiciels, actions utilisateur.

Apprendre des actions utilisateur pour guider le processus de configuration dans les lignes de produits logiciels

François Chastel [1], **Mireille Blay-Fornarino** [1], **Célia da Costa Pereira** [1], **Simon Urli** [2]

1. *Univ. Nice Sophia Antipolis, CNRS, I3S, UMR 7271, 06900 Sophia Antipolis, France*
   prénom.nom@unice.fr
2. *The 6th Screen - Business Pole, 1047 route des dolines, allée Pierre Ziller, 06560 Sophia Antipolis, France*
   prénom.nom@unice.fr

RÉSUMÉ. *Dans les lignes de produits logiciels les choix que doit faire l'"utilisateur" pour configurer un produit peuvent être très nombreux. En fonction de la connaissance de l'utilisateur dans la ligne et également de sa propre vision du produit visé, certains choix peuvent être difficiles et s'avérer erronés. Cependant, les processus de configuration présentent* a priori *des similarités dans l'enchaînement des actions, au moins sur des sous-parties de processus. Dans le contexte d'une ligne de produits pour la production de "systèmes de diffusion d'informations", nous disposons des traces de configurations en terme d'actions utilisateur et nous ambitionnons d'apprendre de ces traces pour aider l'utilisateur dans ses choix. Ce rapport présente les premiers résultats obtenus.*

ABSTRACT. *In software product lines, the choices the "user" has to make to configure a product may be very numerous. According to the user's knowledge of the line as well as her own vision of the target product, some choices may be difficult and may turn out to be wrong. However, the configuration processes usually show similarities in the sequence of actions, at least over sub-parts of the process. In the context of a product line for the production of "information diffusion systems", configuration traces are available in terms of user actions and we intend to learn from these traces to help the user in her choices. This report presents the first results achieved.*

MOTS-CLÉS : *Apprentissage, ligne de produits logiciels, actions utilisateur.*

KEYWORDS: *Learning, Sofware product line, user actions.*

1. Introduction

Une ligne de produits logiciels (LPL) vise à modéliser la variabilité d'une famille logicielle afin de permettre la réutilisation d'éléments logiciels communs. Elle peut ainsi aider la production automatique de systèmes logiciels similaires par la "simple sélection" des fonctionnalités attendues (Pohl *et al.*, 2005). Les bénéfices incluent alors une réduction des coûts de développement, une amélioration de la qualité des logiciels produits, une plus grande rapidité de production. Les *feature models* (FMs) sont largement utilisés pour modéliser la variabilité des systèmes en terme de "features" exigés, optionnels ou exclusifs, de même que les contraintes entre les features (Schobbens *et al.*, 2007). Ils peuvent consister en plus d'un millier de features avec des dépendances complexes entre eux (Passos *et al.*, 2012).
La sélection et déselection de features dans un FM permet de construire des configurations à partir desquelles les logiciels sont produits. Le FM réduit la configuration des features aux seuls combinaisons pour lesquels des produits peuvent être construits. Nous parlons alors de configurations valides. *Un processus de configuration* est un processus interactif et incrémental d'actions utilisateur avec l'objectif de fournir des configurations valides qui correspondent aux exigences de l'utilisateur. La structure du FM peut aider à guider l'utilisateur dans ses choix (Urli *et al.*, 2012), de même que la mise en place d'outils de configuration dédiés (Bosch, Högström, 2001 ; Boucher *et al.*, 2012) mais la grande variabilité des produits à construire rend le processus de configuration long et complexe.

Dans le cadre du projet ANR Emergence YourCast [1], nous avons en particulier développé une ligne de systèmes de diffusions d'informations qui en moyenne exige plus d'une centaine de choix utilisateur, pour 2500 choix automatisés (Urli, Blay-fornarino, Collet, Mosser, Riveill, 2014). Cette ligne en production aujourd'hui a permis à ce jour la production de plus d'une quinzaine d'écrans pour lesquelles nous disposons des traces d'interactions avec les utilisateurs devant construire des configurations. A destination du grand public, elle a été testée sur différents utilisateurs non informaticiens avec l'aide d'ergonomes. Le besoin d'une aide plus appuyée dans la sélection de certaines caractéristiques en même temps que la similarité ressentie de certains comportements utilisateurs nous conduisent à nous intéresser aujourd'hui à l'apprentissage sur la base des traces obtenues. Notre objectif est de guider l'utilisateur sur la base de l'apprentissage du comportement d'autres utilisateurs, avec l'hypothèse vraie dans notre étude que ces traces correspondent à des configurations valides, même si incomplétes (Urli, 2015)

Après un bref positionnement relativement à l'état de l'art (cf. SEC. 2), nous présentons la démarche générale choisie (cf. SEC. 3). Nous montrons comment nous l'avons appliqué à notre étude de cas et les premiers résultats obtenus en section 4. Nous concluons ce rapport par une discussion sur les limites de ce travail et les perspectives envisagées (cf. SEC. 5).

1. http://www.yourcast.fr/

2. Autres travaux

Dans le cadre des lignes de produits logiciels, dès lors qu'elles offrent un grand nombre de produits et tout particulièrement lorsque ceux-ci sont configurables (configurations dans le cloud par exemple (Quinton *et al.*, 2014)) il est difficile et long de produire les bonnes configurations. Une solution est d'assister l'utilisateur (i.e. la personne qui configure un produit dans la ligne) au travers d'un processus interactif qui réduit ses choix à chaque pas (Urli, Blay-fornarino, Collet, 2014). Cependant, parmi les choix restants il n'a pas d'autres indications que les informations associées aux features présentés.
En exploitant les systèmes de contraintes sous-jacents à la ligne de produits, il est possible de guider ces choix en utilisant des heuristiques comme la réduction du nombre d'actions utilisateur ou la minimisation d'impact sur les choix à venir (Mazo, Dumitrescu, 2014). Nous nous situons dans ce rapport dans une démarche différente basée sur l'hypothèse qu'il vaut mieux guider l'utilisateur dans des choix qu'il comprend même s'ils réduisent peu le nombre d'actions à faire, que des choix optimaux dont il ne comprend pas la pertinence à cette étape de la configuration.
Pour faire face à la complexité de configurations et également intégrer des processus de configurations collaboratifs, les notions de vues et de workflows de configurations entre ces vues ont été définies(Abbasi *et al.*, 2011 ; Hubaux *et al.*, 2009). Le choix même de ces workflows reste difficile (Urli, 2015). Nous nous situons dans l'étude présente en amont. Les vues et les workflows de configuration devraient à terme pouvoir être suggérés à partir des apprentissages réalisés sur des lignes non contraintes. De même, pour faciliter les prises de décisions dans les étapes de configurations des langages de modélisation différents des feature models ont été définis tels que par exemple DoplerVML qui est orienté décision (Dhungana *et al.*, 2010). Ils n'abordent cependant pas la problématique de la prise de décision elle-même. Enfin notre approche est plus large qu'une configuration dans un featurem model et nous permet d'aborder les cas où les systèmes à construire sont eux-même composés de plusieurs sous-systèmes. L'utilisateur doit alors construire des *configurations composites* au sens où elles sont un assemblage de sous-configurations dans des FM différents (Urli, Blay-fornarino, Collet, 2014).

Robillard et al. identifient parmi les différentes sources d'informations relatives à un projet, le code source lui-même et les traces des interactions avec l'utilisateur (Robillard *et al.*, 2010, Chapitre 1). Nous nous intéressons à ces dernières, pour suggérer à l'utilisateur les prochaines actions à réaliser dans un processus de configuration. Par la suite nous parlons simplement de traces utilisateur. En cela nous nous rapprochons des recommandations dans le cadre du génie logiciel, dont la définition est redonné dans cet ouvrage : . . . *a software application that provides information items estimated to be valuable for a software engineering task in a given context.* La tâche qui nous intéresse est la configuration d'une ligne et le contexte est déterminé sur la base des actions passées.

Le raisonnement basé sur les traces est une approche de l'IA dans laquelle les inférences sont réalisées sur des objets spécifiques, dits *traces*, qui mémorisent temporellement des évènements pendant un processus d'interaction (Cordier *et al.*, 2013). L'apprentissage doit alors prendre en compte la séquentialité des évènements. L'identification de "signature d'épisode" vise alors à identifier dans les traces des tâches récurrentes. Cependant

alors que les actions sont différentiantes par leur nature dans les travaux utilisant ces techniques (creer une page, éditer une page, etc...) dans le contexte des lignes de produits, les actions sont essentiellement différenciées par les features sélectionnés et des ordres différents peuvent conduire au même résultat. Nous ne savons pas a priori comment caractériser les épisodes à rechercher. Cependant nous cherchons bien à retrouver des situations "ressemblantes " dans le passé qui peuvent aider l'utilisateur à accomplir sa tâche, ce qui nous conduit à nous intéresser également aux mesures de similarité. Une fois des situations similaires identifiées, elles peuvent être utilisées pour automatiser les actions suivantes en les "rejouant" ou simplement comme un guide. Nous nous situons plus spécifiquement à ce niveau.

3. Vers un apprentissage basé sur les traces utilisateur pour l'aide aux configurations dans les LPL

3.1. Traces utilisateur

Pour pouvoir apprendre des traces issues de différentes lignes nous avons formalisé les traces selon le métamodèle suivant, présenté figure 1.

Une *Trace* est un ensemble ordonné de *Step*s. Un *Step* correspond à une étape de l'interaction qui a été délibérément choisie pour être observée (obsel dans (Cordier *et al.*, 2013)). A un *Step* est toujours associée une *Action* caractérisée par un ensemble d'*Argument*s (ensemble de propriétés (paire clef-valeur)) et un nom de type.

Formellement, une *Trace* t est une suite ordonnée de *Step*s : $t = \langle s_1 \dots s_n \rangle$ et nous notons $s_i \prec_t s_j$ pour $i < j$ qui signifie que l'action réalisée au pas i a été faite avant celle du pas j.

Nous le complétons en introduisant les fonctions suivantes :

– deux actions sont en contradiction si elles ne peuvent pas apparaitre dans une même trace : l'exécution de l'une rend impossible l'exécution de l'autre,

– la similarité de deux actions dépend de l'étude réalisée, dans notre cas nous considérons que deux actions de sélection sur un même FM sont "similaires" si elles portent sur le même feature.

Notons que les traces utilisateur se différencient des configurations elles-mêmes. Plusieurs traces d'actions utilisateur peuvent ainsi conduire à la même configuration. La figure 2 montre un simple FM qui présente deux features exclusives *Twitter* et *Picasa*. Choisir l'une interdit de choisir l'autre. De plus, la sélection d'une feuille d'un FM implique de choisir la feature parente ici *Product*[2]. En conséquence, comme on peut le voir sur la figure 2, les 4 extraits de traces présentées à droite de la figure conduisent à définir la même configuration du FM. Cependant, nous considérons actuellement que ces configurations ne

2. Un FM est un arbre, un feature a donc au plus un feature parent, la sélection est alors inférée.

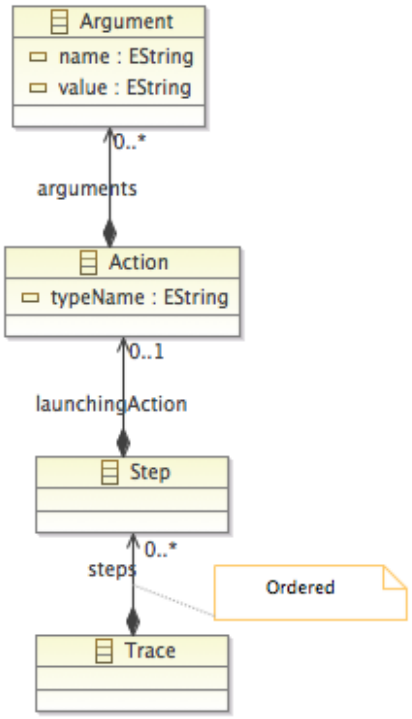

Figure 1. Metamodèle simplifié des traces

sont pas réalisées avec le même objectif, la dé-sélection d'un feature exprimant un "rejet" d'une feature, tandis que sa sélection exprime un "choix". C'est pourquoi dans la suite nous cherchons à apprendre des traces utilisateur et non pas des configurations elles-mêmes.

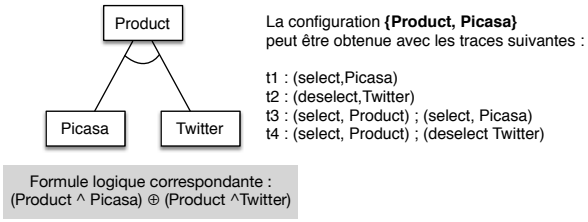

Figure 2. Relations entre FM et traces utilisateur simplifiées

3.2. Vision globale de l'approche

La figure 3 présente la vision globale des enchaînements de tâches pour atteindre le guidage dans la configuration d'une LPL. Nous avons schématisé dans cette figure les deux processus, l'apprentissage qui construit des clusters et la configuration qui utilise les résultats du processus précédent. Il peut à son tour produire de nouvelles traces qui pourront être utilisés dans l'exécution d'un processus d'apprentissage.

La figure 4 décrit les objets utilisés par ce processus.

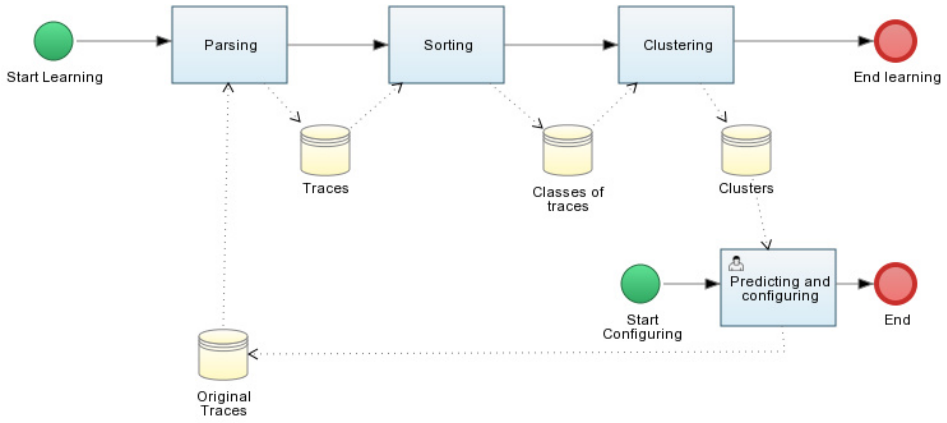

Figure 3. Processus d'apprentissage

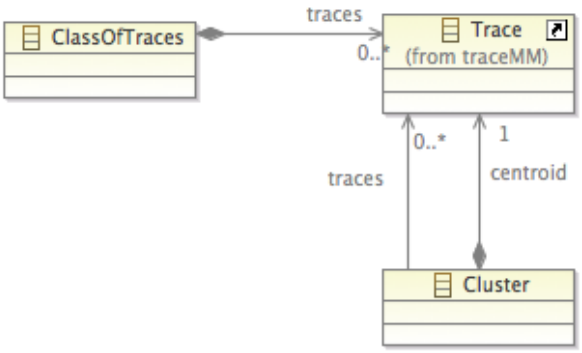

Figure 4. Metamodèle utilisé par le raisonnement

La première tâche de *parsing* permet d'analyser un ensemble de traces données dans un format propre et de les ramener au format présenté dans la figure 1. Nous ne la présentons pas davantage, elle dépend des données en entrée.
La seconde tâche, *Sorting*, permet de ne garder dans les traces que les *Step* jugés pertinents

et de les regrouper sous la forme de classes de traces (*ClassOfTraces*) (cf. SEC. 3.3).
La tâche suivante, dites de *clustering*, consiste en l'apprentissage proprement dit (cf. SEC. 3.4) et produit l'ensemble des *Cluster*. Ceux-ci sont utilisés pour guider les utilisateurs dans leur tâche de configuration (cf. SEC. 3.5).

3.3. Tri

La tâche de tri dépend fortement de la ligne de produits, des actions offertes à l'utilisateur et des types de séquences recherchés. En effet, si l'on se situe dans une seule ligne basée sur un FM "classique"(Kang *et al.*, 1990), les principales actions sont *sélectionner* et *désélectionner* un feature. Lorsque le FM est attribué (Benavides *et al.*, 2010), on va trouver des actions de valuation des attributs. Dans les cas plus complexes, comme la ligne de produits étudiée en section 4, les actions peuvent être la création de configuration et intégrer les notions de sous-systèmes(Urli, Blay-fornarino, Collet, 2014). De plus, il peut être intéressant non pas d'analyser les actions de sélection directement mais de s'intéresser davantage à la hiérarchie de sélection (e.g. ce n'est pas tant le fait que l'utilisateur choisisse la couleur bleu pour sa voiture qui importe que le fait qu'il ait commencé par choisir la couleur de la voiture). Cette tâche de tri vise donc à d'une part éliminer les actions non pertinentes et à regrouper les *Step* en fonction de différents critères.

La fonction de tri a donc pour rôle, à partir d'un ensemble de traces $T = \{t_1...t_n\}$, de produire un ensemble de classes de traces $\{c^1 \dots c^p\}$ telles que les ordres sur les "steps" soient respectés de même que leur appartenance à une même trace. Nous formalisons cette propriété comme suit :
$\forall t^i \in c^i, \forall (s_1, s_2) \in t^i, s_1 \prec_{t^i} s_2, \Rightarrow \exists t \in T, (s_1, s_2) \in t, s_1 \prec_t s_2,$

Cette propriété permet de garantir que les ordres sont préservés et que l'on ne mélange pas les actions utilisateurs issues de différentes traces. Cette propriété est importante parce que les clusters sont calculer au sein des classes calculées à cette étape.

3.4. Clustering

Le "Clustering", (Xu, Wunsch, 2005 ; Chakrabarti *et al.*, 2006), est une méthode d'apprentissage non supervisé dont le but est de construire, à partir d'un ensemble de données non labellisées, des groupes ayant les caractéristiques suivantes :

– les données appartenant à un même groupe ("cluster") sont similaires – homogénéité intra-groupes ;

– les données appartenant à des groupes ("clusters") différents diffèrent – hétégéonérité inter-groupes.

Formellement, si l'on considère un ensemble de données I, de cardinalité n et l'ensemble de ses k partitions (regroupements), $C = \{C_1, \dots, C_k\}$, avec $k \leq n$, nous avons :

1. $C_i \neq \emptyset, \forall i \in \{1, \dots, k\}$: il n'y a pas de "cluster" vide ;
2. $\bigcup_{i=1}^{k} C_i = I$: l'union de tous les "clusters" constitue l'ensemble de données;

3. $C_i \cap C_j = \emptyset, i \neq j$: l'intersection entre deux "clusters" différents est vide.

Chaque "cluster", a, en général, un élément qui le représente – son *centroïde*. Si le i-ème élément du cluster C_j, de cardinalité z, est une trace de valeurs numériques $\langle x^j_{i1}, x^j_{i2}, \ldots, x^j_{ik} \rangle$ son centroïde, $\mathbf{c}^j$, est une trace définie comme suit :

$$\mathbf{c}^j = \left\langle \frac{1}{z} \cdot \sum_{i=1}^{z} x^j_{i1}, \frac{1}{z} \cdot \sum_{i=1}^{z} x^j_{i2}, \ldots, \frac{1}{z} \cdot \sum_{i=1}^{z} x^j_{ik} \right\rangle \quad (1)$$

Lorsque l'élément *représentant* le cluster est l'un de ses éléments, il est appelé *medoïde*. Il existe plusieurs algorithmes de clustering dans la littérature (Xu, Wunsch, 2005). Pour ce travail, nous avons opté pour le choix de l'algorithme des $k-$moyennes (cf. algo 3.4).

Algorithm 1 Algorithme des $k-$moyennes

Require: Ensemble de données I, nombre k de centroïdes
Ensure: Partition C avec k regroupements de données, $C_1, \ldots, C_k$

Initialization : choisir aléatoirement dans I, k centroïdes.
Mettre à jour les k clusters correspondants en affectant chaque donnée au cluster dont le centroïde est le plus proche (similaire) de celle-ci.
Recalculer les centroïdes pour chacun des clusters.
Repéter la deuxième et la troisième étapes jusqu'à ce qu'il y ait convergence, i.e., jusqu'à ce qu'il n'y ait plus de changement dans les clusters.

3.4.1. Algorithme des $k-$moyennes adapté

Dans le contexte des LPL, nous n'avons pas su décider a priori du nombre de clusters, le nombre de features ou de configurations s'étant rapidement révélé non pertinent. Nous avons donc choisi d'adapter cet algorithme en nous basant sur la fonction de calcul de la distance. Nous créons un nouveau cluster chaque fois que nous trouvons une trace dont la similarité est de 0 relativement aux centroïdes déjà identifiés.

3.4.2. Similarité entre traces

Comme mentionné précédemment, l'objectif d'un algorithme de clustering est de construire une collection d'éléments similaires lorsqu'ils sont au sein d'un même groupe et dissimilaires quand ils appartiennent à des groupes différents. Les données étant des traces, nous avons besoin de mesurer la similarité et la distance entre les traces. Ces mesures doivent tenir compte (i) du nombre d'éléments dans les traces, (ii) de la similarité des actions ayant le même ordonnancement, (iii) de l'ordre des actions similaires ayant un ordonnancement différent, et (iv) de la contradiction entre les actions. Rappellons, qu'une trace t peut être représentée comme suit :

$$t = \langle s_1, s_2, \ldots, s_e \rangle, \quad (2)$$

Algorithm 2 Algorithme adapté des $k-$moyennes

Require: Ensemble de données I,
Ensure: Partition C avec k regroupements de données, $C_1, \ldots, C_k$ et la distance entre deux éléments d'une partition n'est jamais égale à 0.

Initialization : choisir aléatoirement dans I, un medoïde; k vaut 1
1- *Mettre à jour* les k clusters correspondants en affectant chaque donnée au cluster dont le centroïde est le plus proche (similaire) de celle-ci. Si la donnée ne peut pas être affectée à un cluster (distance de 0 avec tous les cluster présents), un nouveau cluster est créé avec pour "centroïde" cette donnée et k est incrémenté de 1.
2-*Recalculer* les centroïdes pour chacun des clusters.
3- *Répéter* la première et la deuxième étapes jusqu'à ce qu'il y ait convergence, i.e., jusqu'à ce qu'il n'y ait plus de changement dans les clusters.

où, chaque s_i représente un pas et donc une action; par construction, $s_i \neq s_j$ si $i \neq j$; e est le nombre d'éléments dans la trace [3].

Nous définissons le dégré de similarité sim entre deux traces t et t' comme suit :

$$\mathrm{sim}(t, t') = \frac{1}{N_{max}} \cdot \frac{N_c + (1 - \alpha) \cdot N_{\bar{c}}}{\beta^{N_o}}, \tag{3}$$

où, N_{max} correspond à la dimension de la trace la plus longue ; N_c correspond au nombre d'éléments qui sont en commun et dans la même position dans les traces t et t' ; $N_{\bar{c}}$ correspond au nombre d'éléments qui sont en commun mais dans des positions différentes dans les traces t et t' ; N_o correspond au nombre d'actions contradictoires; $\alpha \in [0, 1]$ est le facteur qui modélise l'influence que nous aimerions donner au fait que les éléments communs ne soient pas dans la même position ; et enfin, $\beta > 1$ est le facteur qui modélise l'influence que l'ont veut donner à la présence de contradiction entre les traces. Les "bonnes" valeurs pour les paramètres α et β seront estimées de façon empirique.

3.4.3. Quelques propriétés de la fonction sim

La fonction sim a les propriétés suivantes :

- sa valeur est comprise entre 0 et 1,
- sa valeur est maximale (égale à 1) si :
  - les deux traces ont la même longueur, et,
  - tous leurs éléments sont communs et dans la même position, et,
  - il n'existe pas de contradiction.
- sa valeur est minimale (égale à 0) si :
  - il n'y a aucun élément commun entre les traces, ou,

3. Les traces peuvent être de dimensions différentes.

- le nombre d'éléments contradictoires est maximal, i.e., il y a autant d'éléments contradictoires que le nombre d'éléments dans la trace de dimension minimale.

Nous pouvons donc calculer la distance d entre les traces comme suit [4] :

$$\mathrm{d}(t, t') = \frac{1}{\mathrm{sim}(t, t')}, \tag{4}$$

La figure 5 donne des exemples de calculs de similarité.

t1 : <(select f1); (select f2)>
t2 : <(select f1); (deselect f2)>
t3 : <(select f1); (select f3); (deselect f2)>
t4 : <(select f1); (select f3); (select f2)>
t5 : <(select f1); (select, f3)>

alpha = 0,25
beta = 2

	t1					t2					t3					t4				
	Nmax	Nc	N/c	No	sim	Nmax	Nc	N/c	No	sim	Nmax	Nc	N/c	No	sim	Nmax	Nc	N/c	No	sim
t1	2	2	0	0	1,00	2	1	0	1	0,25	3	1	0	1	0,17	3	1	1	0	0,58
t2	2	1	0	1	0,25	2	2	0	0	1,00	3	1	1	0	0,58	3	1	0	1	0,17
t3	3	1	0	1	0,17	3	1	1	0	0,58	3	3	0	0	1,00	3	2	0	1	0,33
t4	3	1	1	0	0,58	3	1	0	1	0,17	3	2	0	1	0,33	3	3	0	0	1,00
t5	2	1	0	0	0,50	2	1	0	0	0,50	3	2	0	0	0,67	3	2	0	0	0,67

Soit le cluster contenant {t2,t3,t5}, le centroïde sera :
<(select f1); (select f3>;(deselect f2)>
Soit le cluster contenant {t1,t4}, le centroïde sera :
<(select f1);(select f2); (select f3)>

Figure 5. Exemples de calculs de similarité

En nous basant sur la mesure de similarité définie par les auteurs de (Zarka *et al.*, 2013), la mesure que nous proposons pourrait être enrichie pour prendre en compte les types d'actions, en particulier en utilisant les hiérarchies dans les FM, et les types d'utilisateurs ayant défini les traces [5].

3.4.4. Calcul du centroïde

Nos données n'étant pas des traces de valeurs numériques, la définition standard de centroïde présentée avec l'équation 1 n'est pas adaptée à notre problème. De même les traces étant de longueurs différentes et ces différences pouvant s'expliquer par des choix

4. Il est possible d'utiliser d'autres fonctions de distance.
5. Information dont nous ne disposons pas encore

déjà opérés, un medoïde ne nous semble pas bien caractériser un cluster. Pour ces raisons, nous avons défini le centroïde, $\mathbf{c^j}$, d'un cluster C_j comme suit [6] :

$$\mathbf{c}^j = \left\langle \text{mode}\{x^j_{i1}\}_{i=1,\ldots,z}, \text{mode}\{x^j_{i2}\}_{i=1,\ldots,z}, \ldots, \text{mode}\{x^j_{ik}\}_{i=1,\ldots,z} \right\rangle \quad (5)$$

and $\forall (a_l, a_m) \in \mathbf{c}^j, i \neq j, \Rightarrow$
- $a_l \neq a_m$,, la même action n'apparaît pas deux fois,
- a_l n'est pas contradictoire avec a_m,
- si $a_l = null, l < m \Rightarrow a_m = null$

La figure 5 donne des exemples de calculs de centroïde.

3.5. Guidage

Notre objectif est à présent de suggérer à l'utilisateur la prochaine action à réaliser, lorsqu'il est en train de construire une configuration. Nous utilisons le résultat du clustering pour "reconnaître" la trace d'actions la plus proches des actions déjà réalisées et pouvoir ainsi lui suggérer les actions suivantes. Plusieurs techniques existent pour prédire les éléments dans une trace grâce aux éléments précédents, (Eban *et al.*, 2012). Cependant, en général, l'ordre des éléments dans les traces est très important. Dans notre cas, l'ordre a beaucoup moins d'importance et c'est davantage le nombre d'éléments en commun ou en contradiction qui est pertinent. C'est la raison pour laquelle nous allons procéder de la manière suivante.

Nous construisons la trace des actions réalisées puis par classification, en utilisant les mêmes mesures que pour la construction des clusters, nous déterminons le cluster le plus proche. Finalement, les informations de classification nous permettent d'obtenir une trace contenant un ensemble d'actions utilisateur : cette trace doit tout d'abord être filtrée afin d'éliminer les actions déjà réalisées par l'utilisateur, ainsi que les actions contradictoires avec celles déjà réalisées. Enfin, la première action de la trace ainsi épurée peut être suggérée à l'utilisateur.

4. Etude de cas sur la ligne de produits logiciels Yourcast

Dans le contexte de la ligne de produits logiciels Yourcast [9], nous avons relevé des traces utilisateur pendant plusieurs semaines et montrons ici l'application de la démarche présentée précédemment à ce cas d'étude. Nous commençons par préciser le contenu de la

6. Le mode (statistique) correspond à la valeur la plus représentée d'une variable quelconque dans un ensemble de données. Ici, en cas de parité on considère la valeur dominante parmi l'ensemble des valeurs restantes possibles.

7. Cette étape pourra être complétée en utilisant la logique pour également éliminer les actions qui sont des conséquences logiques des actions de l'utilisateur et sont alors automatisées.

8. La validation expérimentale de l'algorithme nous permettra de déterminer si en cas de réponse négative, il vaut mieux laisser l'utilisateur faire son propre choix et/ou lui proposer d'autres choix.

9. http://www.yourcast.fr/

Algorithm 3 Prédire à partir d'actions

Require: trace des actions utilisateur $\langle s_1, \ldots, \rangle$; ensemble des k centroïdes correspondant aux clusters obtenus par l'algorithme de clustering

Ensure: Proposition de l'action suivante à l'utilisateur

1: *Calculer* en utilisant l'Equation 4 les distances entre $\langle s_1, \ldots, \rangle$ et les centroïdes des k clusters,
2: *Sélectionner* le centroïde le moins distant;
3: *Réduire* la trace correspondant au centroïde en éliminant les actions déjà réalisées et celles contradictoires [7];
4: *Proposer* à l'utilisateur l'une des actions dans la trace ainsi obtenue;
5: *Considérer* la réponse de l'utilisateur. Si positive et que la configuration n'est pas terminée, proposer une autre action dans le centroïde, si négative passer à une autre trace la plus proche dans le même cluster ou dans un cluster proche [8].

ligne (cf. SEC. 4.1), puis explicitons comment la démarche a été appliquée à notre étude et les premiers résultats obtenus (cf. SEC. 4.2).

4.1. La ligne de produits "YourCast"

Les configurations auxquelles nous nous intéressons ont la particularité d'être complexes au sens où elles font intervenir plusieurs sous-configurations, conformément à un modèle du domaine. Les utilisateurs sont donc amenés à configurer simultanément plusieurs sous-systèmes dont la cardinalité n'est pas connue *a priori* et à les lier entre eux. Les choix faits sur un sous-système peuvent impacter les choix sur un autre sous-système.

La figure 6 illustre le modèle du domaine associé à la ligne de produits logiciels, YourCast, et donne un exemple simplifié de configuration composite. Il s'agit de construire des affichages dynamiques de diffusion d'informations en composant un `Layout`, des `Zones`, des `Sources` d'informations, des `Renderers` qui explicitent comment traiter les informations issues des sources et des `Behaviours` qui explicitent les transitions entre les rendus. Dans l'exemple donné, trois zones ont été créées dont une correspond à une zone vocale, la météo est affichée dans la zone d'entête et le menu du jour et du lendemain est affiché dans la zone principale tandis que le menu peut être lu par la synthèse vocale. Il n'y a pas d'ordre prédéfini et ce système de diffusion peut être obtenu aussi bien en sélectionnant toutes les sources, les zones, les renderers, qu'en créant une source et son renderer puis en l'associant à une zone.

4.1.1. Processus de configuration

Le processus de configuration est réalisé à travers une interface graphique de configuration couplée à un moteur de raisonnement afin de ne permettre à l'utilisateur que des choix valides d'après notre modélisation de la variabilité. Chaque concept du modèle du domaine possède, en effet, un FM qui représente sa variabilité. La table 1 donne les chiffres sur la variabilité de ces différentes FMs.

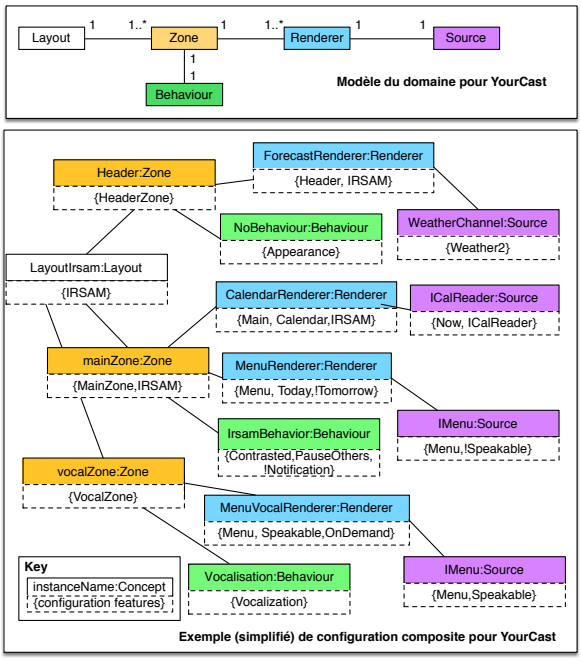

Figure 6. Ligne de systèmes de diffusion d'informations (YourCast) : Modèle du domaine et exemple de configuration composite

Tableau 1. Métriques sur les FM dans YourCast

Concept	*#Features*	*#Contraintes*	*#Configuration*
Sources	81	154	68
Renderers	76	347	74
Behaviour	33	45	15
Zones	49	160	27
Layouts	51	59	13
Moyenne	58	149	39
Total	290	765	–

En outre, ce processus de configuration est réalisé au travers d'un grand nombre d'actions utilisateur, qui peuvent être de différents types : sélection ou exclusion de feature, nommage d'éléments, validation des sous-configurations etc.

4.1.2. Traces utilisateur

Nous stockons l'ensemble ordonné des actions effectuées par l'utilisateur sous la forme d'une trace sérialisée au format XMI, telle que présentée dans la figure 7.

```
<history:Past>
  <step id="268c...">
    <launchingAction xsi:type="useraction:UserDeselect"
    domainElementName="Behaviour"  contextID="cl_c4b41f..."
    featureName="Notification"/>
  </step>
    <step id="37c1...">
    <launchingAction xsi:type="useraction:UserValidConfiguration"
    domainElementName="renderer" contextID="cl_661eeb..."/>
  </step>
   ...
   <step id="3cf6...">
    <launchingAction xsi:type="useraction:UserRenameElement"
    name="Calendrier_renderer" elementType="cps"
    elementID="cl_661eeb..._renderer"/>
  </step>
  <step id="e068...">
    <launchingAction xsi:type="useraction:UserLinkConfiguration"
    confSourceName="conf_cl_661eeb..._renderer"
    confTargetName="conf_cl_fe6728..._zone"
    assoName="AssoRendererZone"/>
  </step>
 </history:Past>
```

Figure 7. Extraits d'une trace d'actions utilisateur

4.2. Apprentissage dans le contexte de la LPL Yourcast

Après une étape de parsing des traces au format XMI sous la forme d'objets java respectant le métamodèle donné dans la figure 1, nous avons utilisé les algorithmes précédemment présentés pour mettre en place l'apprentissage.

Notons que les actions de sélection et dé-sélection qui nous intéressent prennent trois arguments en paramètre : le nom du concept en train d'être configuré, le nom du feature, et l'identifiant du sous-système dans lequel l'action est réalisée.

La notion de sous-système est complexe dans le cadre de la ligne mais dans l'apprentissage, nous n'avons travaillé que sur leur identifiant pour différencier les traces.

Les données étudiées correspondent à 33 traces. Chacune des traces représente une configuration d'un système composite de diffusion d'information. Dans le framework, il n'est pas possible de construire de configurations non réalisables (il existe forcément un produit qui intègre la configuration en cours) par contre parmi le jeu de données, certaines configurations ne sont pas terminées, ce qui nous le verrons impacte les raisonnements.

4.2.1. Tri des traces

Nous avons choisi dans cette étude de limiter notre analyse aux actions de sélection et dé-sélection de features. Ainsi notre but est de suggérer à l'utilisateur les prochaines actions à effectuer durant la configuration d'un concept, en fonction de ce que les autres utilisateurs auront fait. Dans cette phase de tri, nous regroupons dans un premier temps les traces par sous-système (i.e. créer un renderer dans un sous-système faisant déjà référence

à une source twitter ne donne pas les mêmes possibilités que de le créer dans le contexte d'un sous-système faisant référence à une source album photo par exemple).

Cette phase de tri se découpe en 3 étapes :

1. découpage des traces obtenues après parsing, et regroupement de ces traces par sous-système : *traces contextualisées*;

2. découpage des traces contextualisées pour obtenir des traces par sous-système et par concept,

3. regroupement des traces obtenues pour former des classes par concept.

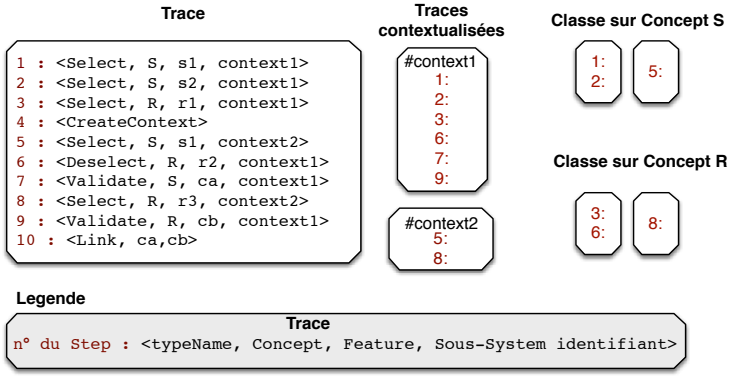

Figure 8. Exemple de tri : la trace initiale fait référence à deux sous-systèmes (context1 et context2) et deux concepts (R pour Renderer et S pour Source). Dans un premier temps, les actions sont regroupées par sous-système, puis nous extrayons de ces sous-systèmes les actions de sélection et dé-sélection propres à chaque concept. En l'occurrence nousobtenons deux traces pour le concept S et deux traces pour le concept R.

Les traces globales que nous avons obtenues à partir du processus de configurations de différents utilisateurs contiennent un ensemble d'actions faisant référence à de multiples sous-systèmes. Ces sous-systèmes sont exploités par l'utilisateur pour réaliser des actions sur des concepts qui seront liés entre eux. Il pourra par exemple définir au sein d'un même sous-système une source d'information, la manière de l'afficher et la zone dans laquelle l'afficher.

Cependant, si ces informations de sous-systèmes sont indispensables dans le cadre du processus de configuration, nous souhaitons pour l'apprentissage ne conserver que les informations des actions effectuées pour chacun des sous-systèmes. Nous obtenons ainsi pour une trace globale donnée, un ensemble de traces correspondant à chacun des sous-systèmes exploités durant le processus. Ainsi plusieurs configurations d'un même concept peuvent être extraites d'une seule trace initiale.

Par ailleurs, chacun de ces sous-systèmes peut contenir des références à différents concepts : par exemple, une référence sur une source, un renderer et une zone. Nous sou-

haitons exploiter l'apprentissage afin d'effectuer des suggestions à l'utilisateur lors de la réalisation d'une configuration pour un concept spécifique.

Ainsi, pour chacune des traces contextualisées, nous obtenons plusieurs traces correspondant à la configuration des différents concepts au sein du sous-système. La figure 8 montre un exemple de trace, les traces contextualisées extraites et le regroupement des traces par concept. Les analyses sur les données présentées au paragraphe précédent s'appuyaient sur cette étape de tri.

Une fois les algorithmes appliqués nous avons analysé les résultats obtenus.

Longueurs des traces d'actions par concept

Nous pouvons remarquer dans la figure 9 que les configurations relatives aux concepts de sources et de renderer présentent le plus grand nombre d'actions en tout. Dans la figure 10 on constate que les traces relatives aux sources sont un peu plus longues, ce qui rend le résultat de l'apprentissage plus pertinent pour les configurations relatives à ce concept.

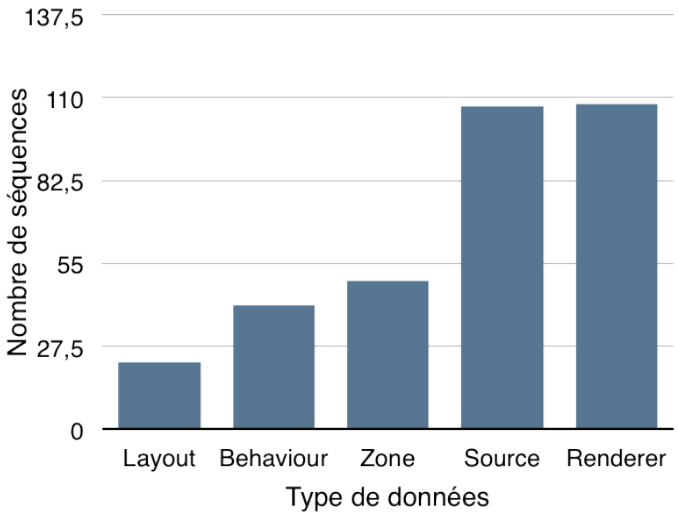

Figure 9. Nombre de traces par concept

Features utilisés

Il est bien connu (Bosch, 2015) que peu de features sont réellement utilisés directement par les utilisateurs. Nous le verifions dans les traces où comme on peut le voir sur la figure 11 que le nombre de features utilisés est nettement plus faible que celui des features présentes dans la ligne. Ce point est d'autant plus marquant que de nombreuses actions uti-

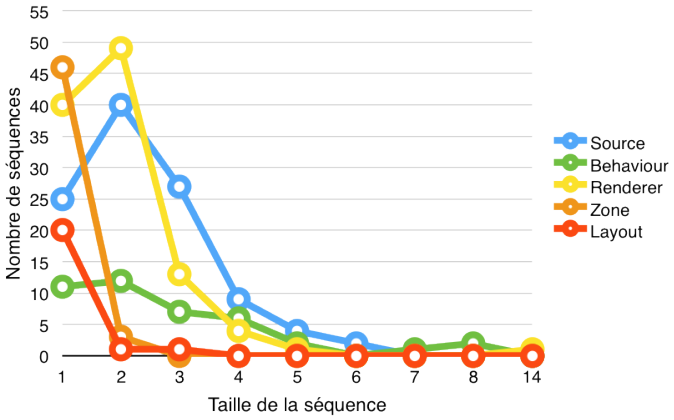

Figure 10. traces en fonction de la taille

lisateurs ont été réalisées. De manière empirique [10], il s'agit de features automatiquement sélectionnés. La connaissance de cette information peut permettre de réduire la vision donnée sur le FM aux utilisateurs. Remarquons que ce n'est pas le cas pour les Behaviours qui semblent avoir des features plus discriminants exposés.

Nous complétons cette brève analyse par la figure 12 qui visualise la fréquence d'utilisation d'une action : deux actions sont les mêmes si elles sont de même nature (select ou deselect) et si elles portent sur le même feature. Nous constatons ici que seules 54 actions sont présentes une seule fois sur les 645 actions répertoriées dans les traces. Les actions les plus fréquentes sont pour les sources la sélection de `Limit` (22), `Now` (19) et `Album` (17) et pour les renderers `GLC` qui est présente 43 fois. La présence d'actions répétées conforte l'idée que des tendances se dégagent dans l'usage de la ligne.

Ainsi le feature `Limit` décrit dans les sources un paramètre permettant de spécifier la limite du nombre d'informations à afficher : s'il n'est pas obligatoire, il apporte cependant énormément de flexibilité sur l'affichage ce qui en fait un des paramètres les plus utilisés. De même, le feature `Now` permet de se baser sur l'heure dynamique pour différentes sources au lieu d'avoir à entrer manuellement une date, il est donc également extrêmement populaire pour l'amélioration de l'expérience utilisateur. Enfin la feature `Album` désigne le choix des sources telles que *FlickR* ou *Picassa* qui sont très fréquentes dans les déploiements réalisés. Pour les Renderers, nous avons déployé un grand nombre de SDI avec un style `GLC`, ce qui explique la forte présence de ce feature.

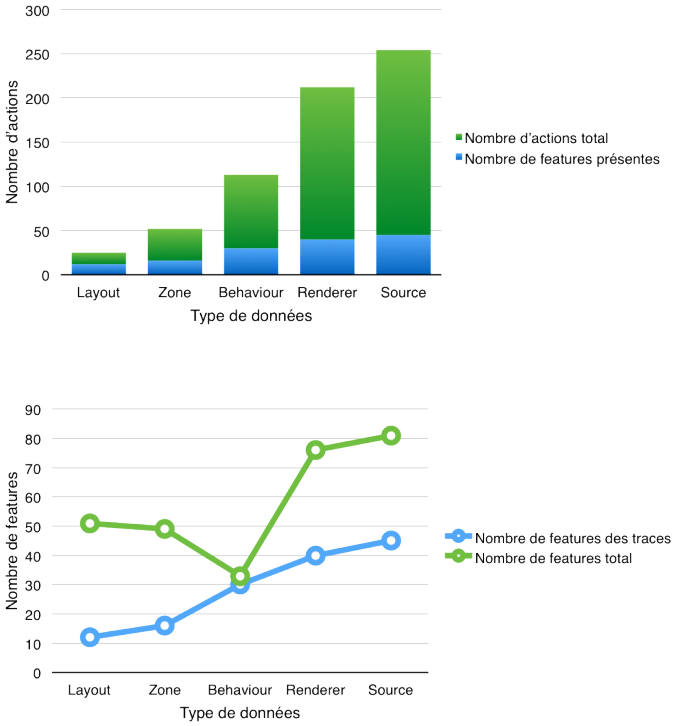

Figure 11. Features utilisés par concept

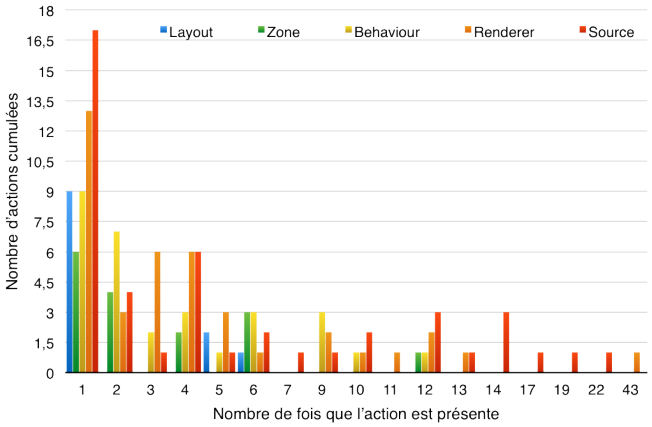

Figure 12. Présence des actions

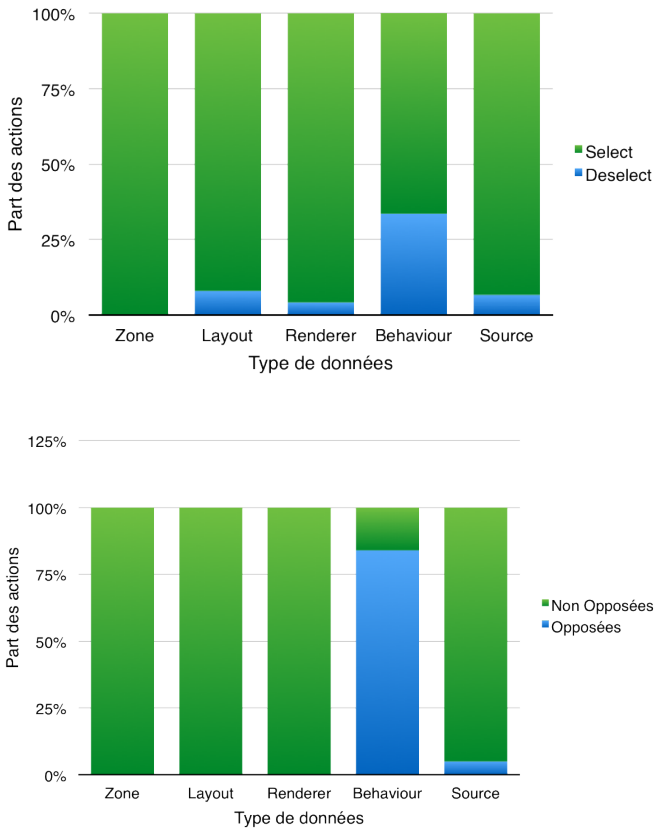

Figure 13. Actions de sélection et désélection et actions contradictoires

Actions contradictoires

Nous avons introduit dans le calcul de distance, la prise en compte des actions contradictoires. Nous avons donc caractérisé dans notre jeu de données leur présence. Dans la figure 13, le premier graphique visualise la présence faible des actions de déselection (voir inexistante pour les zones). On trouve cependant des actions de déselection au sein du FM Behaviour dues à la construction de ce FM : en effet, certains comportements ne sont disponibles que par déselection. Par exemple, la transition la plus simple, l'apparition d'une information sans effet, n'est disponible que par déselection de la caractéristique décrivant une transition avec fondu, qui est le comportement par défaut. Nos utilisateurs sont plus enclin à sélectionner des caractéristiques qu'à en rejeter. En conséquence, les traces in-

10. Ce point devrait être démontré, en vérifiant la présence des features non sélectionnées par l'utilisateur dans les configurations finales, nous ne l'avons pas encore fait.

compatibles au sens où elles contiennent des actions contradictoires sont relativement rares sauf dans le cas des *Behaviours* qui présentent également le plus fort taux de dé-sélection. La prise en compte du coefficient de contradiction est donc à modérer avec le faible taux de contradiction dans les traces. Notons bien que nous ne caractérisons ici que des "contradictions" évidentes mais que les contraintes présentes dans les FMs induisent un nombre bien plus important d'incompatibilité entre les traces. Ce point reste à améliorer.

4.2.2. Clustering

Pour l'étape de clustering, nous avons appliqué les algorithmes présentés en section 3.4, sans modification.

La taille moyenne des traces n'étant pas élevée (voir figure 10), dans cette première série d'expériences, nous avons fixé la valeur de β à 2 et fait varier la valeur de α. Dans la suite, afin de déterminer les valeurs du couple (α, β) qui permettraient d'obtenir des résultats pertinents, nous effectuerons des tests avec plusieurs choix possibles de α et β.

L'algorithme de clustering étant non-déterministe, pour évaluer la pertinence d'un choix, il a fallu faire plusieurs tests pour chacun des couples de valeurs considérées. Nous avons calculé le coefficient "kappa" pour chacun de ces résultats. Puis, nous avons calculé la moyenne et l'écart-type de ces valeurs de kappa.

4.2.2.1. Nombre de clusters et calcul de la distance

La construction d'un centroïde est basée sur le mode des actions les plus présentes à une position donnée dans son cluster; la prise en compte des traces d'actions de longueurs différentes, dont des traces de longueur 1, perturbe donc le calcul du centroïde. Après 1000 itérations, nous avons obtenu en moyenne 18 clusters pour alpha =0,25 et beta = 2 pour les sources. Dans le cas où alpha vaut 1, donc que l'on donne un poids très fort au fait que deux actions ne sont pas placées à la même place, on constate une augmentation du nombre de clusters (cf. figure 14).

L'absence de variation du nombre de clusters sur les zones en fonction de alpha est à corréler avec le fait que les traces sur les zones sont très courtes, ce qui fait que la variabilité a pu être absorbée directement par les clusters qui contiennent uniquement des traces identiques.

La figure 15 visualise les résultats pour tous les concepts après seulement 100 itérations. Pour les sources et les renderers qui présentent prêts de 50 séquences, ce nombre d'itération faible donne seulement une idée de l'ordre de grandeur. On constate que l'on obtient des clusters d'environ 50 traces pour les renderers ce qui démontre la présence de tendances, cela semble aussi le cas pour les sources. Pour le nombre de clusters, relativement au nombre de traces initiales (voir Figure 9) et à la détermination non déterministe du nombre de clusters, nous en déduisons également que des tendances se dégagent.

Nous nous sommes également interrogé sur la pertinence de l'algorithme de calcul du centroïde par rapport au choix d'un médoïde, la fréquence montre que en effet dans de nombreux cas les centroïdes calculés correspondent à des médoïdes, ce qui peut faire douter de la nécessité de calculer les centroïdes, en particulier pour les Layout et les zones. Des

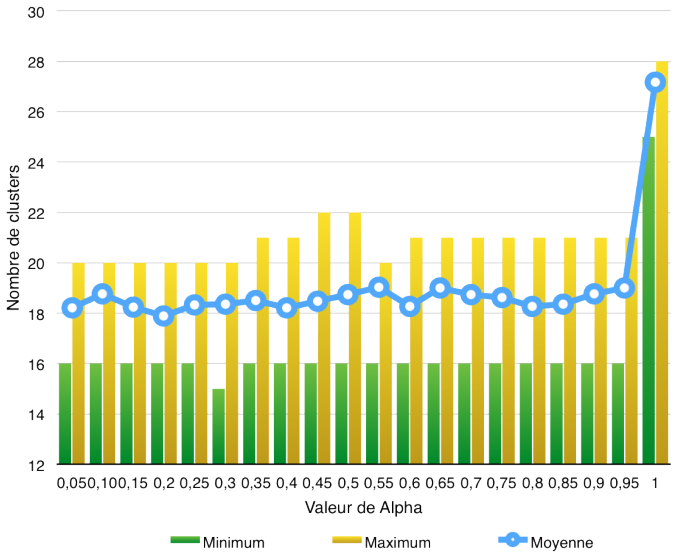

Figure 14. Nombre de clusters en fonction d'alpha

chiffres obtenus sur un plus grand nombre d'itérations devraient nous permettre d'affiner ce résultat.

		Layout	Behaviour	Zones
alpha =0	[taille min des clusters, taille max des clusters]	[1.0, 3.0]	[1.0, 13.0]	[1.0, 4.0]
	Freq. des Medoids (mean, min, max, average)	[100, 100, 100, 100]	[71.6, 66.7, 83.3, 75.0]	[100, 100, 100, 100]
	Nbre de Clusters (mean, min, max, average)	[7.04, 7.0, 8.0, 7.5]	[6.2, 6.0, 7.0, 6.5]	[14.0, 14.0, 14.0, 14.0]
alpha =0,25	[taille min des clusters, taille max des clusters]	[1.0, 3.0]	[1.0, 13.0]	[1.0, 4.0]
	Freq. des Medoids (mean, min, max, average)	[100, 100, 100, 100]	[73.0, 66.7, 85.7, 76.2]	[100, 100, 100, 100]
	Nbre de Clusters (mean, min, max, average)	[7.03, 7.0, 8.0, 7.5]	[6.22, 6.0, 7.0, 6.5]	[14.0, 14.0, 14.0, 14.0]
alpha =1	[taille min des clusters, taille max des clusters]	[1.0, 3.0]	[1.0, 7.0]	[1.0, 4.0]
	Freq. des Medoids (mean, min, max, average)	[100, 100, 100, 100]	[87.8, 87.5, 100.0, 93.8]	[100, 100, 100, 100]
	Nbre De Clusters (mean, min, max, average)	[8.05, 8.0, 9.0, 8.5]	[8.01, 8.0, 9.0, 8.5]	[14.0, 14.0, 14.0, 14.0]

		Renderer	Source
alpha =0	[taille min des clusters, taille max des clusters]	[1.0, 48.0]	[1.0, 24.0]
	Freq. des Medoids (mean, min, max, average)	[86.2, 75.0, 86.7, 80.9]	[86.3, 80.0, 93.8, 86.9]
	Nbre de Clusters (mean, min, max, average)	[16.12, 15.0, 17.0, 16.0]	[14.88, 14.0, 15.0, 14.5]
alpha =0,25	[taille min des clusters, taille max des clusters]	[1.0, 47.0]	[1.0, 24.0]
	Freq. des Medoids (mean, min, max, average)	[84.6, 76.5, 87.5, 81.9]	[86.2, 80.0, 87.5, 83.7]
	Nbre de Clusters (mean, min, max, average)	[16.77, 15.0, 18.0, 16.5]	[14.99, 14.0, 16.0, 15.0]
alpha =1	[taille min des clusters, taille max des clusters]	[1.0, 35.0]	[1.0, 24.0]
	Freq. des Medoids (mean, min, max, average)	[89.6, 85.0, 90.0, 87.5]	[79.8, 76.5, 95.0, 85.7]
	Nbre De Clusters (mean, min, max, average)	[19.53, 19.0, 20.0, 19.5]	[17.43, 17.0, 19.0, 18.0]

Figure 15. Variabilité des clusters en fonction des concepts et d'alpha, pour beta=2

4.2.2.2. Pertinence des clusters

La Figure 16 représente les actions présentes sur ces 1000 itérations dans les centroïdes. Nous pouvons remarquer que les actions les plus fréquentes sont bien prises en compte dans les centroïdes (plus 6% des actions totales). Par contre pour les actions présentent une seule

fois, leur prise en compte dans les centroïdes est lié à la manière de construire les clusters lorsque la distance est de 0.

4.2.3. Guidage

Avec Alpha = 0,25 et Beta = 2, on constate que la majorité des actions sont bien prises en compte dans les centroïdes (cf. figure 16). Les actions qui ne sont pas prises en compte comme par exemple la désélection de `YCAnnouncement` qui apparait 4 fois, est dû à sa faible désélection, et à la position aléatoire de l'action.

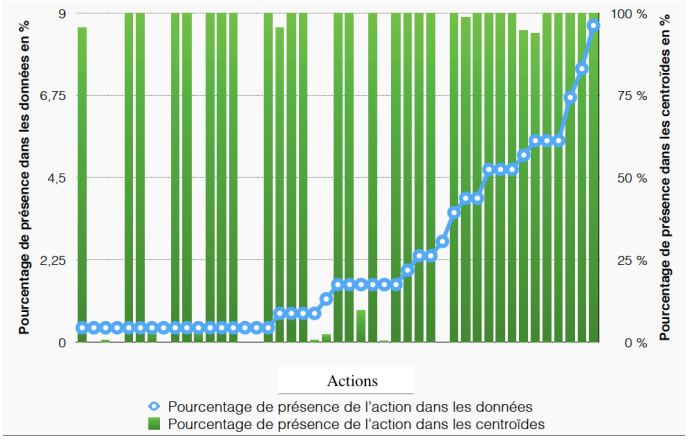

Figure 16. Pourcentage de présence des actions dans les centroïdes dans le cas des sources

Dans les résultats de la figure 17, le coefficient beta est fixé à 2, et nous faisons varier alpha. Le choix de alpha dépend des concepts. Avec les résultats obtenus, pour l'instant, le choix qui semble le plus judicieux serait celui de alpha compris entre 0.5 et 0.7, mais un choix en fonction des concepts devrait donner de meilleurs résultats. Cela s'explique par les différences dans les jeux de données. En effet, nous cherchons à (1) avoir la meilleure valeur possible de la moyenne des coefficients Kappa (moyenne élevée) et à (2) réduire l'écart type afin que la probabilité de tomber sur des valeurs proches de la "moyenne élevée" soit grande. De plus, nous avons constaté que quand la valeur de α varie entre 0.5 et 0.7 le nombre de clusters est constant ce qui rend la comparaison entre les différents résultats plus pertinente. En s'approchant de alpha=1, on constate un accroissement important du nombre de clusters, l'ordre d'apparition des actions étant plus fortement discriminant, inversement, trop proche de 0, il y a généralement un coefficient Kappa bas.

Un travail est en cours pour déterminer le choix simultané de alpha et beta en fonction des mêmes critères.

4.3. Discussion sur la validité des résultats

Les traces que nous avons étudiées ont correspondu à un ensemble de 500 actions environ. La décomposition de ces traces dans l'étape de tri réduit très nettement leur nombre.

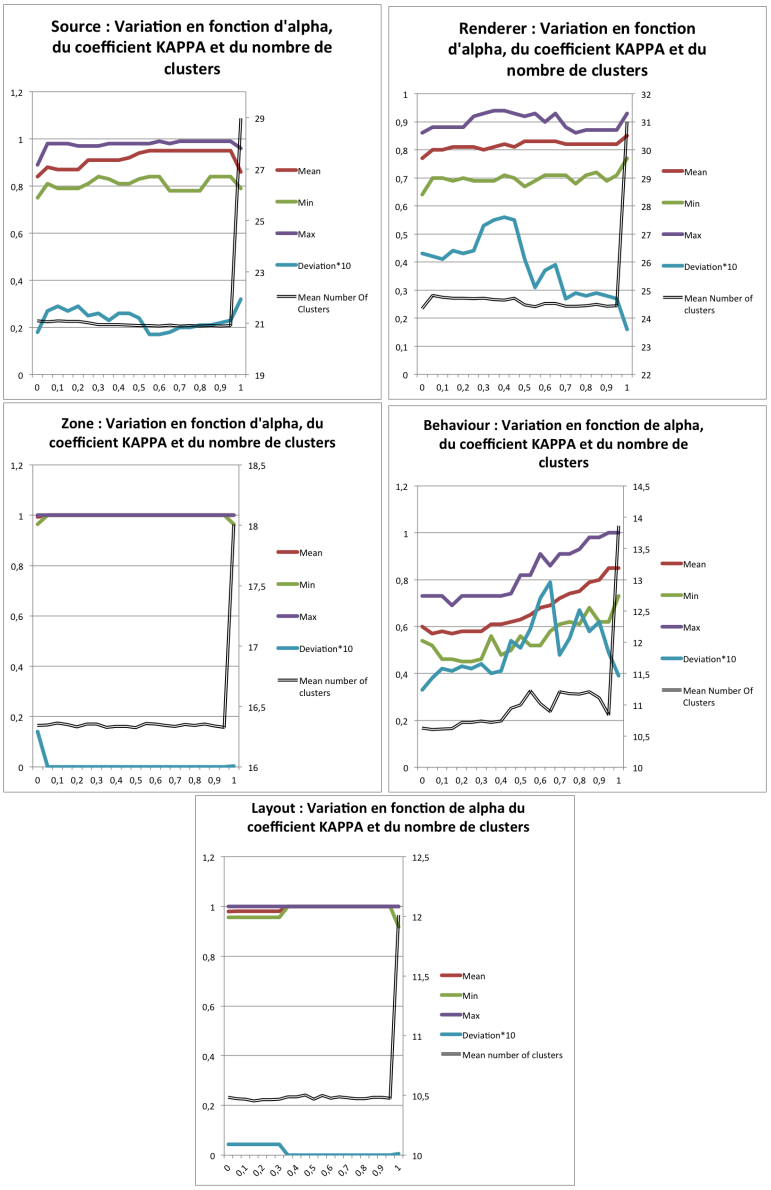

Figure 17. Variations du coefficient kappa sur une prédiction n+1 en fonction de la valeur Alpha; l'axe de gauche présente les variations du coefficient, l'axe de droite le nombre moyen de clusters, en fonction d'alpha.

D'une part, nous pouvons en déduire que l'apprentissage peut se faire sur des bases relativement faibles de données, d'autre part, il nous reste à prouver la pertinence de nos algorithmes sur de grandes masses de traces. Nous espérons pouvoir aborder ce point en travaillant sur des traces issues d'autres lignes de produits.

Des recommandations peuvent s'avérer fausses relativement à la validité des produits, des actions pouvant être contradictoires par le jeu des contraintes dans la ligne. Eliminer ces cas dans la construction des centroïdes et peut-être également dans le calcul des similarités pourrait nous permettre d'améliorer nos résultats avec la question cependant sous-jacente de l'impact sur le nombre de clusters et les temps de calcul.

5. Conclusion

Nous avons posé les bases d'un apprentissage pour guider l'utilisateur dans le processus de configuration dans des lignes de produits logiciels. Nous avons appliqué ces travaux à l'apprentissage dans le cadre de la ligne de produits logiciels Yourcast qui permet la configuration de systèmes composés de plusieurs sous-systèmes. Cette étude a conforté l'intuition que nous avions que des "tendances" de configuration existaient bien. Nous avons également appris que les algorithmes standards devaient être adaptés pour prendre en compte le cadre applicatif particulier qui nous intéresse ici. Ce travail de défrichage a ouvert de nombreuses perspectives dont certaines sont encore sous la forme de questions ouvertes.

De plus en plus les LPL sont vues comme des écosystèmes qui intègrent des outils de traçages des configurations voir même de l'exécution des produits (Schmid, 2013). Les apprentissages devraient alors non seulement pouvoir intégrer les nouvelles traces obtenues mais également intégrer les nouveaux features, alors que ceux-ci ont été peu choisi précédemment.

L'utilisation des sous-systèmes comme fil directeur va influencer le processus de configuration en proposant de préférence des actions dans le même sous-système. Les traces globales et d'autres actions de configuration du produit final pourraient également être exploitées afin d'offrir une approche plus globale des suggestions possibles à l'utilisateur. Nous pensons utiliser les travaux sur les arbres attribués pour aborder cette question (Pasquier *et al.*, 2015).

A terme, nous souhaitons comparer les premiers résultats avec les heuristiques proposées dans (Mazo, Dumitrescu, 2014), en particulier en tenant compte des profils des utilisateurs. Nous avons de manière empirique constaté que les utilisateurs expérimentés utilisent intuitivement de telles stratégies pour accélérer la production des applications. De telles stratégies seraient-elles cependant satisfaisantes pour un utilisateur non expérimenté qui ne comprendra pas forcément les choix qui lui sont proposés lorsqu'ils se situent trop loin de sa compréhension du domaine? De même, nous pensons qu'il devrait être possible d'utiliser les traces d'interactions pour identifier des profils utilisateurs, ce qui permettrait d'améliorer la pertinence des recommandations dès les premières étapes de la configuration. Par exemple, la configuration d'écrans de diffusion pour des établissement de per-

sonnes handicapées, ou au contraire pour des écoles ou des évènements conduit-elle à des comportements de configuration caractéristiques?

La prise en compte des recommandation au niveau de l'interface devra être étudiée avec soin, et nous nous intéressons à une automatisation dans certains cas des actions préconisées par exemple en proposant non pas une mais un ensemble d'actions. L'identification de profils utilisateurs ainsi que l'ajout d'information sur les configurations pourraient aider à enrichir cette étape de présentation des recommandations.

Bibliographie

Abbasi E. K., Hubaux A., Heymans P. (2011). A toolset for feature-based configuration workflows. In *Proceedings - 15th international software product line conference, splc 2011*, p. 65–69.

Benavides D., Segura S., Ruiz-Cortés A. (2010). Automated analysis of feature models 20 years later: A literature review. *Information Systems*, vol. 35, n° 6, p. 615–636.

Bosch J. (2015). *From Opinions to Facts: Building Products Customers Actually Use, Keynote CAISE'15.* Consulté sur http://www.bits-chips.nl/fileadmin/docs/Commercieel/Jan_Bosch_Keynote.pdf

Bosch J., Högström M. (2001). Product instantiation in software product lines: A case study. *Lecture Notes in Computer Science*, vol. 2177, p. 147–162. Consulté sur http://www.springerlink.com/index/n1771uw66052x4p0.pdf

Boucher Q., Perrouin G., Heymans P. (2012). Deriving Configuration Interfaces from Feature Models : A Vision Paper. In *Variability modelling of software intensive systems vamos.*

Chakrabarti D., Kumar R., Tomkins A. (2006). Evolutionary clustering. In *Proceedings of the 12th acm sigkdd international conference on knowledge discovery and data mining*, p. 554–560. New York, NY, USA, ACM. Consulté sur http://doi.acm.org/10.1145/1150402.1150467

Cordier A., Lefevre M., Champin P.-A., Georgeon O., Mille A. (2013, mai). Trace-Based Reasoning — Modeling interaction traces for reasoning on experiences. In *The 26th international flairs conference.* Consulté sur http://liris.cnrs.fr/publis/?id=5955

Dhungana D., Grünbacher P., Rabiser R. (2010). The DOPLER meta-tool for decision-oriented variability modeling: a multiple case study. *Automated Software Engineering*, vol. 18, n° 1, p. 77–114. Consulté sur http://www.springerlink.com/index/10.1007/s10515-010-0076-6

Eban E., Birnbaum A., Shalev-Shwartz S., Globerson A. (2012). Learning the experts for online sequence prediction. In *Proceedings of the 29th international conference on machine learning, ICML 2012, edinburgh, scotland, uk, june 26 - july 1, 2012.* icml.cc / Omnipress. Consulté sur http://icml.cc/discuss/2012/471.html

Hubaux A., Classen A., Heymans P. (2009). Formal Modelling of Feature Configuration Workflows. In *Proceedings of the 13th international software product line conference (splc'09)*, p. 221–230.

Kang K. C., Cohen S. G., Hess J. A., Novak W. E., Spencer Peterson A. (1990). *Feature-Oriented Domain Analysis (FODA) Feasability Study*. Rapport technique n° 1, vol. 17.

Mazo R., Dumitrescu C. (2014). Recommendation Heuristics for Improving Product Line Configuration Processes. *Recommendation ...*, p. 1–27. Consulté sur http://hal.archives-ouvertes.fr/hal-00914021/

Pasquier C., Sanhes J., Flouvat F., Selmaoui-Folcher N. (2015). Frequent pattern mining in attributed trees: algorithms and applications. *Knowledge and Information Systems*, p. 1–24. Consulté sur http://dx.doi.org/10.1007/s10115-015-0831-x

Passos L., Guo J., Teixeira L., Czarnecki K. (2012). Coevolution of Variability Models and Related Artifacts: A Case Study from the Linux Kernel. In *Proceedings of the 16th international software product line conference - volume 1*, p. 76–85. Consulté sur http://gp.uwaterloo.ca/sites/default/files/coevolution.pdf

Pohl K., Böckle G., Linden F. van der. (2005). *Software product line engineering: Foundations, principles and techniques*. Springer, Berlin Heidelberg New York.

Quinton C., Romero D., Duchien L. (2014). Automated selection and configuration of cloud environments using software product lines principles. In *2014 IEEE 7th international conference on cloud computing, anchorage, ak, usa, june 27 - july 2, 2014*, p. 144–151. IEEE. Consulté sur http://dx.doi.org/10.1109/CLOUD.2014.29

Robillard M., Walker R., Zimmermann T. (2010). Recommendation systems for software engineering. *IEEE Software*, vol. 27, n° 4, p. 80–86.

Schmid K. (2013). Variability support for variability-rich software ecosystems. *2013 4th International Workshop on Product LinE Approaches in Software Engineering (PLEASE)*, p. 5–8. Consulté sur http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6608654

Schobbens P.-Y., Heymans P., Trigaux J.-C., Bontemps Y. (2007). Generic semantics of feature diagrams. *Computer Networks*, vol. 51, n° 2, p. 456–479.

Urli S. (2015). *Processus Flexible de Configuration pour Lignes de Produits Logiciels Complexes*. Thèse de doctorat non publiée, Université Nice Sophia Antipolis. Consulté sur https://tel.archives-ouvertes.fr/tel-01134191v1

Urli S., Blay-fornarino M., Collet P. (2014). Handling Complex Configurations in Software Product Lines : a Tooled Approach. In ACM (Ed.), *Proceedings of the 18th international software product line conference (splc'14)*, p. 112–121. Florence, Italy.

Urli S., Blay-fornarino M., Collet P., Mosser S., Riveill M. (2014). Managing a Software Ecosystem Using a Multiple Software Product Line : a Case Study on Digital Signage Systems. In *Proceedings of the euromicro conference series on software engineering and advanced applications (seaa'14)*, p. 344–351.

Urli S., Perez G., Zitoun H., Blay M., Collet P., Renevier-gonin P. (2012). Vers des interfaces graphiques flexibles de configurations. In *Journées sur les lignes de produits logiciels (jldp).*

Xu R., Wunsch I., D. (2005, May). Survey of clustering algorithms. *Neural Networks, IEEE Transactions on*, vol. 16, n° 3, p. 645-678.

Zarka R., Cordier A., Egyed-Zsigmond E., Lamontagne L., Mille A. (2013). Mesures de Similarité pour Comparer des épisodes dans les Traces Modélisées. In *11èmes rencontres des jeunes chercheurs en intelligence artificielle (rjcia 2013)*, p. 1–15. Consulté sur http://liris.cnrs.fr/publis/?id=6136