Feature models pour l'acquisition des préférences d'un utilisateur en situation de déficience visuelle

Austin Kouhoué, Marianne Huchard, Thomas Bouetou

Feature models pour l'acquisition des préférences d'un utilisateur en situation de déficience visuelle

Austin Waffo Kouhoué[1], Marianne Huchard[2], and Thomas Bouetou Bouetou[1]

[1] LIMSI, Ecole Nationale Supérieure Polytechnique de l'Université de Yaoundé I
Yaoundé, Cameroun
{wkaustin1,tbouetou}@gmail.com
[2] LIRMM, CNRS & Université de Montpellier,
Montpellier, France
marianne.huchard@lirmm.fr

Résumé

Cet article s'intéresse à l'étude des options d'accessibilité de plusieurs systèmes d'exploitation, dans l'intention de déterminer les points communs qui serviront de base à la réalisation d'une interface de recueil des besoins des personnes en situation de handicap visuel. Nous commençons par décrire la modélisation des options d'accessibilité d'un système d'exploitation sous forme d'un *feature model* (FM) à l'aide de l'outil *FAMILIAR*. Puis nous présentons les pistes de recherche pour déterminer les points communs des FMs de plusieurs systèmes d'exploitation à travers l'application des méthodes suivantes : l'alignement des vocabulaires, quelques opérations de *FAMILIAR* et l'Analyse Formelle de Concepts (AFC). Ce travail vise à affiner la première étape d'une chaîne complète d'adaptation de pages Web à destination des personnes en situation de handicap visuel (moyenne et basse vision) qui est actuellement centrée sur un faible nombre de préférences [1].

1 Introduction

La dématérialisation des procédures et des échanges (*e-government*, *e-commerce*, *e-learning*, etc.) est en pleine croissance. Elle se fait en majeure partie à travers les interfaces Web. Malheureusement, ces interfaces ne sont pas exploitables par tous. En particulier, les personnes âgées ou atteintes d'une altération physique partielle ou totale de l'œil éprouvent d'énormes difficultés lors de l'utilisation de ces interfaces.

Face à ce problème, le *World Wide Web (W3C)*, organisme international reconnu pour le web, publie un ensemble de recommandations à respecter par les concepteurs et développeurs de sites web. Cependant, ces recommandations ne font l'objet d'aucune obligation d'application surtout dans les cas où les interfaces sont conçues par des particuliers. Par conséquent, un bon nombre de ressources Web demeurent encore inaccessibles [4].

Au-delà des normalisations, des standards et des recommandations, des solutions ont été proposées par des dispositifs présents dans les systèmes d'exploitation, dans les navigateurs Web et par les outils d'assistance, qui sont commerciaux pour la plupart d'entre eux. Ces dispositifs proposent des modifications globales, qui ne prennent pas en compte les problèmes spécifiques d'un utilisateur final. Des travaux proposent au contraire des adaptations personnalisées [4] et cherchent à la fois à satisfaire les préférences d'un utilisateur et à préserver autant que possible l'esprit du design initial de la page. Le problème d'adaptation ainsi défini est traité comme un problème d'optimisation combinatoire car les préférences et le design initial peuvent comporter des éléments contradictoires.

Pour mettre en œuvre ces solutions d'adaptation, il est nécessaire d'acquérir les besoins et les préférences des utilisateurs potentiels de manière à améliorer leur confort visuel, la qualité de leurs interactions et dans certains cas tout simplement de manière à leur

1. http ://ewpa.lirmm.fr/

permettre l'accès aux ressources. C'est une tâche complexe car des personnes ayant une même pathologie peuvent posséder des préférences différentes [4].

Dans cet article, nous nous intéressons à cette étape d'acquisition. Nous proposons d'analyser les options d'accessibilité offertes par plusieurs systèmes d'exploitation afin d'en tirer les options les plus communément proposées. Nous nous intéressons aussi à leur organisation afin de disposer des grandes lignes d'une interface d'acquisition des préférences utilisateur. Pour effectuer cette tâche, nous modélisons les options d'accessibilité à l'aide de *Feature Models*, afin de capturer à la fois ces options et la manière dont elles sont présentées à un utilisateur. Nous présentons ensuite des pistes de recherche pour aligner le vocabulaire et l'organisation de ces *Feature Models* et extraire un *Feature Model* intersection qui sera la base de notre future interface d'acquisition. Ce travail vise à affiner la première étape d'une chaîne complète d'adaptation de pages Web à destination des personnes en situation de handicap visuel (moyenne et basse vision) qui est actuellement centrée sur un faible nombre de préférences.

Nous présentons la modélisation des options d'accessibilité d'un système d'exploitation par un FM à la section 2. Ensuite, nous décrivons quelques pistes d'extraction de points communs de plusieurs de ces FMs à la section 3. L'article se conclut section 4.

2 Modélisation des options d'accessibilité des systèmes d'exploitation

Du fait que, au sein des systèmes d'exploitation, les options d'accessibilité sont disposées pour un parcours utilisateur sous une forme hiérarchique et qu'elles expriment des configurations possibles, une représentation simple et intuitive nous a semblé être les *Feature Models* (FMs). Dans cette partie nous expliquons quel procédé nous avons utilisé pour la modélisation et les FMs obtenus.

Brève présentation des features models et de FAMILIAR Un *Feature Model*, ou modèle de caractéristiques, est une représentation graphique sous forme d'un arbre logique d'un ensemble de caractéristiques ainsi que de leurs dépendances. Il représente la variabilité au sein d'une famille de produits, de manière compacte et compréhensible [6]. Le formalisme que nous utilisons ici possède quatre opérateurs hiérarchiques (And, Optionnel, groupe Or, groupe Xor) et deux opérateurs transverses (qui indiquent des contraintes sur des *features* qui ne sont pas en relation mère-fille dans l'arbre) : Require et Mutex.

Dans notre travail, nous avons utilisé FAMILIAR (pour FeAture Model scrIpt Language for ManIpulation and Automatic Reasoning), outil introduit par Acher et al. [3] qui fournit une solution opérationnelle à la gestion de multiples FMs. En effet,il vise à définir, combiner, analyser et manipuler les modèles de *features*. De plus, FAMILIAR [1] offre un certain nombre d'opérations utiles pour la manipulation telles que le comptage du nombre de configurations valides, la fusion de modèles de *features*, l'insertion d'un modèle dans un autre, le filtrage par une configuration partielle, ou la projection.

Construction des FMs Pour réaliser les FMs, nous parcourons les options de façon exhaustive et hiérarchisée, comme un utilisateur qui créerait son profil complet d'accessibilité. Les *features* sont créées et ajoutées au FM au fur et à mesure du parcours. Ce procédé permet à la fois de connaître les options mais également la manière dont leur organisation est présentée à un utilisateur. Mais nous avons été confrontés à un problème de réapparition d'une même caractéristique dans plusieurs chemins de parcours, ce qui fait sortir de la représentation sous forme arborescente des FMs. D'ailleurs FAMILIAR n'admet pas que deux caractéristiques aient la même étiquette pour cette raison. Nous expliquons comment nous avons traité ce problème, puis nous présentons les FMs construits.

Traitement des caractéristiques accessibles par plusieurs chemins Pour traiter ce problème, nous considérons :

— E_1 l'ensemble des configurations initiales, qui peuvent être obtenues par parcours utilisateur des options et que nous souhaitons représenter ;
— FM le modèle de *features* construit à partir de E_1 ;
— E_2 l'ensemble des configurations valides générées à partir de FM.

Notre but est d'avoir $E_1 = E_2$ ou à défaut que E_1 puisse être reconstruit de manière non ambiguë à partir de E_2. Dans le cas de *features* apparaissant dans plusieurs arborescences, nous avons observé, lors de l'activité de modélisation, qu'il est difficile d'obtenir facilement un FM, possédant exactement l'ensemble de caractéristiques visé, et dont l'organisation soit intuitive et non encombrée de contraintes.

Nous avons étudié deux approches pour améliorer la modélisation, qui toutes deux vont introduire de nouvelles *features* : la première s'intéresse à construire un FM sans contraintes et la seconde s'intéresse à construire un FM avec des contraintes.

La première approche consiste simplement à dupliquer (en donnant un nom différent à chaque duplication) la *feature* accessible par plusieurs chemins (répétée) en autant de fois qu'elle réapparaît dans les chemins de navigation. La deuxième approche consiste, lorsque l'on rencontre une *feature* dans plusieurs chemins, à accrocher à un point de jonction des chemins une *feature* optionnelle représentant la *feature* répétée. Puis des *features* artificielles sont introduites, autant de fois que la *feature* était rencontrée, qui servent en quelque sorte de références (ou pointeurs) vers la *feature* répétée et on ajoute des contraintes d'implication complétant la modélisation.

L'approche sans contraintes utilisant une duplication des *features* présente le désavantage, en cas de modification d'une *feature* dupliquée, que la modification soit mal reportée ou réalisée de manière incohérente. La modélisation avec création de *features* artificielles servant de pointeurs vers la *feature* en cause, accompagnée de contraintes simples à comprendre, nous a semblé être une meilleure alternative qui évite les problèmes liés aux duplications.

Application aux FMs Dans un cadre expérimental, nous nous sommes intéressés aux options d'accessibilité de deux versions de trois systèmes d'exploitation très répandus. La table 1 présente ces systèmes accompagnés de leurs versions et leurs dimensions. Ces différents *FMs* sont disponible en ligne dans les formats *FAMILIAR*[2] et *SPLOT*[3].

Feature Model	nb. de Features	nb. de Contraintes
Android5.1	210	38
Android6.0	179	26
Ubuntu12.04	263	84
Ubuntu14.10	261	72
Windows7	329	108
Windows8	451	138

TABLE 1 – Quelques caractéristiques des FMs réalisés

3 Pistes pour l'extraction des options communes

Pour l'extraction des options communes, nous sommes en cours d'étude de plusieurs solutions.

2. https://www.lirmm.fr/users/utilisateurs-lirmm/marianne-huchard/phd-students/austin-waffo-phd/feature-models-for-accessibility-options
3. http://www.splot-research.org

Alignement des vocabulaires Nous avons tout d'abord défini un ensemble de thématiques inspirées des options d'accessibilité du système d'exploitation Windows7. Puis, dans chacune d'elle, nous mettons en correspondance des ensembles de noms de *features* issus des différents *FMs* choisis (voir Table 2). Ce travail a été effectué manuellement sur trois des FMs (*Android51, Ubuntu1410, et Windows7*) et nous l'étendrons aux suivants.

N	Thématique	FM Android51	FM Ubuntu1410	FM Windows7
1	Améliorer la lisibilité de l'ordinateur	TextToSpeechOutput	ScreenReader	HearTextAndDescriptionReadAloud
		AutoRotateScreen	Rotation	
		LargeText, FontSize et TextSize	LauncherIconSize, IncreaseTextSize, DecreaseTextSize et LargeText	ChangeTheSizeOfTextAndIcon
		MagnificationGesture	ZoomIn et ZoomOut	
		HighContrastText	HighContrast et HighContrastOnOrOff	HighContrast
		ColorInversion, ColorCorrection		
		Wallpaper	Background et Theme	FineTuneDisplayEffects
2	Utiliser l'ordinateur sans écran	TextToSpeechOutput	ScreenReader	HearTextReadAloud
3	Utiliser l'ordinateur sans souris ni clavier	AndroidKeyboardAOSP	ScreenKeyboard	UseOnScreenKeyboard
				UseSpeechRecognition
4	Rendre la souris plus facile à utiliser	MouseOrTouchpad	PointingAndClicking	MakeTheMouseEasierToUse
5	Rendre le clavier plus facile à utiliser		TextEntry	MakeTheKeyboardEasierToUse

TABLE 2 – Regroupement des différentes *features* par thématique

Utilisation des opérations de FAMILIAR *FAMILIAR* offre une opération de *merge intersection* [1] qui réalise l'intersection de *FMs* qui nous semble intéressante pour notre travail. Lorsqu'on fait l'intersection de deux *FMs*, on obtient un nouveau *FM* dont les configurations valides sont des configurations valides de ces deux *FMs* [2]. Malheureusement, son usage direct produit une intersection vide, à cause de la divergence dans le nommage et l'organisation. Pourtant il existe bel et bien des *features* appartenant aux trois *FMs* considérés (voir Table 1). Il nous faut donc approfondir l'utilisation de cette opération sur nos données et notamment après alignement des vocabulaires pour que sa mise en œuvre soit pertinente.

Utilisation de l'Analyse Formelle de Concepts Nous avons pensé ensuite utiliser l'Analyse Formelle de Concepts pour ses qualités d'extraction d'attributs communs à des objets. Les objets seraient les *FMs* et les attributs seraient les *features*. Comme au sein de ces *FMs*, on retrouve des *features* qui ont quasiment la même signification mais avec des noms différents à l'instar de *MagnificationGestures* de Android51, *ZoomIn ZoomOut* de *Ubuntu1410* et *Magnifier* de *Windows7* (voir table 2), nous uniformiserions d'abord les noms des *features* à l'aide de cette table, avant de procéder par la suite à l'analyse.

Mais un problème sera que l'AFC utilisée de cette manière n'extrait que des ensembles communs de *features* et pas la structure des FMs. Une solution sera peut-être d'utiliser

l'Analyse Relationnelle de Concepts (ARC) comme on le fait pour les modèles UML [7]. Une autre approche avec l'AFC consiste à générer les configurations d'option, prendre leur intersection, puis ré-extraire un FM à partir de la structure conceptuelle obtenue, mais elle n'est applicable dans l'état actuel de la recherche que sur des petites parties du FM [5]. Il faudrait donc procéder à une décomposition préalable.

Autres pistes Nous voudrions étudier également les méthodes *Catalogue Rules*, *Compositional Approach*, *Transformational Approach*, et *Boolean Logic Based* citées dans [2].

4 Conclusion et perspectives

Ce travail s'inscrit dans le cadre d'un projet d'accessibilité numérique. Il se base sur l'analyse des options d'accessibilité offertes par plusieurs systèmes d'exploitation afin d'en tirer les options les plus communément proposées qui serviront de base à la réalisation d'une interface d'acquisition des préférences utilisateurs. Pour effectuer cette tâche, nous avons modélisé ces options d'accessibilité à l'aide de *Feature Models* et de l'outil *FAMILIAR*. Dans l'intention d'extraire les points communément proposés, nous avons commencé à explorer les possibilités suivantes : alignement des vocabulaires, opérations offertes par *FAMILIAR*, Analyse Formelle de Concepts (AFC). Le travail est encore en cours.

Remerciements. Nous remercions Y. Bonavero, M. Meynard, E. Bourreau, J.-C. König qui prennent part à ce projet et l'entreprise Berger Levraut qui lui apporte son soutien.

Références

[1] Mathieu Acher. familiar-documentation. https://github.com/FAMILIAR-project/familiar-documentation, 2016.

[2] Mathieu Acher, Philippe Collet, Philippe Lahire, and Robert B. France. Comparing approaches to implement feature model composition. In *ECMFA 2010*, pages 3–19, 2010.

[3] Mathieu Acher, Philippe Collet, Philippe Lahire, and Robert B. France. FAMILIAR : A domain-specific language for large scale management of feature models. *Sci. Comput. Program.*, 78(6) :657–681, 2013.

[4] Yoann Bonavero. Une approche basée sur les préférences et les méta-heuristiques pour améliorer l'accessibilité des pages Web pour les personnes déficientes visuelles Thèse de doctorat de l'Université de Montpellier. https://hal-lirmm.ccsd.cnrs.fr/tel-01312294, 2015.

[5] Jessie Carbonnel, Marianne Huchard, André Miralles, and Clémentine Nebut. Feature Model composition assisted by Formal Concept Analysis. In *12th Int. Conf. ENASE*, pages 27–37, 2017.

[6] Aymeric Hervieu. Approche à contraintes pour la sélection de Covering Array. Thèse de doctorat de l'Université de Rennes 1. https://tel.archives-ouvertes.fr/tel-00915223, 2013.

[7] André Miralles, Marianne Huchard, Xavier Dolques, Florence Le Ber, Thérèse Libourel, Clémentine Nebut, and Abdoulkader Osman Guédi. Méthode de factorisation progressive pour accroître l'abstraction d'un modèle de classes. *Ingénierie des Systèmes d'Information*, 20(2) :9–39, 2015.