Adaptation des interfaces à l'environnement

Étude technologique

&

Synthèse de recherche

Geneviève Cirera SI5-WEB

Table des matières

1 Étude technologique

Dans cette partie, nous allons réaliser une étude de deux technologies de type RWD [1]. Les technologies choisies sont Bootstrap et Fondation, après une description de chacune d'entre elle, nous les comparerons afin d'en tirer les avantages et inconvénients de chacune.

1.1 RWD

Les deux framework sont des technologies RWD, ils permettent à une application web de s'adapter au support sur lequel elle est visualisée, à la taille de l'écran sur lequel est visualisée l'application.
En effet, si l'utilisateur dispose d'un écran 15", il aura beaucoup plus d'aisance à visualiser un site, tous les éléments des pages auront de la place pour s'afficher alors qu'un écran plus petit, de mobile par exemple, il faudra jouer avec les fonctionnalités du téléphone afin de visualiser chaque partie de la page, grossissement, défilement horizontal et vertical...
C'est pourquoi le responsive design est devenu incontournable. Les technologies RWD permettent une adaptation des interfaces à l'espace dont elles disposent suivant le support sur lequel elles sont visualisées.
Le responsive design repose sur des concepts qui permettent une maniabilité plus aisée de l'interface. Il utilise les media queries en CSS3, ce qui permet d'appliquer des styles selon la condition du type de support et/ou la taille de l'écran.
Une page est organisée en bloc afin que le dimensionnemnet et la position de chacun soit modifiable en fonction du support, on parlera de grille de fluide.
Les ressources, images, sont également dimensionné en fonction de l'espace disponible.
Pour la création de sites et applications web, un des choix pertinent serait d'utiliser des outils qui faciliterait le responsive design et ainsi la plasticité des interfaces, pour cela, plusieurs alternative dont Twitter Bootstrap et Foundation from ZURB.

1. Responsive Web Design

1.2 Contexte d'usage

Le contexte d'usage est l'ensemble <plateforme, environnement, utilisateur>.
Pour les frameworks étudiés ici, les plateformes peuvent être très diverses, en effet, les frameworks permettent le développement de site web qui peuvent être visualisé sur différents supports, mobile, tablette... Ces frameworks permettent le développement de sites qui s'adaptent au format des supports.
Les deux frameworks demandent des connaissances minimale en HTML et CSS pour une bonne utilisation, cependant, Bootstrap fournit à l'utilisateur un environnement plus rapide et facile d'utilisation. L'utilisateur n'a pas forcément la nécessité d'avoir de connaissance approfondie des langages alors que Foundation fournit seulement une base de projet.
Ils utilisent tous les deux l'approche "Mobile-First" c'est à dire la construction du site pour un mobile puis qui s'adapte à des supports au format plus grand.

1.3 Moment

1.3.1 À la conception

Le but de tout framework est de faciliter le développement en fournissant à l'utilisateur un environnement de travail avec tout ce dont il aura besoin, notamment les bibliothèques.
Contenant les bibliothèques les plus utilisées pour le web et fournissant une base de projet RWD, Bootstrap et Foundation contribuent à donner à l'utilisateur un gain de temps et une facilité de développement dans la recherche et l'utilisation des bibliothèques.

1.3.2 À l'exécution

À l'exécution, les sites crées grâce à un framework RWD sont responsives, ils s'adaptent à la taille de l'écran afin d'augmenter le confort du client. Il sera donc plus aisé pour lui de cliquer sur un bouton car celui-ci sera plus gros ou d'une autre forme.
Bootstrap et Foundation ont le même but, adapter l'interface d'une application selon la taille de l'écran.

1.4 Twitter Bootstrap

Twitter Bootstrap a été développé par Mark Otto et Jacob Thornton de Twitter. Ce framework est sorti en août 2011 comme projet open-source sur Github, ce qui a permis à d'autres développeurs d'apporter leurs contributions.
Bootstrap fournit un ensemble d'éléments et de fonctions web-design personnalisables dans un seul est même outil. Ce qui permet à l'utilisateur de créer facilement un site. En effet, il n'a besoin que de télécharger les éléments en HTML, CSS et/ou Javascript qui lui définiront tout le design du site, puis éventuellement de les modifier si jamais il souhaite les personnaliser.
Ces pages de code peuvent être modifié par l'utilisateur, ainsi il garde la liberté et peut personnaliser le design.
Par défaut, les templates Bootstrap sont responsive ainsi l'utilisateur ne se préoccupe que de mettre ses données et modifier ce qu'il souhaite personnaliser.

En août 2013, Bootstrap 3 est sorti, il offre un tout nouveau design, des options de personnalisation plus avancées, une meilleure gestion des erreurs et surtout une refonte complète de la grille. Les projets écrits avec la version 2 doivent être réécrit complètement pour bénéficier des avantages de la version 3, mais peuvent cependant tourner sans modification de code.
Il propose des plugins jQuery, une bibliothèque et une documentation complète pour des modèles CSS. Less et Sass sont les deux préprocesseurs utilisés par Bootstrap pour compiler le code CSS. Ainsi le code CSS reste compatible avec les navigateurs et des variables sont créées depuis les propriétés CSS et peuvent être réutilisées. Les préprocesseurs font gagner du temps à l'utilisateur car fait toutes les choses fastidieuses à sa place tel qu'emboiter les selecteurs.
Less, écrit en Javascript, est utilisé pour garder un comportement dynamique. Sass, écrit en Ruby, est utilisé pour apporter des syntaxes au CSS comme l'emboitement de règles, fonctions...

1.5 Foundation from ZURB

ZURB Foundation est une collection d'outils gratuit open source pour la création d'applications webs créé en 2011. Il contient des bases de design pour la typographie, les boutons, la navigation...

Il a été le premier framework responsive et le seul à mener l'approche "Mobile-First" durant un certain temps.
En novembre 2013 est sorti la dernière version de Foundation, Foundation 5.
Foundation possède du HTML et du CSS basé sur des modèles et optionnellement une extension JavaScript. Il laisse à l'utilisateur l'oportunité de choisir précisément ce dont il a besoin afin de minimiser les téléchargements inutiles.

Tout comme Bootstrap, une communauté de développeurs contribue à Foundation et à son aide à l'utilisation. Ce qui se traduit notamment par la mise à jour régulière des feuilles de style par défaut qui applique un style aux balises HTML.
Il possède tout une série de feuille de style Sass en plus de modules. Les feuilles de style peuvent bien sûr être modifiées par l'utilisateur facilité par la syntaxe simple et intuitive des feuilles de style.
L'utilisation du framework peut se faire par l'intermédiaire de la ligne de commande.

Foundation a recour à une structure de grille afin de mettre en page les divers éléments et de s'adapter à tout type de support. Ainsi l'utilisateur n'a plus qu'à choisir les éléments pour chaque bloque et à les placer dans la grille. Ensuite, la grille place chaque bloque en fonction de la place disponible. Par exemple, sur un écran d'ordinateur les blocs se mettront en ligne de 5 alors que sur un mobile, les blocs se mettront les uns en dessous des autres pour faciliter la visibilité. Les développeurs peuvent adapter le fichier Foundation lui-même, en sélectionnant les composants qu'ils souhaitent utiliser dans leur projet.

1.6 Comparaison

Bootstrap et Foundation sont deux frameworks front-end. Ils permettent de mettre en page une interface de manière responsive.
Tout d'abord, Bootstrap et Foundation étant deux frameworks différents, il est normal de trouver des différences tels que la syntaxe même si elle est négligeable.
Foundation possède quelques fonctionnalités en plus tel que le block grid qui permet de mettre les différents contenus d'une liste non ordonnée dans un tableau dont on précisera le nombre de lignes souhaité. Mais aussi, le système de grille est à un nombre de colonne, pour Bootstrap il est fixé à 12 alors que Foundation permet de faire varier ce nombre de 1 à 16 et d'utiliser des demi-colonnes. Foundation a donc un plus grande précision dans le placement d'éléments.
L'unité de taille pour Bootstrap est le pixel alors que Foundation utilise les rems. Les

rems est une proportion alors que le pixel est fixe. Pour un framework RWD, l'unité des tailles peut avoir son importance car le rems s'adapte alors que le pixel ne le fait pas. Malgré tout, le même résultat peut être obtenu avec les deux systèmes d'unité.
Au sujet du style, Bootstrap possède des composants fluides et utilise la précompilation CSS ce qui permet la construction rapide d'un site web alors que Foundation fournit un environnement qui permet une personnalisation plus fine mais qui peut prendre plus de temps.
De plus, Bootstrap a des particularités de design qui permet au client de reconnaitre immédiatement qu'il a été fait par ce framework, alors que les thèmes de Foundation ne sont pas forcément reconnaissables au premier regard.
Comme dit précédemment, Bootstrap permet la construction d'un site rapidemment, en effet, Bootstrap possède des templates déjà définis prêts à l'emploi sans besoin de compétences particulières alors que Foundation demande un minimum de configuration pour lancer le projet.
Les navigateurs supportés sont les mêmes exceptés les versions d'Internet Explorer.
Bootstrap utilise deux préprocesseurs Less et Sass alors que Foundation n'utilise que Sass.

Voici un résumé de la comparaison :

	Bootstrap	Foundation
Créateur	Twitter	Zurb
Open source	Oui	
Navigateurs supportés	Firefox Chrome Opera Safari	
	IE8	IE9
Unité de taille	Pixels	Rems
Grid System	Fluide, 12 colonnes	Fluide, 1-16 colonnes
PréprocesseurCSS	Sass	
	LESS	
Semi-colonne	Non	Oui
Documentation	Complète avec exemples de code	
Efficaté	Différence négligeable	
Style	Style Bootstrap	Thème propre
Rapidité de mise en oeuvre	Rapide	Plus lent
Niveau de l'utilisateur	Basique	Avancée

1.7 Conclusion

Bootstrap et Foundation sont deux frameworks très semblables dans les fonctionnalités qu'ils proposent avec parfois des différences de moyens pour y parvenir mais rien n'empêche un des frameworks de réaliser ce que l'autre fait. Le choix parmis ces deux frameworks est plus une histoire de goût.

2 Synthèse sur les travaux de recherche

2.1 L'article

L'article choisi est "Users need your models ! Exploiting Design Models for Explanations". Dans cet article, le problème posé est comment apporter un outil d'aide à l'utilisateur sur une application.
En effet, l'utilisateur final n'a pas la même vision du système que le designer qui l'a créé.

2.1.1 Designer's model VS end users' model

Il est très important de faire la différence entre le modèle que le designer a mis en place et le modèle qu'imagine l'utilisateur final.
Si le designer pense un modèle pas du tout instinctif pour l'utilisateur, ce dernier ne pourra pas utiliser le système avec aisance, aura du mal à planifier les actions à réaliser pour aboutir au résultat escompté et donc chaque tâche sera très fastidieuse pour lui. Le système n'aura donc pas de succès, un autre sera mis à profit. D'où le besoin de se mettre à la place de l'utilisateur afin de penser un modèle et que le designer ait un modèle le plus proche possible de celui que se fera l'utilisateur lors de l'utilisation du système.
Pour parler de la différence entre les visions du designer et de l'utilisateur final, on utilisera le terme "Gulf of Quality".

2.1.2 Gulf of Quality

Le "Gulf of Quality" définit la distance entre le modèle créé par le designer de l'application et la représentation mentale qu'a l'utilisateur lors de l'utilisation de l'application. Cette intéraction entre la machine et l'utilisateur est définie par 7 étapes, perception, interprétation, évaluation, goal, intention, action et exécution. Dans les 3 premières étapes, c'est le "Gulf of Evaluation", l'utilisateur analyse le système et met en place un modèle de ce système dans son imagination qui décrit le comment fonctionne l'application selon ses observation.
Une fois ce modèle déterminé, l'utilisateur peut décider de ce qu'il souhaite faire et le mettre en action, c'est le "Gulf of Execution".

Afin de réduire ce “Gulf”, l'ingénieurie dirigée par les modèles [1] est employée.

2.1.3 Ingénieurie dirigée par les modèles

L'ingénieurie dirigée par les modèles créer et transformer des modèles dans le but de monter le niveau d'abstraction et ainsi de créer des modèles, métamodèle [2], métamétamodèle [3] et donc manipuler, générer et lier des fonctionnalités, applications plus facilement.
Dans cet article, MDE décrit 4 niveaux d'abstraction de modèle, Tasks, Abstract UI [4], Concrete UI et Final UI. L'interface utilisateur est obtenu par la définition des tâches puis par transformation “Top-down” des tâches jusqu'au code (Final UI).
CUI [5] fournit un premier niveau d'abstraction de l'interface finale, dépendante de la plateforme mais pas du langage. C'est-à-dire qu'on pourra avoir une CUI pour mobile ou ordinateur, mais le langage d'implémentation de l'interface finale n'est pas défini par ce niveau. AUI [6] quant à lui est un niveau d'abstraction au dessus de la CUI. Elle est complètement indépendante du support sur lequel sera implémenté l'interface. Elle définit les éléments nécessaires dans l'interface.
Le niveau de tâches est le plus abstrait, il définit les besoins de l'utilisateur et donc les interactions que l'interface devra pouvoir fournir à l'utilisateur, par exemple, éditer, copier, coller...
La FUI [7] est le niveau le moins abstrait, l'interface est implémentée.

2.1.4 Principes de design

Comme vu à la section précédente, l'interface est construite à travers plusieurs transformations du plus abstrait au plus concret. Le modèle des tâches étant le plus abstrait, c'est lui qui est la base de toute transformation.
Afin de créer une interface pour un système d'aide à l'application, cette interface aura besoin d'être construite de la même manière, c'est à dire par transformation top-down depuis le modèle de tâches car elle a besoin d'être construite de la même manière que l'application pour pouvoir réaliser des requêtes et trouver les informations demandées par l'utilisateur.

1. MDE : Model Driven Engineering en anglais
2. Modèle de modèle
3. Modèle de métamodèle
4. User Interface
5. Concrete User Interface
6. Abstract User Interface
7. Final User Interface

Dans les modèles, les réponses aux questions que peut poser l'utilisateur doivent être défini et facilement retrouvable au moment de la question de l'utilisateur, d'où l'utilité d'avoir la même architecture de modèle entre l'application et le système d'aide. Une fois la réponse trouvé, il faut la convertir dans un format compréhensible par l'utilisateur.
Lors de la recherche de la réponse, le lien entre le système d'aide et l'application peut se faire à différent niveau. S'il se fait dans un niveau abtrait, seul une requête sera lancé, mais la réponse sera plus approximative que si le lien se fait à un bas niveau mais dans ce cas là, deux requêtes seront lancées pour trouver une réponse précise.

2.2 Contextes d'usage

Le contexte d'usage est l'ensemble <plateforme, environnement, utilisateur> possible. En effet, il peut en exister une multitude car de nos jours les plateformes sont de plus en plus nombreuses et de plus en plus diverses et les utilisateurs sont tous différents, pensent de manière différente et n'ont donc pas la même vision des systèmes.

Dans cet article, le problème posé est "comment fournir un système d'aide ?". Le problème du contexte est soulevé dans les premières lignes. En effet, la vision du designer est différente de celle de l'utilisateur et les visions des utilisateurs sont différentes entre elles car très souvent basée sur des expériences passées d'où l'intérêt de réaliser des interfaces génériques.
Les contextes d'usage pouvant être très variés et en nombre incalculable, il est impossible de réaliser des interfaces personnalisées pour chacun, cependant elles doivent pouvoir s'adapter aux contextes. C'est pourquoi, on soulèvera ici la notion de système d'aide **dynamique**.
La solution proposée dans cet article est l'utilisation d'une architecture générique MDE [8] lors de la conception afin que l'extraction et l'exploitation des informations nécessaires à l'utilisateur se fasse depuis les modèles au moment de l'exécution. Les réponses seront donc en adéquation avec le support sur lequel s'exécute le programme.
De plus, le problème posé par les différentes plateformes disponibles est résolu grâce à l'architecture générique mise en place. En effet, l'architecture possède quatre niveau, dont deux (niveau des tâches et AUI) étant totalement indépendants de la plateforme, ce qui diminue considérablement le travail d'adaption.

8. Model-Driven Engineering

2.3 Moment

2.3.1 À la conception

Le problème posé à la conception est “Comment le designer peut-il réaliser une interface compatibles avec les différents supports et compréhensible par l'utilisateur ?”.
Ce problème est en réalité multiple. En effet, divers supports impliquent diverses conceptions, mais le coût de toutes ces conceptions est élevés et plus il y a de supports, plus le coût s'élève.
La solution proposé dans cet article est donc judicieuse, car l'utilisation d'une architecture générique permet un gain de temps, en effet avec les différents niveaux d'abstraction, la même base est réutilisée pour les différents supports.

2.3.2 À l'exécution

Dans cet article, le problème posé est aussi par rapport à l'utilisateur. Gagner en coût de développement et faciliter le travail du développeur ne suffit pas, ça ne sert à rien si l'utilisateur a des difficultés à utiliser le système.
C'est pourquoi cet article parle tout d'abord de la notion de “Gulf of Quality”, on veut se rapprocher de l'image du système qu'a l'utilisateur final afin qu'il ait la plus grande aisance d'utilisation. Le designer a besoin d'essayer de penser l'application qu'il est en train de développer du point de vue de l'utilisateur final. En effet, si le designer pense comme l'utilisateur, les visions du système seront très semblables et l'utilisateur n'aura aucun mal à l'utiliser.
Cependant, les utilisateurs sont très divers c'est pourquoi cet article présente une solution en essayant de garder les points redondants dans la majorité des systèmes d'aide existants. Par exemple, le fait que le système d'aide soit dans une fenêtre séparée de l'application.
La solution proposée à quatre niveaux d'abstration est très personnalisable, ainsi les divers supports peuvent accueillir l'application et en tirer les bénéfices du support en ne modifiant que les niveaux les plus bas.

2.4 Les solutions

2.4.1 Présentation

Cette article présente une solution au problème posé. C'est-à-dire, comment organiser un système d'aide d'une application.
Comme vu dans les principes de design, l'application et le système d'aide sont construit de la même manière. Ils sont conformes au même métamodèle, ce qui leur permet une communication aisée.
On peut voir l'architecture proposée pour ce problème en figure 2.1.
L'utilisateur final communique avec les interfaces finales, l'interface d'aide et l'interface de l'application qui sont reliées à ce niveau là.
Chacune de ses interfaces sont générées par transformation de niveau d'interface plus abstraite, CUI, AUI et Task par un controleur indépendant pour chaque interface finale.
Cependant, les controleurs sont reliés à un bloc central où sont définis les modèles et métamodèles afin que les deux interfaces finales soient conformes à la même structure.
Ce bloc central définit les niveaux, transformations entre niveaux et les métamodèles.

Pour ce qui est du fonctionnement du système d'aide, l'utilisateur pose sa question à travers l'interface d'aide auquel il a accès. Cette action lance une requête au controleur qui la transmet au service interprète. Ce dernier la comprend et dit au processor service quelle information est demandée. Le processor service accède aux modèles et recherche l'information, il peut la rechercher dans n'importe lequel des modèles indépendamment grâce à la symétrie de structure entre l'application et le système d'aide.
Une fois l'information trouvée, le Generator Service prépare la réponse pour l'utilisateur, il la transforme afin qu'elle soit compréhensible par ce dernier. Elle peut être de différent type, langage naturel, animation...
L'interface est alors mise à jour par le controleur afin d'informer l'utilisateur de la réponse donnée par le Generator Service.

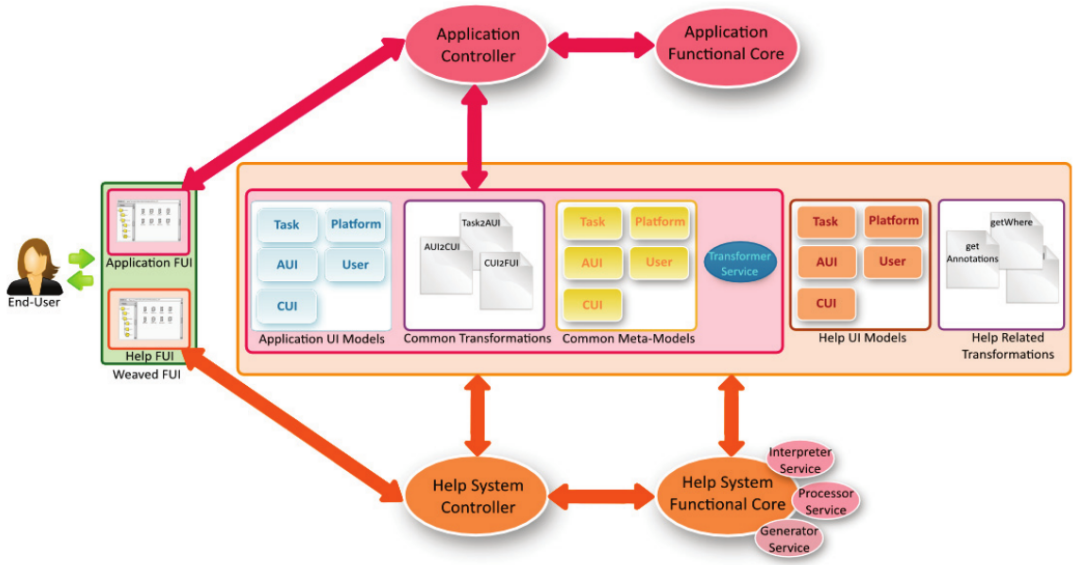

FIGURE 2.1 – Architecture Générique pour le Model-Driven du système d'aide

2.4.2 Exemple

Cet article présente également un exemple d'implémentation à travers l'IDE [9] UsiComp. Cet IDE permet la définition des tâches (le plus haut niveau d'abstraction) puis AUI, CUI et enfin générer l'interface finale.

UsiComp permet à l'utilisateur, à travers un éditeur de créer tous les modèles dont il a besoin pour définir son application.

Pour donner un exemple concret d'implémentation d'un système d'aide, voici tous les modèles dans l'éditeur UsiComp.

9. Integrated Development Environment

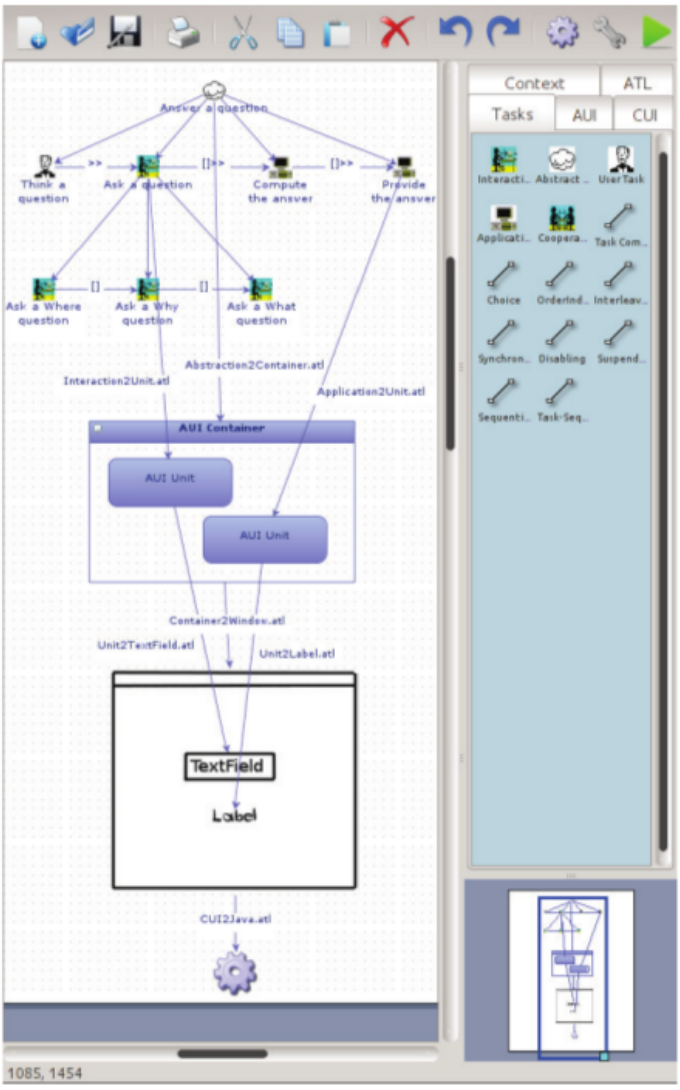

FIGURE 2.2 – Tous les modèles nécessaires au système d'aide

On peut voir le premier niveau de tâches tout en haut. Il explicite les besoins en terme de fonctionnalité à développer. Le but principal est répondre à une question qui se décline en "l'utilisateur pense à une question", "il l'a pose", elle peut être de trois types : Où, Pourquoi ou Quelle. Ensuite il faut rechercher la réponse et la fournir à l'utilisateur.

Le niveau du dessous est une première concrétisation du niveau de tâches, c'est l'AUI. Il représente les éléments nécessaires et principaux au développement du système d'aide. Ici, le but principal est "répondre à une question", tout est contenu à l'intérieur d'où la

représentation de cette tâche par un container qui englobe tout le reste.
Ensuite, les deux tâches en relation avec l'utilisateur est "Poser une question" et "Fournir une réponse", d'où la représentation de ces deux tâches par deux unités à l'intérieur du container.

Pour concrétiser un peu plus cette application, on aura la couche CUI. Cette couche détermine les éléments qui devront être employé afin de fournir à l'utilisateur les outils dont il a besoin. Dans ce cas, nous auront un TextField pour qu'il puisse poser sa question et un Label pour y présenter la réponse mais ça pourrait être sous une autre forme (selection dans une combobox du type de la question et clique sur un clavier virtuel pour finir la question par exemple, la réponse peut être donnée avec la synthèse vocale également). Tout ceci dans une fenêtre qui représente le container.
Ce niveau est indépendant du langage dans lequel sera implémenté l'application mais dépendant de la plateforme. C'est-à-dire que ce système fonctionne pour une plateforme Desktop mais s'il s'agit de Mobile, on ne parlera plus de fenêtre mais d'écran.

Enfin, le dernier niveau, on a l'application, produit par génération de code à partir du modèle CUI et exécution. On obtient ici un modèle de niveau M1 la pyramide des quatre niveaux. En effet, on a le modèle de plus bas niveau possible. Si on exécute l'application, c'est du niveau M0, données utilisateur.

On pourra lors de l'exécution de l'application avoir un type de système d'aide telle que celui-ci :

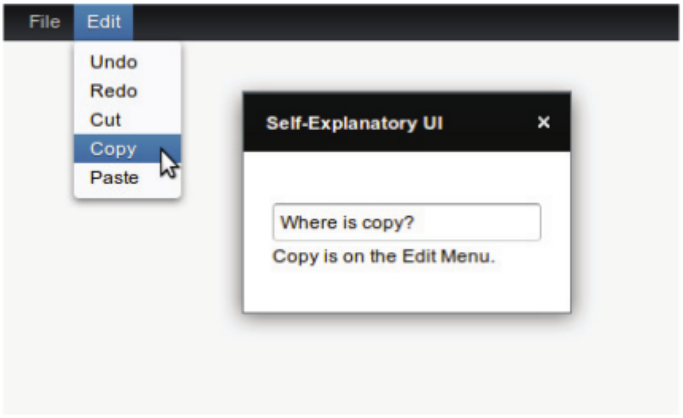

FIGURE 2.3 – Exemple d'interface d'aide lié au niveau CUI

Si on demande où est le bouton "Copy", le système répondra dans le Menu Edit car c'est le niveau où l'on détermine que "Edit" sera une liste déroulante et que l'un de ses

items sera “Copy”.

2.5 Autre exemple

On peut imaginer d’autres exemples conforme aux 4 couches d’abstraction et le Model-Driven du système d’aide.
L’interface graphique peut aisément être modifié, au lieu de donner un champs texte à l’utilisateur pour qu’il pose sa question, on peut lui proposer une série de mots interrogatifs ainsi que l’ensemble des éléments de l’application. L’utilisateur n’aurait qu’à choisir et le système d’aide fonctionnerait également. Les couches différentes (par rapport à l’exemple précédent) seraient la AUI, la CUI et la FUI.

Un autre exemple un peut plus éloigné peut être imaginé, donner à l’utilisateur la possibilité de changer le style de son application par l’intermédiaire d’une fenêtre externe.
Dans ce cas, la couche de tâches sera différente, en haut on aura “Changer le style”, l’utilisateur pensera à un élément à changer ainsi qu’à une couleur puis demandera ce changement. Le système récupèrera la pétition de l’utilisateur, l’appliquera si possible et retournera à l’utilisateur si oui ou non les changements ont été effectués.
La couche AUI sera composé d’un container (pour la fenêtre) et de trois unités, une pour l’élément que l’utilisateur souhaite modifier, une pour la couleur demandée et une troisième pour le retour à l’utilisateur de l’application du style.
La couche CUI pourra contenir une Window, deux champs TextField et un Label.
La couche FUI quand à elle est générée automatiquement.

Cet exemple fonctionnerait également avec le Model-Driven du système d’aide car l’utilisteur demande quelque chose, la pétition est analysée et est demandée à la même couche où le système de style et l’application sont liés. Les deux applications sont conçut à la base du même modèle donc la communication entre les deux est possible.
Lors de l’exécution, la différence avec l’exemple de la partie précédente est que plus le lien est haut dans la pyramide, plus les couleurs s’appliqueront à un plus grand ensemble.
Par exemple, si l’utilisateur choisi de mettre en rouge tout ce qui est liée à la tâche “copier”, tous les moyens visuels de copier seront mis en rouge, en revanche si l’on est lié à la couche la plus concrète, seul les éléments (le champs “Copier” de la toolbar par exemple) précisés par l’utilisateur sera mis en rouge.

2.6 Avis

2.6.1 Avantages

La solution apportée par cet article a de nombreux avantages.
Tout d'abord, la flexibilité sur comment et où le système d'aide et l'application sont liés. Le lien peut se faire à un des quatre niveaux d'abstraction ce qui donne des réponses différentes aux questions que peuvent poser les utilisateurs.
La distribuabilité, le fait d'avoir quatre niveaux d'abstraction permet de développer le système depuis le même métamodèle pour différentes plateformes. Et donc avoir une compatibilité entre support.
Si une fonctionnalité est déjà défini dans un système, elle peut être réutilisé dans un autre système à partir du moment où ce dernier est conforme au même métamodèle. Le design du système d'aide étant défini au niveau CUI, les designers peuvent reprendre toutes la partie tâches et AUI et ne changer que le style. Le designer gagne en coût de développement.
De plus, aucune restriction n'est faite sur la forme de communication avec l'utilisateur. Dans l'exemple de cet article, l'utilisateur pose une question en langage naturel, mais ce n'est pas obligatoire, il peut trouver les informations désirées par d'autres moyens si le designer le lui propose.

2.6.2 Inconvénients

Malgré tous les avantages qui peuvent être trouvés, certains inconvénients sont là. Par exemple, le fait d'avoir plusieurs couches et de les lier au niveau le plus abstrait, le nombre d'éléments de l'interface n'est pas accessible depuis ce niveau. De ce fait, lorsqu'une requête d'aide est lancée, elle passe par la couche des tâches et doit refaire une requête pour connaitre la réponse précise du niveau FUI.
Ce modèle à quatre couches parait très modulable et donne beaucoup de liberté au designer, mais sa mise en oeuvre en devient plus difficile. Un fois ce modèle choisi, il est difficile d'en changer, toute la structure serait à refaire. C'est pourquoi, si le designer choisi cette structure, il doit être sûr qu'il correspondra à ses besoins.