UNIVERSITE CATHOLIQUE DE LOUVAIN
INSTITUT D'ADMINISTRATION ET DE GESTION

Développement d'un maquetteur d'interfaces à l'aide de Microsoft Visio.

Directeur : Prof. J. Vanderdonckt

Mémoire-projet présenté par
Manuel Van Sluys

En vue de l'obtention du titre d'
Ingénieur de gestion

Mes remerciements iront tout d'abord à mon promoteur, le Pr. J. Vanderdonckt pour m'avoir proposé ce mémoire qui, bien que difficile à mon sens, n'en fut pas moins passionnant dans ses phases de réflexion.

Je tiens également à remercier le Pr. M. Saerens et les assistants de l'IAG, Mrs. L.Bouillon, A. Coyette et Q. Limbourg de m'avoir guidé dans les méandres tumultueux de la programmation itérative, aussi basique ait elle pu être.

Enfin, ma reconnaissance va à tous ceux qui m'ont patiemment et gracieusement offert leur temps, leurs informations et leur soutien, Mrs. B. Michotte, assistant à l'IAG, Y. Vanden Berghe, étudiant en informatique à l'UCL, M. de Lacheisserie et M. Mouraux, mes amis, ainsi que tous les relecteurs de ce mémoire.

Van Sluys Manuel INGE23

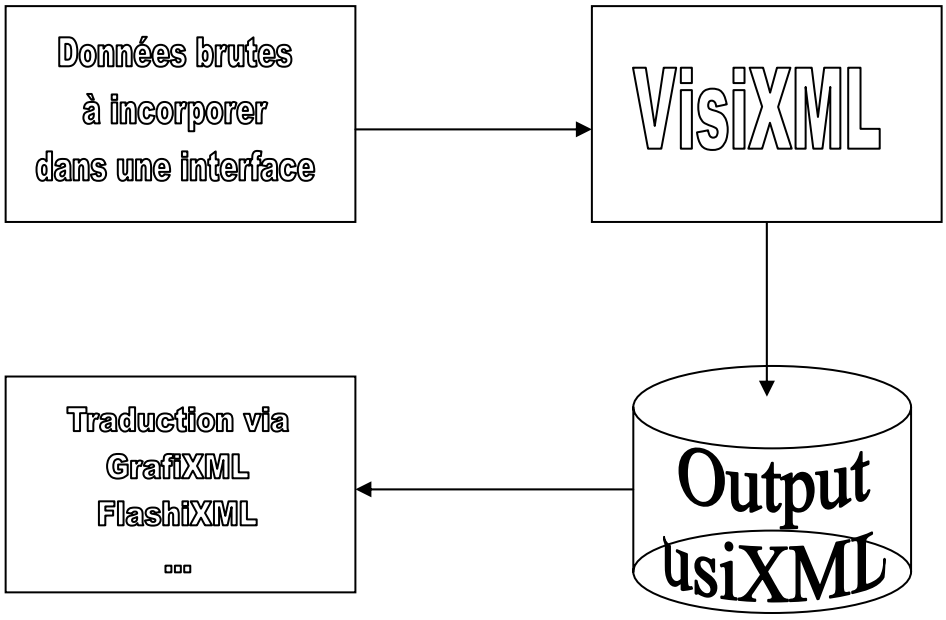
2003-2004

Développement d'un maquetteur d'interfaces à l'aide de Microsoft Visio.

A l'heure où le recours aux technologies de l'information se généralise dans nos pays industrialisés, un besoin de génération de contenu sous une forme adéquate et de manière rapide se fait de plus en plus sentir. C'est dans ce contexte qu'intervient la problématique du développement d'interfaces, celles-ci devant présenter les délais de création les plus courts possibles tout en respectant un niveau de fidélité élevé.

L'objectif de ce mémoire est donc de fournir aux développeurs un outil de prototypage répondant au mieux à leurs attentes, cet outil est baptisé VisiXML et permet de créer rapidement des spécifications d'interfaces en usiXML.

Le principe de création d'une interface en passant par un prototype en usiXML est le suivant : les données brutes qui sont destinées à être transformées en une interface graphique ou un formulaire online sont introduites dans VisiXML via le « drag and drop » de shapes et la spécification des différents attributs de celles-ci. Ces données sont ensuite transformées en spécifications usiXML valides, spécifications qui seront finalement envoyées à un éditeur ou un traducteur de manière à produire l'interface correspondant aux données initiales dans le langage désiré :



De cette manière, sur base d'un seul et même encodage des données initiales dans VisiXML on peut fournir, sans effort supplémentaire, l'interface correspondante dans tous les langages pour lesquels un traducteur usiXML existe.

Afin de définir clairement les fonctions que VisiXML doit remplir, le mémoire est structuré de la manière suivante :

- L'introduction explique de manière précise la problématique des systèmes d'information et l'intérêt du prototypage au sein de cette problématique, en en détaillant le principe.
- Le chapitre 2 présente un état de l'art représentatif de ce qui se fait en matière d'outils de développement, l'analyse de la littérature permettant de voir ce qui a déjà été fait pour essayer de l'améliorer par la suite. Une grille de critères est donc établie afin de juger des différents outils existants et d'identifier les caractéristiques importantes de chacun de ces outils afin de les incorporer dans le développement de VisiXML.

- Le chapitre 3 introduit les technologies XML sur lesquelles se basent VisiXML, on y retrouve donc une description du metalangage XML en lui-même ainsi que de son extension développée au sein de l'unité ISYS, le langage UsiXML. Ce chapitre définit la mesure dans laquelle VisiXML supporte les objets et attributs d'UsiXML, que ce soit au niveau des spécifications ou de l'aperçu visuel qu'a l'utilisateur du générateur. On y retrouve donc la description complète des spécifications usiXML, qui sont les objets et contrôles qui serviront de base à VisiXML.
- Le chapitre 4 détaille les différentes étapes du développement du maquetteur d'interface et les différents choix de développement qui ont été faits.
- Dans le cinquième chapitre le lecteur pourra trouver une étude de cas détaillant la création d'une partie d'interface avec VisiXML depuis les données à y incorporer qui se trouvent sur un formulaire papier jusqu'aux spécifications valides vérifiées par le biais d'un éditeur ou d'un traducteur UsiXML.
- La conclusion du mémoire présente les possibilités de développements futurs de VisiXML.

Table des matières

1. Introduction

A l'heure où le recours aux technologies de l'information se généralise dans nos pays industrialisés, un besoin de génération de contenu sous une forme adéquate et de manière rapide se fait de plus en plus sentir. C'est dans ce contexte qu'intervient la problématique du développement d'interfaces, celles-ci devant présenter les délais de création les plus courts possibles tout en respectant un niveau de fidélité élevé. De plus, il est intéressant de fournir des moyens de génération accessibles aux plus grand nombre, ce qui permettra aux entreprises de ne pas mobiliser leur personnel qualifié pour le développement de toutes leurs interfaces, même les plus basiques.

1.1. Les technologies de l'information

Pour comprendre la généralisation du recours aux technologies de l'information, il faut tout d'abord se pencher sur ce qu'elles peuvent apporter aux entreprises (aux particuliers aussi, mais dans une moindre mesure). Les avantages des technologies de l'information sont nombreux, et certains de ceux-ci sont repris par Kenneth C. Laudon dans son livre « Management Information Systems » [Lau01] :

- «Les réseaux internationaux sont une des technologies de l'information qui permet la division du travail à l'échelle mondiale : les opérations de l'entreprise ne sont plus déterminées par sa situation géographique ; l'entreprise prend de l'envergure mondialement ; le coût d'une coordination mondiale est plus bas et les coûts de transaction chutent. »
- Les réseaux d'entreprise, eux, favorisent le travail de collaboration et le travail d'équipe : il est maintenant possible de coordonner l'organisation du travail au-delà des frontières de chaque division ; il en résulte une orientation vers le client et vers les produits ; des groupes de travail très dispersés prennent une position dominante et les coûts de la gestion (coûts d'agence) diminuent. Les processus d'affaires changent.

- L’informatique distribuée permet une certaine habilitation du personnel : les personnes et les groupes de travail peuvent prendre des initiatives car ils disposent maintenant des informations et des connaissances qui leur sont nécessaires. On repense les processus d’affaires et on les rationalise. Les frais de gestion diminuent, de même que la hiérarchisation et la centralisation des organisations.
- L’informatique mobile permet les organisations virtuelles ; le travail n’est désormais plus lié à un emplacement en particulier. Les connaissances et les informations arrivent là où elles sont nécessaires, sans aucune restriction de temps. Le travail devient mobile, les coûts diminuent et les biens immobiliers revêtent de moins en moins d’importance.
- Les interfaces graphiques permettent l’accessibilité : tous les membres d’une organisation, même les cadres supérieurs, peuvent accéder aux informations et aux connaissances ; les opérations sont automatisées ; tous les employés, même ceux qui se trouvent dans des endroits éloignés, contribuent à l’essor de l’entreprise. Les coûts diminuent lorsqu’on passe du papier à l’image, aux documents et aux enregistrements vocaux numérisés.

Au vu de ces différents avantages, on peut aisément comprendre la grande progression de la demande de systèmes d’informations, ces derniers devant être de tailles diverses et devant répondre à certaines spécifications bien définies par les besoins des utilisateurs.

1.2. Le prototypage

Une des phases les plus importantes du processus de développement d’un système d’information est le prototypage. Elle va permettre aux développeurs de rencontrer aux mieux les attentes des utilisateurs en ce qui concerne les spécifications des systèmes d’information. Afin de comprendre pourquoi nous allons en détailler le principe, les

définitions des différents concepts de ce chapitre sont librement traduites des travaux de Mr. Saul Greenberg [Gree98] :

Le prototypage consiste en la création de prototypes, qui sont des designs d'interface expérimentaux et incomplets créés rapidement et à peu de frais. Le prototypage fait partie intégrante du design de systèmes centré sur l'utilisateur car il permet aux designers d'essayer leurs idées avec les utilisateurs et de récolter un certain feedback de la part de ceux-ci. Le but principal du prototypage est en effet d'impliquer les futurs utilisateurs de l'interface dans la phase de test des idées de design et de récolter leur feedback lors des phases préparatoires du développement. C'est donc un processus idéal lorsqu'il s'agit de raffiner ou d'optimiser des interfaces.

Trois catégories principales de prototypage sont admises dans la littérature, par degré croissant de fidélité : basse, moyenne et haute. La fidélité en ce qui concerne les prototypes est définie comme étant le degré de précision avec lequel le prototype représente l'apparence et l'interaction de l'interface. Il ne s'agit donc pas ici de juger si le code ou tout autre attribut invisible pour l'utilisateur sont précis.

1.2.1. Le prototypage de basse fidélité

Les prototypes de basse fidélité sont créés très rapidement et servent principalement à donner une idée des alternatives de design possibles, de l'agencement des différents éléments à l'écran et des concepts repris dans l'interface. Ils sont créés afin de donner aux utilisateurs un aperçu de l'apparence globale de l'interface et en aucun cas ne peuvent servir de base au codage, au test ou à la définition détaillée des mécanismes opératoires cachés de l'application. Ce genre de prototype est généralement constitué d'un schéma papier reprenant les différents éléments d'une interface mais ce n'est pas une solution unique. Voici quelques-unes des possibilités de prototypage basse fidélité :

1.2.1.1. Le schéma sur papier

Les schémas dessinés à la main sur une feuille de papier constituent, comme nous venons de le dire, la majeure partie des prototypes de basse fidélité. Ils sont essentiels pour cristalliser les idées dans les premiers stades du design et le simple fait de poser leurs idées sur papier permet en général aux designers de voir de nouvelles relations et/ou de nouveaux dispositifs conduisant à un raffinement, voire une mise à jour de leurs idées initiales. Ce sont des prototypes fort utiles pour les premiers stades du développement, car ils sont rapides et peu coûteux à produire ; néanmoins ils sont à déconseiller pour les stades ultérieurs où l'important sera, pour le développeur, de tester l'interaction et la consistance de son design.

1.2.1.2. Le storyboard

Les storyboards sont des descriptions graphiques de l'apparence extérieure du système mais qui ne sont accompagnées d'aucune fonctionnalité de ce dernier. Ils photographient en quelque sorte l'interface à un moment donné de l'interaction avec l'utilisateur de manière à ce que ce dernier puisse juger rapidement de la qualité du design à cette étape précise de l'interaction.

Pour donner un exemple, un storyboard pourrait par exemple se constituer de deux schémas faits à la main et représentant chacun un écran. Sur le premier schéma on verrait un écran de formulaire, avec des labels et des champs d'édition ainsi que trois boutons standards, dont notamment un bouton « save ». On ferait alors partir une flèche de ce dessin du bouton « save » qui irait vers le schéma de l'autre écran, et celui-ci représenterait alors l'écran qui apparaîtrait lorsque l'utilisateur appuie sur ce bouton, à savoir un simple écran de confirmation avec un label « Save work in progress ? » et deux boutons, « yes » et « no ».

Les storyboards sont donc en réalité constitués d'une série de schémas sur papier reliés entre eux par des flèches, celles-ci représentant la succession des écrans de

l'interface en fonction des interactions de l'utilisateur avec tel ou tel objet de l'écran précédent.

1.2.1.3. PICTIVE

PICTIVE est l'acronyme de "*Plastic Interface for Collaborative Technology Initiatives through Video Exploration* ». C'est un procédé de prototypage développé par Bell Communications Research en 1990 pour promouvoir le design participatif.

Comme son nom l'indique, les prototypes réalisés avec PICTIVE comprennent des éléments en plastique coloré et une caméra, mais également des éléments de bureautique standard tels que papiers et Post-It™ de couleur, autocollants, trombones, etc. Le principe est simple : tout d'abord les utilisateurs sont invités à réfléchir sur ce qu'ils voudraient que le système fasse pour eux et sur les étapes nécessaires à l'accomplissement de cette tâche. Les développeurs sont eux invités à construire sur base d'une discussion préliminaire avec les utilisateurs un ensemble de composants plastiques que ceux-ci pourront manipuler. La seconde étape consiste à asseoir les utilisateurs et les développeurs tous ensemble face à la caméra, lors d'une session où chacun pourra participer à la création du design en plaçant les différents objets dans telle ou telle position, en inter changeant les couches d'objets, en dessinant des composants à l'aide d'un stylo, etc. Toute la session est filmée et les voix enregistrées, et c'est cet enregistrement qui servira par la suite de présentation dynamique du design.

Le grand avantage de ce procédé, qui est également le but recherché par ses concepteurs, est son interactivité couplée à l'égalité des chances qu'il engendre. En effet, contrairement à d'autres procédés de prototypage assisté par ordinateur, tous les intervenants peuvent placer leur grain de sel et le design final n'est pas complètement influencé par le développeur, qui est celui qui dans ces cas maîtrise le mieux l'outil. Il est à noter que PICTIVE ne sera réellement utile que dans les cas où les utilisateurs comprennent et sont conscients de ce que le système devra leur apporter.

1.2.2. Le prototypage de moyenne fidélité

Les prototypes de moyenne fidélité sont des outils qui, en plus de présenter un aperçu du design de l'interface, simulent partiellement l'interaction et la fonctionnalité du système. Ils en existe également plusieurs types :

1.2.2.1. La simulation assistée par ordinateur

Il s'agit de prototypes informatisés qui simulent ou animent quelques-uns mais pas tous les dispositifs du système prévu. Il y a trois approches en ce qui concerne la limitation des fonctionnalités prises en charge par le prototype :

- le prototypage vertical : le prototype informatisé n'inclut pas tous les dispositifs du système mais ceux qu'il inclut le sont entièrement. L'utilisateur peut donc avoir un accès en profondeur à seulement quelques-unes des fonctions du système.
- Le prototypage horizontal : ici c'est l'inverse, l'utilisateur a accès à toutes les fonctions du système, mais de manière superficielle uniquement. Il peut donc apercevoir tous les dispositifs mis en place dans l'interface mais ne peut accéder aux fonctionnalités qui se trouveront derrière.
- Le scénario : c'est en quelque sorte un mélange des deux approches précédentes, en ce sens qu'il limite à la fois le nombre de dispositifs et le niveau de fonctionnalité. Dans le cas d'un scénario, le prototype simule l'interface aussi longtemps que l'utilisateur suit un chemin prédéfini (si le scénario est prévu pour montrer le menu « Edit » rien ne se passera si l'utilisateur essaie de clicker sur le menu « Tools » par exemple).

1.2.2.2. Le slide show et le prototypage vidéo

Ces deux techniques utilisent le média pour faciliter la création du prototype. Dans le slide show, le prototype papier de type storyboard est simplement encodé dans un

ordinateur avec une simple activation de la transition entre les différents schémas d'écran par un input de l'utilisateur, un click par exemple. Les slides peuvent dès lors former soit un prototype horizontal (si l'on a par exemple dessiné tous les écrans correspondants aux items d'un menu) soit un prototype vertical (si l'on a dessiné les écrans successifs qui apparaissent lors de l'exploration poussée de l'une des fonctions du menu).

Le principe du prototypage vidéo est assez semblable bien que peut-être plus fastidieux ; il s'agit de reprendre les schémas sur papier qui ont été faits pour une interface et de bouger par-dessus ceux-ci un objet représentant la souris. A chaque fois que la souris atteint l'élément choisi sur le schéma papier, on coupe l'enregistrement vidéo, on remplace le schéma de l'écran par l'écran suivant et on relance la vidéo, ce qui donnera l'impression que le changement d'écran a été initié par un click de souris.

Ces deux types de prototypage sont destinés à simuler une partie très restreinte du système et paraissent se comporter exactement comme le système, bien qu'ils n'en comportent que quelques scènes. Ils ne permettent aucune interaction avec l'utilisateur.

1.2.3. Le prototypage de haute fidélité

Il s'agit cette fois de prototypes informatisés telles que les simulations de moyenne fidélité, à la différence que tous les dispositifs de l'interface y sont représentés ainsi que les fonctionnalités correspondantes. Ce sont donc en quelque sorte des interfaces navigables qui, mises ensemble, peuvent constituer un système fonctionnel apte à subir un alpha testing.

L'outil que nous allons tenter de développer dans ce mémoire fournira des prototypes de moyenne fidélité, de type horizontal, c'est-à-dire représentant tous les dispositifs de la future interface mais sans fonctionnalités derrière. Cet outil, baptisé VisiXML a pour objectif de fournir un moyen de développement d'interfaces rapide et nécessitant peu, voire pas de connaissances spécifiques. Il se basera sur le langage de spécifications UsiXML ; ce choix sera explicité par la suite.

On peut avoir ci-dessous (Fig.1.1) un aperçu du cycle d'utilisation de VisiXML en tant qu'outil de prototypage. Les données brutes qui sont destinées à être transformées en une interface graphique ou un formulaire online sont introduites dans VisiXML via le « drag and drop » de shapes et la spécification des différents attributs de celles-ci. Ces données sont ensuite transformées en spécifications UsiXML valides, spécifications qui seront finalement envoyées à un éditeur ou un traducteur de manière à produire l'interface correspondant aux données initiales dans le langage désiré :

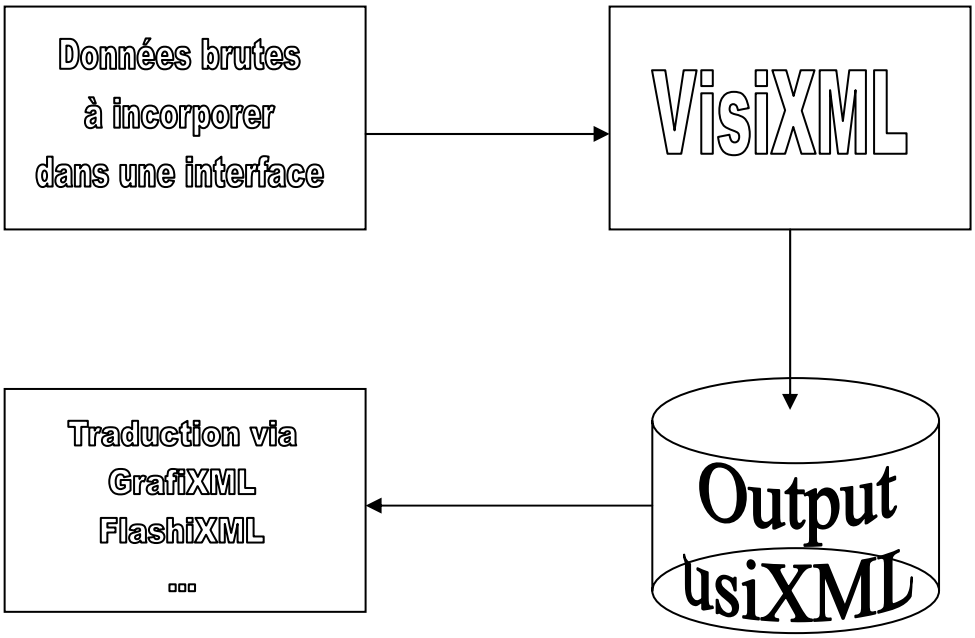


Fig.1.1 : cycle d'utilisation de VisiXML

De cette manière, sur base d'un seul et même encodage des données initiales dans VisiXML on peut fournir, sans effort supplémentaire, l'interface correspondante dans tous les langages pour lesquels un traducteur UsiXML existe.

1.3. VisiXML

C'est donc dans le contexte avantageux du prototypage que VisiXML devra trouver sa place. La génération des interfaces actuelles devant être la plus rapide et la plus

précise possible, tout en ne nécessitant pas de compétences exceptionnelles afin d'en permettre l'accès au plus grand nombre, VisiXML présentera trois caractéristiques principales qui devraient lui permettre de répondre au mieux à ces attentes :

1.3.1. Gain de temps lié au « drag and drop »

VisiXML sera tout d'abord un outil de « drag and drop », ce qui réduit les délais de création par rapport à la méthode de génération classique d'interfaces (totalement manuelle) tout en garantissant un certain niveau de fidélité, ce qui ne serait peut-être pas le cas avec des méthodes de génération encore plus rapides telles que le sketching. Ces différentes méthodes seront analysées de manière plus approfondie dans le chapitre 2.

1.3.2. Un autre niveau de représentativité

Les spécifications UsiXML composeront l'output de VisiXML. Elles présentent l'avantage de ne pas être complètement descriptives, ce qui facilitera leur traduction ultérieure en d'autres langages au moyen de traducteurs. On remarque ici encore un avantage non négligeable au niveau du gain de temps, qui sera qu'il ne faudra qu'un seul développement de l'interface pour en avoir une version dans tous les langages compatibles avec UsiXML. Pour toute information relative aux caractéristiques d'UsiXML, on se réfère dans ce mémoire aux données disponibles sur le site www.UsiXML.org .

1.3.3. Cible, environnement et différences de VisiXML

La dernière caractéristique principale de VisiXML est sa cible particulière. Comme dit précédemment, VisiXML vise à fournir même aux non-developpeurs la possibilité de générer des interfaces via UsiXML. Il s'agit donc bien d'un prototypage et de spécifications de moyenne fidélité mis à la portée du plus grand nombre, et ce par le

biais d'une méthode de « drag and drop » qui, en plus d'être rapide, se veut la plus intuitive possible.

1.4. Objectif du mémoire

L'objectif de ce mémoire est de fournir aux développeurs un outil de prototypage répondant au mieux à leurs attentes. Pour ce faire, le chapitre 2 sera constitué d'un état de l'art représentatif de ce qui se fait en matière d'outils de développement, l'analyse de la littérature permettant de voir ce qui a déjà été fait pour essayer de l'améliorer par la suite. Une grille de critères sera donc établie afin de juger des différents outils existants et d'identifier les caractéristiques importantes pour VisiXML. Le chapitre 3 reprendra la description complète des spécifications UsiXML, qui sont les objets et contrôles qui serviront de base à VisiXML. Dans le chapitre 4 nous détaillerons les différentes étapes du développement du maquetteur d'interface et les différents choix de développement qui ont été faits. Le chapitre 5 sera constitué d'exemples de génération d'interfaces sur base de formulaires administratifs, c'est-à-dire que sur base de formulaires papiers standards nous essayerons de fournir des spécifications UsiXML que nous visualiserons au moyen d'un éditeur (GrafiXML) et d'un interpréteur (FlashiXML). Enfin, la conclusion de ce mémoire présentera quelques remarques sur l'outil et l'intégration de celui-ci à son environnement, en plus de quelques pistes de réflexion pour les développements futurs.

2. Etat de l'art des outils de développement

Ce chapitre a pour objectif de fournir au lecteur un aperçu de ce qui existe en matière d'outils de développement d'interfaces. Il commencera par la distinction des 3 grands types d'outils existants, dans lesquels seront choisis un certain nombre d'outils jugés représentatifs de leur catégorie. Je présenterai alors les différents critères choisis pour élaborer une grille d'analyse aussi complète que possible qui servira à positionner les différents outils au sein de la fonction de développement prise au sens large. A la lumière des résultats de cette analyse multicritères, j'essaierai enfin de spécifier les besoins auxquels doit répondre VisiXML pour satisfaire ses objectifs.

2.1. Trois optiques pour le développement d'interfaces

A ce jour, lorsque l'on projette de développer une interface (qu'elle soit orientée Web ou non) pour un projet quelconque, on se trouve face à 3 possibilités [Moli02] :

- Tout d'abord le design manuel, c'est-à-dire le développement de l'interface en lignes de code, en entrant directement dans le programme approprié les spécifications correspondant à l'interface souhaitée. On retrouve généralement ce type d'approche dans les langages orientés objet (OO), qui requièrent des raffinements additionnels pendant la phase de développement, raffinements ne pouvant être apportés que par un designer.

- Une seconde possibilité est le recours aux outils de design semi-automatique. Dans ce cas le développement de l'interface ne se fait pas totalement à la main ; le développeur est assisté de « wizards » qui le guident afin d'obtenir le design et l'implémentation. Les outils les plus courants dans cette approche sont les « Toolkits » et les « Interface Development Tools » [Nune04].

- La dernière possibilité consiste à faire appel à des outils de design automatique qui, comme leur nom l'indique, permettent la génération automatique du code de l'interface. Ce sont principalement ces outils qui seront analysés dans cet état de l'art. On peut identifier deux sous-catégories dans cette approche :

    - Les outils de « drag and drop » (glisser-déposer), i.e. des outils qui vous permettent de ne plus vous occuper directement du code et génèrent celui-ci plus ou moins automatiquement sur base d'éléments prévus à cet effet dans un menu que vous faites glisser sur votre espace de travail. Les outils analysés dans cette catégorie seront Visual Basic, Dreamweaver et CanonSketch.

    - Les outils d'esquisse (i.e. sketching tools). Ce sont des outils qui, sur base d'un simple dessin dans un espace prévu à cet effet, reconnaissent certaines formes et traduisent celles-ci dans un langage de spécifications prédéterminé. Les outils analysés dans cette catégorie seront JavaSketchIt [Caet02] et Silk [Land01].

Pedro J. Molina propose une grille d'analyse des critères principaux de ces trois approches de développement [Moli02] :

	Manual Design	Semiautomatic Design	Automatic Design
Design Effort	High	Medium	Low
Design Choices	Many	Many	Few
Error prone	Yes (high)	Yes (moderate)	No
Mappings form Conceptual Patterns	It depends on designer	It depends on designer, but assisted	Always
Prototyping speed	Slow	Medium	Quick
Customization	High	Medium	Low
Fiability	It depends on designer	Medium	High

Le but de ce chapitre est de fournir une grille d'analyse semblable pour les outils de design automatique mentionnés plus haut ; les critères jugés pertinents sont explicités ci-dessous.

2.2. Construction d'une grille d'analyse multicritères

Il convient de noter tout d'abord que le développement d'une interface et la manière dont il s'opère affecte trois intervenants distincts :

- Le concepteur, dont le rôle est de définir les fonctions que l'interface devra remplir et de spécifier un certain nombre de critères qu'elle devra satisfaire. Son rôle consiste donc à cerner les besoins que l'interface devra combler, à définir son utilité.
- Le développeur, qui se charge de générer une interface correspondant aux spécifications fournies.
- L'utilisateur final de l'interface développée

De ce fait les critères choisis pour analyser les outils distingueront dans la mesure du possible les impacts de ceux-ci du point de vue des trois intervenants.

2.2.1. Critères d'analyse affectant le concepteur

Coût de l'outil : l'outil de développement est-il gratuit à utiliser, faut-il payer une licence, une redevance annuelle ?

Coût moyen d'un projet : quel sera l'investissement à réaliser pour le concepteur et le développeur afin de réaliser un projet au moyen de l'outil, que ce soit un investissement financier ou un investissement de temps?

Durée moyenne d'un projet : on évaluera par ce critère l'investissement en temps nécessaire au développement d'une interface au moyen des différents outils, ce qui sera à mettre en relation avec la flexibilité de l'output.

Bootstrapping : l'interface de l'outil peut-elle être générée à partir de l'outil lui-même ? Il s'agit d'un critère complémentaire afin de juger de la flexibilité des outputs de l'outil.

Niveau de couverture : les interfaces générées par l'outil exploitent-elles au maximum les possibilités des langages dans lesquelles elles sont formulées ? (critère complémentaire à la représentativité)

Types d'interfaces supportés : quels types d'outputs l'outil permet-il ? applications graphiques, applications Web, applications multimédia, systèmes d'information, systèmes experts, autres, …

2.2.2. Critères d'analyse affectant le développeur

Aperçu en temps réel : l'outil de développement donne-t-il à chaque moment à l'utilisateur un aperçu de son travail ? Les différentes possibilités sont soit aucun aperçu, soit une représentation complète, soit la représentation d'une ou plusieurs étapes intermédiaires.

Niveau de représentation : il s'agit ici de déterminer le type d'output généré par l'outil ; est-ce un langage de description complet ou sont-ce seulement des spécifications de certains paramètres demandant un ajustement manuel ultérieur ?

Niveau d'intervention de l'outil : ce critère est complémentaire au niveau de représentation, il s'agit de déterminer si l'outil intervient au niveau du prototypage de basse, moyenne ou haute fidélité.

Facilité d'utilisation : on comparera ici le niveau de compétences requis pour utiliser l'outil de développement en lui-même, il ne s'agit en aucun cas de juger l'ergonomie de l'output.

Déboguage : lorsqu'un problème surgit dans la création, existe-t-il des possibilités pour l'utilisateur de les corriger, que ce soit par une interaction directe en ligne de commandes ou tout autre moyen ?

Contrôlabilité du processus de production de l'output : l'utilisateur de l'outil peut-il intervenir et si oui dans quelle mesure ? Je m'en référerai pour ce critère à la définition donnée par C. Mariage, J. Vanderdonckt, J. Eisenstein et A. Puerta : le processus de production de l'output sera dit contrôlable « […] si la traçabilité du processus peut être spécifiée ou contrôlée par l'utilisateur ». Encore faut-il pour cela que le processus soit traçable : « L'utilisation de directives (recommandations) observables est dit traçable si le processus peut être suivi étape par étape […] » [Mari04].

2.2.3. Critères d'analyse affectant l'utilisateur final

Flexibilité de l'output : il s'agit de déterminer dans quelle mesure l'output peut être transformé pendant le développement ou par la suite pour répondre à différents types d'utilisation. Un langage de spécifications admettra de ce fait plus de possibilités ultérieures de transformation de l'output moyennant un effort moindre qu'une description complète en un langage devant être retraduite.

Représentativité : ce critère permettra d'évaluer dans quelle mesure l'outil de développement permet de générer un output correspondant exactement aux attentes de l'utilisateur final.

2.3. Analyse des outils suivant la grille

2.3.1. Macromedia Dreamweaver

Dreamweaver est l'un des éditeurs d'interfaces les plus utilisés à l'heure actuelle ; il fonctionne en mode WYSIWYG (What You See Is What You Get) et permet de créer des interfaces Web sans avoir aucune notion de HTML, tout en visualisant l'aspect de l'interface en même temps qu'on l'édite. Dans sa première version il n'offrait comme possibilité que la génération d'interfaces HTML, mais ses fonctions ont été fortement élargies depuis, avec l'intégration partielle ou complète des langages comme CFML, ASP.NET, JSP, PHP, XML, CSS et XHTML (Fig.2.1).

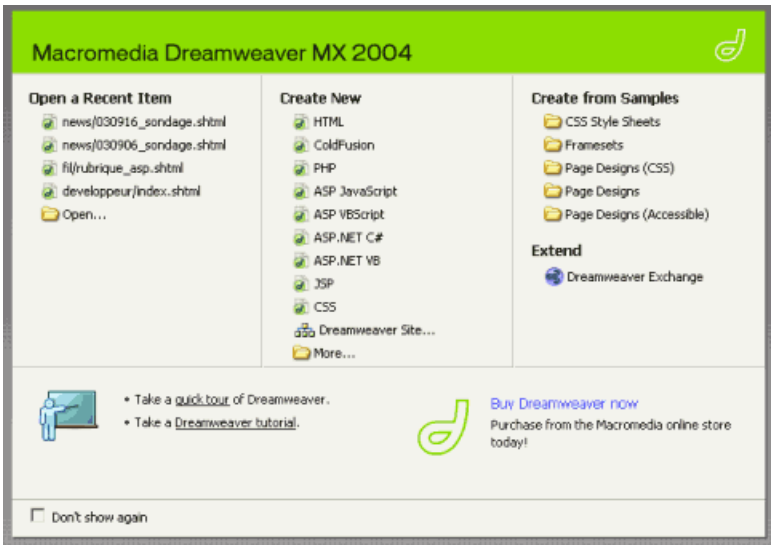


Fig.2.1 : Aperçu des langages supportés par Dreamweaver

Analysons maintenant cet outil suivant les différents critères proposés en amont :

- Pour le concepteur :
  - Coût de l'outil : Dreamweaver n'est pas un outil gratuit, son prix variant entre 479 € et 1129 € pour une version complète.
  - Coût moyen / Durée moyenne d'un projet : il s'agit d'un critère difficile à évaluer ; le coût et la durée dépendront évidemment de la complexité de l'interface que le concepteur désire générer. Cependant, le fait que

Dreamweaver soit un outil de design automatique en mode WYSIWYG permet de supposer que ces deux critères seront relativement faibles. En effet, un outil d'utilisation simple de par son mode de fonctionnement (cfr. infra) et rapide de par son caractère automatique ne devrait théoriquement pas engendrer de coûts moyens de développement exorbitants.

- Types d'interfaces supportés: c'est un outil conçu pour créer des applications graphiques, web et multimédia. Il n'intègre aucun support de base de données ou autres pouvant en faire un générateur de systèmes d'information ou de systèmes experts.
- Bootstrapping : il ne sera jamais possible de recréer grâce à Dreamweaver l'environnement de celui-ci, du fait de la présence au sein de son interface d'éléments graphiques non reproductibles dans ses langages d'output.
- Niveau de couverture : le niveau de couverture offert par Dremweaver est maximal ; en effet, la quasi-totalité des balises HTML existantes est représentée sur la palette d'outils, et si par hasard il en manquait une à l'utilisateur celui-ci pourrait toujours se servir du mode « code » pour combler le manque (cfr. infra).

- Pour le développeur :
  - Aperçu en temps réel : Dreamweaver permet à l'utilisateur de travailler soit en mode « code », soit en mode graphique, et ce dernier lui fournit à tout instant une représentation de l'interface qu'il est en train de coder.
  - Facilité d'utilisation : Dreamweaver se compose de deux palettes, une palette d'outils qui permet de sélectionner les éléments HTML à poser sur le projet en cours et une palette propriétés qui permet de définir les propriétés de l'objet choisi de manière extensive. Le niveau de compétences requis pour utiliser Dreamweaver est donc minimal.
  - Contrôlabilité du processus de production de l'output : dans le cas de Dreamweaver, le processus de production de l'output n'est pas à proprement parler « traçable », c'est-à-dire qu'il n'est nulle part possible de suivre pas à pas la mise à feu des différents événements et leurs

conséquences. Et comme le processus n'est pas « traçable », il ne pourra pas non plus être contrôlable au sens où cela a été défini précédemment mais, comme mentionné plus haut, Dreamweaver permet également de travailler en mode « code », c'est-à-dire que, si le produit de la génération automatique du code ne satisfait pas aux exigences de l'utilisateur, ce dernier peut effectuer ses corrections de manière manuelle, corrections qui seront par la suite automatiquement répercutées sur l'aperçu en temps réel du mode graphique. On n'a donc aucun contrôle sur la partie génération automatique, mais le résultat de cette dernière peut être « corrigé » directement dans l'outil via la fenêtre de code.

- Déboguage: en plus de l'interaction directe que l'utilisateur peut avoir sur le code, Dreamweaver comporte un historique des dernières modifications effectuées qui permet, si besoin est, d'en annuler quelques unes pour « revenir en arrière ».
- Niveau de représentation / Niveau d'intervention de l'outil : tous les outputs générés par Dreamweaver interviennent au niveau du prototypage haute fidélité, c'est-à-dire que les spécifications générées sont complètes (couleur, taille, police,…). Dreamweaver produit donc une interface concrète, ne laissant plus de place à l'abstraction.

- Pour l'utilisateur final :
  - Flexibilité de l'output : de par sa nature de spécification concrète l'output HTML généré par Dreamweaver n'est pas facilement transformable, en ce sens qu'une modification à apporter au code prendra beaucoup plus de temps que si le prototypage avait été de basse fidélité. En effet, il faudra manuellement rechercher tous les critères (couleur, taille, police, …) afin de les supprimer / modifier.
  - Représentativité : la représentativité des interfaces générées est quant à elle très forte. En effet, dans la mesure où l'utilisateur final désire interagir avec une interface Web, celles générées par Dreamweaver sont parmi les

plus complètes du fait de son niveau de couverture idéal du langage HTML (cfr. supra) et de son intégration des différents autres langages disponibles pour la création d'interfaces Web.

2.3.2. Microsoft Visual Basic

Visual Basic est un outil développé par Microsoft pour développer facilement des applications fonctionnant sous Microsoft Windows. C'est, comme Dreamweaver, un outil de « drag and drop » permettant de créer avec de faibles notions de programmation des interfaces graphiques, à la différence près que celles-ci ne sont plus orientées Web (ce qui vient de changer avec la version .NET) mais destinées à l'Operating System de Microsoft.

L'intérêt de cet outil est de pouvoir associer aux éléments de l'interface des portions de code associées à des événements (clic de souris, appui sur une touche, ...). Pour cela, Visual Basic utilise un petit langage de programmation dérivé du BASIC (signifiant *Beginners All-Purpose Symbolic Instruction Code*, soit *code d'instructions symboliques multi-usage pour les débutants*). Le langage de script utilisé par Visual Basic est nommé à juste titre **VBScript** ; il s'agit ainsi d'un sous-ensemble de Visual Basic[Ccm04].

Passons en revue les différents critères de la grille d'analyse pour cet outil :

- Pour le concepteur :
  - Prix de l'outil : le prix recommandé par Microsoft pour une version standard de Visual Basic est actuellement de 109$.
  - Coût moyen / Durée moyenne d'un projet : à nouveau, ce critère est difficile à juger mais on peut comme précédemment se baser sur le fait qu'il s'agit d'un outil de développement automatique qui ne requiert qu'un peu d'expérience pour être employé efficacement. Le coût moyen et la durée moyenne d'un projet seront donc tous deux relativement faibles, en

tout cas en comparaison de ce qu'aurait été le développement d'une application semblable en suivant une approche manuelle ou semi-automatique.

- Types d'interfaces supportés: comme expliqué dans la description préliminaire, Visual Basic permet de créer des applications complètes pour Microsoft Windows, et d'en gérer à la fois la partie statique (interface) et la partie dynamique (actions résultant d'événements, par exemple de clicks sur une partie de l'interface). Cela fait de Visual Basic un générateur non seulement d'applications graphiques, mais également de systèmes d'information et de systèmes experts (programme informatique simulant l'intelligence humaine dans un champ particulier de la connaissance ou relativement à une problématique déterminée [Uqam96]). De plus, la dernière version en date de l'outil (Microsoft Visual Basic .NET 2003) permet la création d'applications Web.
- Niveau de couverture : ce critère n'a presque pas lieu d'être dans ce cas-ci, puisque le langage Visual Basic ne peut être généré, à ma connaissance, que par cet outil. La couverture est donc totale, toutes les possibilités du langage sont exploitées par l'outil.
- Bootstrapping : le bootstrapping consiste, rappelons-le, à savoir si l'interface de l'outil peut être générée par l'outil lui-même. Dans le cas de Visual Basic l'interface se compose de menus, boutons et surfaces de dessins qui ont leur équivalent dans le programme lui-même ; de plus, les opérations résultant d'une action sur tel ou tel élément peuvent normalement être reproduites grâce au système d'événements de Visual Basic. En théorie il semble donc que Visual Basic soit effectivement capable de bootstrapping, mais en pratique la reconstruction de tout l'outil semble tellement pharaonique que je n'ai pas le courage de m'atteler à la vérification de mes dires.

- Pour le développeur :
  - Aperçu en temps réel : dans Visual Basic, l'output graphique est en permanence représenté, puisque le formulaire sur lequel on dépose les widgets (Fig.2.2) est quasiment en tout point similaire à l'output réel.

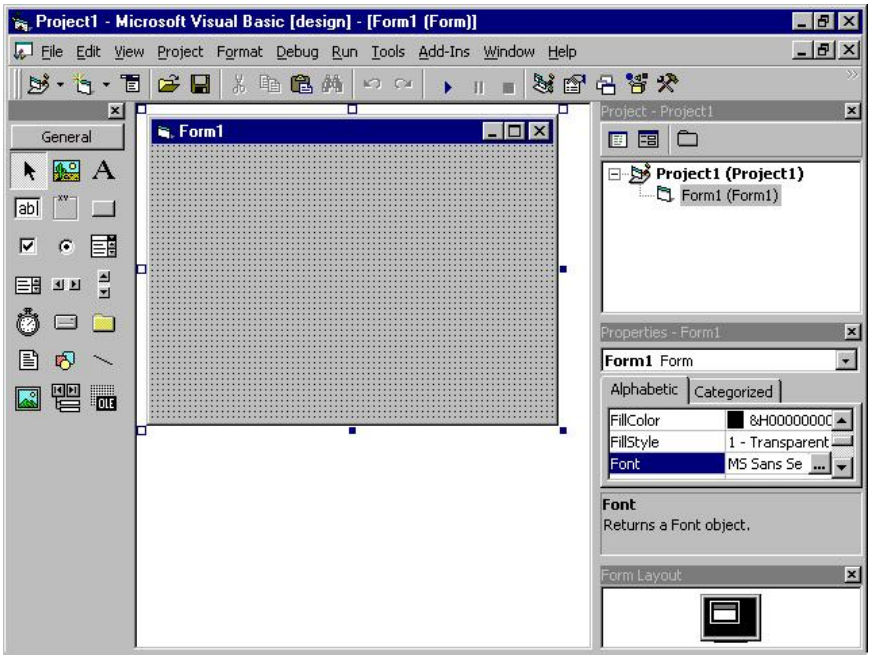


Fig.2.2 : Apercu de la surface de travail de Microsoft Visual Basic

En ce qui concerne la partie dynamique du code, il est à tout moment possible d'exécuter ce dernier afin d'en avoir un aperçu.

  - Facilité d'utilisation : rappelons le, Visual Basic est un dérivé de BASIC, qui signifie *Beginners All-Purpose Symbolic Instruction Code*, un nom qui n'est pas trompeur puisque tout est fait dans cet outil pour en faciliter l'utilisation par les débutants. Le programme différenciera par exemple grâce à la couleur tous les noms de procédures, fonctions et opérations reconnues. De plus, il y a en permanence au dessus de la fenêtre de code deux listes déroulantes ; une liste déroulante « Objets », qui contient tous les éléments appartenant à la feuille dont dépend la fenêtre de code, et une liste déroulante « Procédures », qui affiche tous les événements disponibles pour l'objet actif (Fig.2.3):

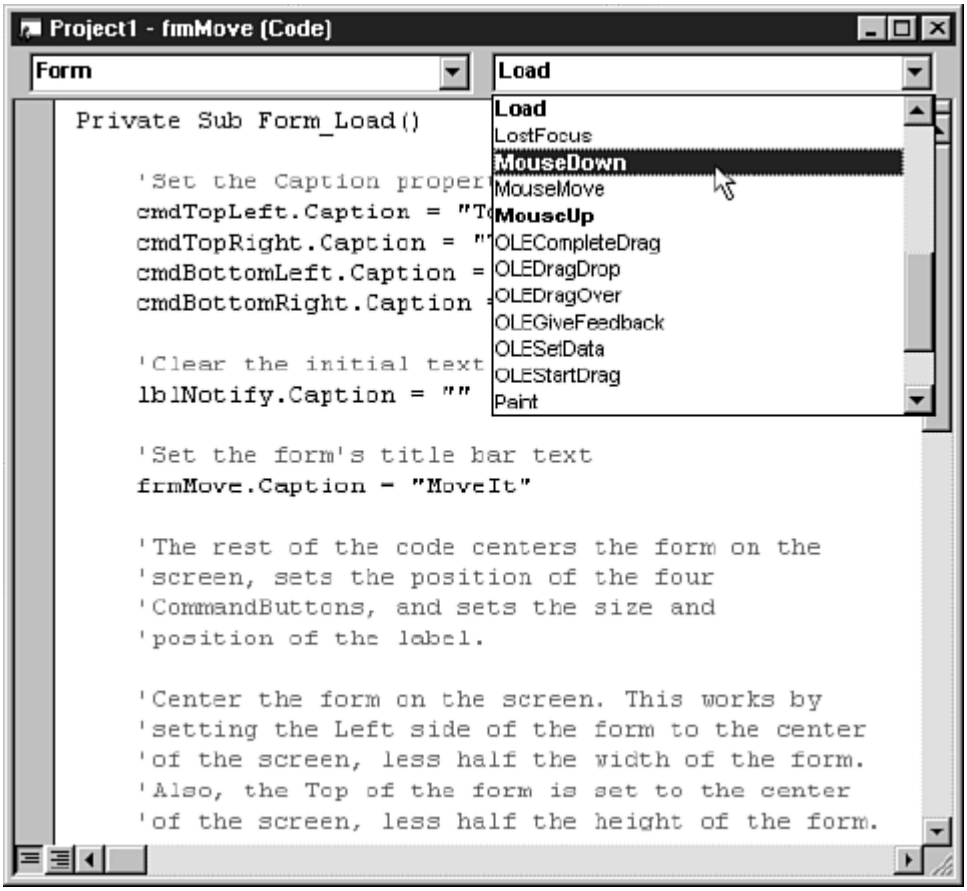


Fig.2.3 : Liste déroulante « Procédures » de Microsoft Visual Basic

- Contrôlabilité du processus de production de l'output : rappelons que la contrôlabilité nécessite des directives (recommandations) observables, qui peuvent être suivies étape par étape (traçables), et dont on peut spécifier la traçabilité. Il n'y a cependant aucun suivi des directives dans Visual Basic et l'utilisateur n'a aucun moyen d'influencer l'ordre dans lequel les opérations sont effectuées au moment où il dépose un widget sur un formulaire. Visual Basic peut donc à mon sens être qualifié de non contrôlable.
- Niveau de représentation / Niveau d'intervention de l'outil : Visual Basic est un langage de description complète ; tous les aspects des éléments graphiques qu'il va générer sont possibles à définir dans le programme. C'est donc un outil qui intervient au niveau du prototypage de haute fidélité, et qui convient mieux pour une utilisation en phase finale de

développement ; on n'y retrouve que des objets interactifs concrets et aucune structure qui pourrait permettre de représenter un modèle de manière plus abstraite.

- Déboguage : Visual Basic est un des outils les mieux équipés pour le déboguage, il permet en effet de faire appel à une multitude de fonctions telles que le mode arrêt, l'insertion de variables espions, l'affichage de la valeur des variables lors de l'exécution avec la commande debug.print, etc.

- Pour l'utilisateur final :
  - Flexibilité de l'output : de par sa nature de langage de description complète, Visual Basic requerra un effort assez conséquent si l'on veut transformer l'output généré pour le destiner à une autre utilisation.
  - Représentativité : dans quelle mesure Visual Basic permet-il de générer une interface correspondant exactement aux attentes de l'utilisateur final ? Il s'agit une fois de plus d'un critère assez difficile à évaluer tant que l'on n'a pas soi-même été confronté aux limites du programme. Cependant, les fonctionnalités offertes par Visual Basic reprennent tous les widgets fréquemment utilisés dans les interfaces et la liste déroulante des procédures est souvent tellement impressionnante qu'on se demande comment ne pas y trouver son bonheur. Je pense donc pouvoir dire que Visual Basic offre aux utilisateurs une bonne représentativité.

2.3.3. CanonSketch

CanonSketch est un outil particulier dans son genre en ce sens qu'il donne une représentation de l'interface à trois niveaux différents pendant la fabrication de celle-ci. Il y a tout d'abord la « Wisdom view » qui montre le diagramme de classes UML des différents objets interactifs abstraits ; il y a également la « Canonical view » (Fig.2.4) qui montre ces mêmes objets tels qu'ils ont été déposés sur la feuille depuis la palette, et enfin la vue HTML qui permet, après génération via un bouton spécial, d'observer

l'output HTML tel qu'il serait si on arrêtait le travail à ce moment-là. CanonSketch est donc un outil de « drag and drop » classique, mais qui permet la représentation de l'interface générée à trois niveaux d'abstraction différents.

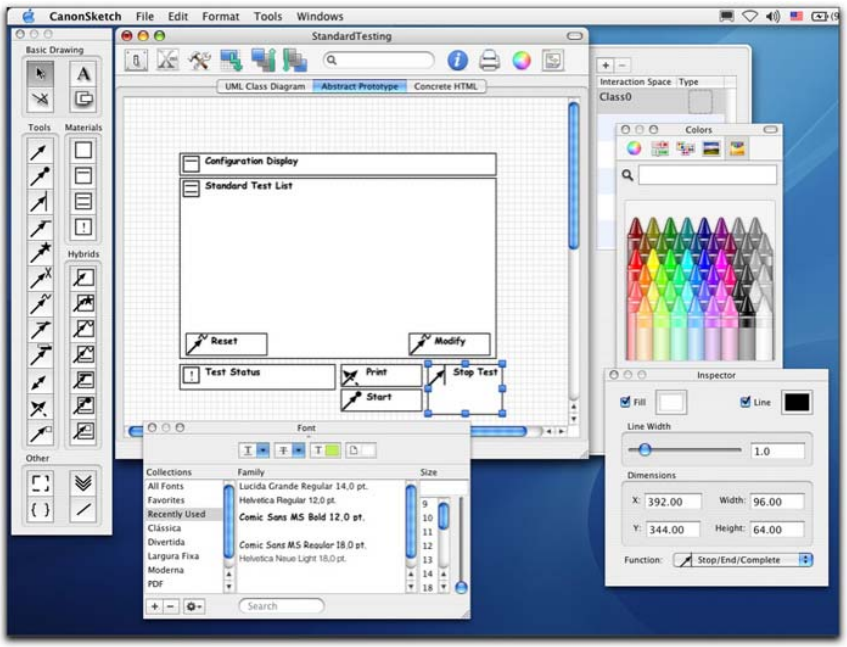


Fig.2.4 : Canonical View de l'outil Canonsketch

Analysons maintenant cet outil suivant les différents critères :

- Pour le concepteur :
  - Prix de l'outil : à l'heure actuelle, CanonSketch est gratuit à utiliser.
  - Coût moyen / Durée moyenne d'un projet : à nouveau, ce critère est difficile à évaluer sans une batterie de tests permettant de comparer l'outil à tous les autres outils du genre, mais une fois de plus, il s'agit d'un outil de « drag and drop » et l'on peut donc raisonnablement affirmer que la durée moyenne et le coût moyen de développement sont réduits.
  - Types d'interfaces supportés : CanonSketch permet l'exportation de l'output en HTML mais permettra également bientôt l'exportation en XMI, ce qui accroîtra les possibilités d'échange avec les autres outils « UML-based » [Cam04]. CanonSketch permet donc pour le moment la génération d'interfaces orientées Web complètes mais sans prise en charge

de base de données, ce qui l'empêche de générer complètement des systèmes plus complexes, tels les systèmes experts. Ci-dessous, une illustration des deux types d'output actuellement supportés (Fig.2.5).

- Niveau de couverture : il convient ici de distinguer au moins deux niveaux de couverture. La couverture HTML actuelle, bien que fort étendue, n'est pas encore complète, mais il ne me semble pas que ce soit là l'objet de cet outil. En effet, selon les développeurs de CanonSketch, la partie HTML n'est qu'une « représentation d'une implémentation concrète possible » [Cam04] et l'accent est mis sur la partie « prototypage abstrait » du développement, pour laquelle le niveau de couverture est déjà assez conséquent puisqu'ils reprend des objets tels que les « wizards » ou les « fenêtres d'exploration » semblables à celles de windows explorer. Il est d'ailleurs précisé qu'un des développements futurs sera l'identification des notations préférées des développeurs aux trois niveaux de prototypage (basse, moyenne et haute fidélité) et l'intégration de celles-ci au sein de CanonSketch ; on peut donc raisonnablement penser que le niveau de couverture ira en s'améliorant.

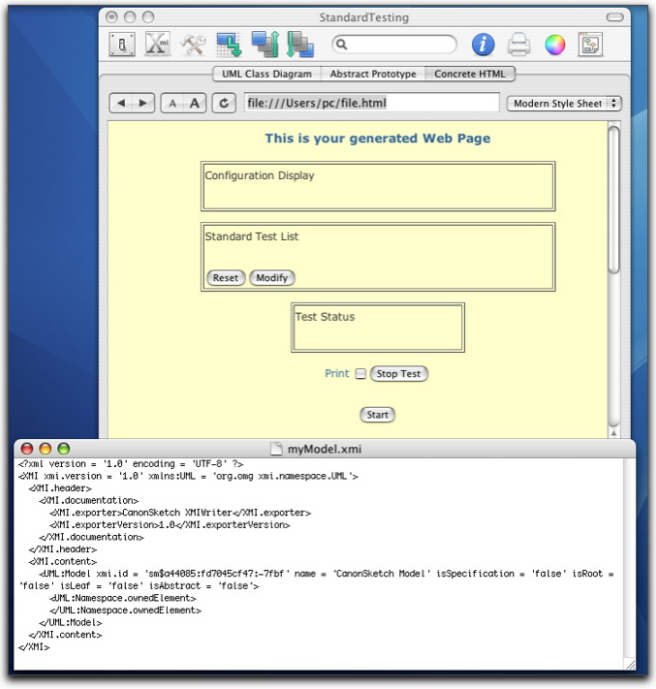


Fig.2.5 : XMI et HTML sous Canonsketch

- Bootstrapping : l'interface de CanonSketch n'est pas bootstrappable, du moins pas en termes d'interface concrète, elle reprend beaucoup d'éléments particuliers qui ne peuvent pas être représentés via ses outputs HTML et XMI, mais qui pourraient cependant être identifiés dans le diagramme de classes UML. CanonSketch est donc à mon sens capable de produire un schéma structurel abstrait de sa propre interface, mais pas de reproduire cette dernière telle quelle.

- Pour le développeur :
  - Aperçu en temps réel : comme déjà dit plus haut, CanonSketch propose à tout moment une représentation abstraite de la structure de l'interface ainsi qu'un aperçu d'une des implémentations concrètes possibles de cette structure (i.e. la version HTML de l'interface).
  - Facilité d'utilisation : la facilité d'utilisation est sans doute le point faible de CanonSketch, en effet la superposition permanente de ses trois couches de représentation peut induire assez rapidement l'utilisateur en erreur. C'est un outil qui semble réservé aux utilisateurs avertis plutôt qu'aux débutants, qui auront du mal à trouver leurs marques.
  - Contrôlabilité du processus de production de l'output : même remarque que pour tous les outils précédents : le processus de production de l'output n'étant pas directement traçable il est impossible d'en influencer l'ordre de quelque manière que ce soit.
  - Niveau de représentation / Niveau d'intervention de l'outil : CanonSketch intervient à tous les niveaux de prototypage (basse, moyenne et haute fidélité) grâce à ses 3 vues différentes. Il est possible d'imprimer directement sous format pdf le diagramme de classes UML ainsi que la représentation canonique de l'output. Pour ce qui est du prototypage haute fidélité, il peut directement être généré en HTML mais, comme précisé plus haut, il ne s'agit que d'un des outputs possible, la génération en XMI permettant des transformations ultérieures diverses de l'interface.

- Déboguage : je ne suis pas en mesure de tester moi-même cet outil, car n'étant pas en possession d'un mac sous OS X 10.3, et il n'est fait aucune référence aux possibilités de déboguage dans le pdf mis à disposition sur le site des développeurs.

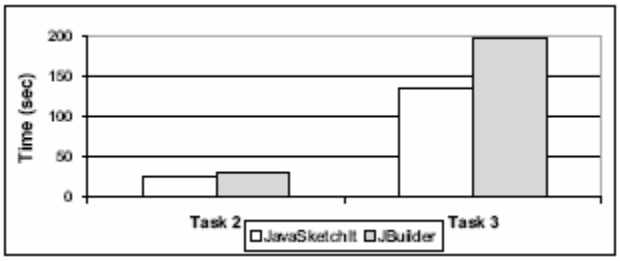
- Pour l'utilisateur final :
  - Flexibilité de l'output : bien que l'output HTML ne soit pas directement et facilement modifiable si l'on veut le destiner à une autre utilisation, son homologue en XMI permet l'export de l'interface vers d'autres outils « UML-based » qui permettront des modifications. De plus, il est toujours possible de se servir du prototypage basse fidélité (i.e. le diagramme UML) afin de recadrer l'interface dans sa nouvelle attribution, ce qui évite de devoir repartir à zéro et représente donc un gain de temps certain.
  - Représentativité : il s'agit là d'un point fort de CanonSketch par rapport aux autres outils de « drag and drop », puisque dans ce cas-ci ce ne sont pas les widgets « concrets » que l'on dépose sur le plan de travail mais bien des objets « canoniques », qui ne reprennent que les propriétés essentielles de l'élément, celles-ci étant repositionnables par la suite et permettant l'ajout d'attributs supplémentaires via un menu déroulant.

2.3.4. JavaSketchIt

JavaSketchIt est un outil qui permet la création d'interfaces via la reconnaissance de formes géométriques dessinées à la main et du mouvement qui a permis de les dessiner. Cet outil génère une interface Java dont le layout est *a posteriori* rendu plus beau grâce à l'application de règles grammaticales (e.g. alignement de groupes d'objets). Les créateurs de cet outil ont, pour valider leur approche, conduit des études d'utilisabilité mesurant leur système avec l'outil commercial servant à générer les interfaces en Java, JBuilder. Il en est ressorti des avantages au niveau de la vitesse de création et de l'utilisation intuitive de l'outil [Caet02].

Passons en revue les différents critères de la grille d'analyse pour cet outil :

- Pour le concepteur :
  - Prix de l'outil : JavaSketchIt en est pour le moment au stade de prototype et est donc gratuit à l'utilisation
  - Coût moyen / Durée moyenne d'un projet : les développeurs de l'outil ont demandé à 6 utilisateurs d'effectuer deux tâches (une tâche facile et une tâche complète) à la fois sous JBuilder et sous JavaSketchIt, et ce après leur avoir donné les instructions basiques relatives à l'emploi de ces deux outils. Il en ressort un net avantage pour JavaSketchIt en termes de temps de création d'interfaces complexes :



    Il faut cependant noter que l'utilisation de JavaSketchIt requiert un matériel plus complexe et donc également plus coûteux (i.e. un Tablet pc, leurs tests ont été effectués sur des modèles Sony Vaio LX900).
  - Types d'interfaces supportés : étant donné que JavaSketchIt en est encore au stade de prototype, le nombre d'éléments graphiques qu'il couvre est encore limité, mais en voie d'expansion. Il est pour le moment uniquement possible de générer des interfaces statiques simples, à l'aide des widgets standards (button, text, radiobutton, checkbox, combobox, field text, text area, image, menu et scrollbar). Aucune précision n'est donnée quand à la possible inclusion future des éléments dynamiques des interfaces.

- Niveau de couverture : comme précisé ci-dessus, le niveau de couverture des éléments graphiques de JavaSketchIt est pour le moment limité mais va grandissant [Caet02].
- Bootstrapping : l'interface de JavaSketchIt est pour l'instant relativement simple et je n'ai pu y identifier aucun élément qui ne soit pour le moment pas représentable par l'application elle-même.

- Pour le développeur :
  - Aperçu en temps réel : la surface de dessin de JavaSketchIt affiche directement après la reconnaissance de la forme le label correspondant à celle-ci (Fig.2.6). Cela permet d'avoir une vue assez représentative de l'interface que l'on est en train de créer. Cependant, on peut regretter le manque de liens entre les différentes parties d'interface qu'on crée et, de ce fait, le manque de vision de l'interface dans sa globalité.

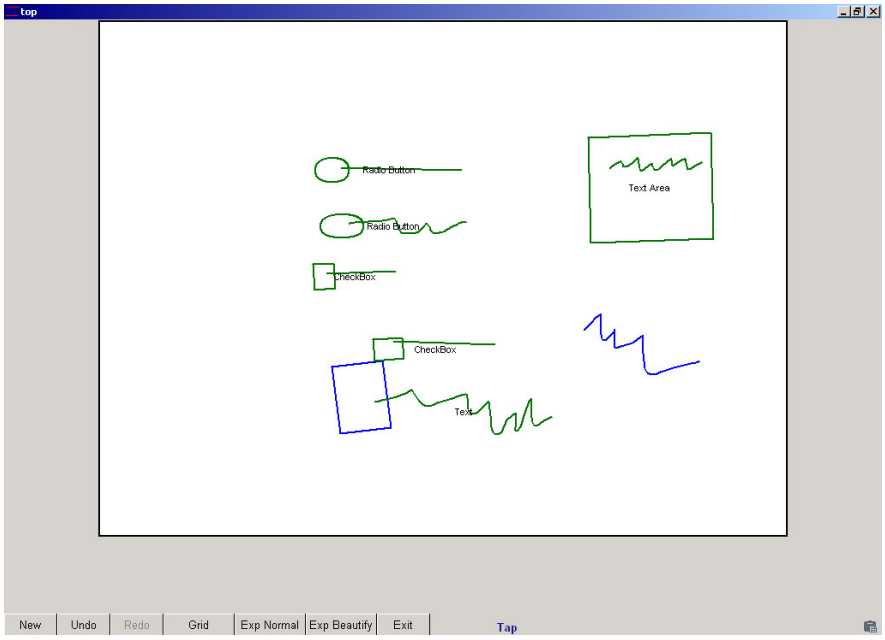


Fig.2.6 : Affichage sous forme de texte des éléments reconnus par JavaSketchIt

  - Facilité d'utilisation : au premier abord JavaSketchIt semble un peu austère et l'on a du mal à y trouver ses marques ; cependant, pour les utilisateurs avertis, le sketching est tout ce qu'il y a de plus facile. On peut donc dire que JavaSketchIt est un outil relativement facile à utiliser pour

qui sait dessiner plus ou moins correctement à l'aide de la souris ou du tablet pc (autrement la non-reconnaissance des formes rend la tâche plus compliquée).

- Contrôlabilité du processus de production de l'output : comme pour Dreamweaver et Visual Basic, le processus de production de l'output n'est pas directement « traçable », c'est-à-dire qu'il n'y a aucun moyen de suivre les mises à feu successives des différents événements qui vont composer la génération de l'output. De ce fait il n'y a donc également aucun moyen d'en influencer l'ordre et l'on peut dire que la production de l'output sous JavaSketchIt n'est pas contrôlable.
- Niveau de représentation / Niveau d'intervention de l'outil : JavasketchIt intervient au niveau du prototypage de moyenne fidélité, ce sont bien des objets interactifs concrets qui sont générés, mais tous leurs attributs ne sont pas définis, loin de là. C'est donc un outil qui doit servir à faire une première structure de l'interface afin notamment de pouvoir en discuter avec un client pour qu'il puisse y apporter des modifications.
- Déboguage : JavaSketchIt ne présente pas de possibilité de déboguage directe, i.e. si à la suite du dessin d'une checkbox à l'écran l'outil la reconnaît comme étant un radiobutton, il n'y a pas d'autre moyen que d'effacer le dessin et le recommencer pour que la reconnaissance se fasse de la manière souhaitée.

- Pour l'utilisateur final :
  - Flexibilité de l'output : les remarques faites pour les deux outils précédents sont d'application pour JavaSketchIt, à savoir que ce dernier produit un output en java, donc des spécifications concrètes qu'il faudra modifier à même le code si l'on veut changer tel ou tel paramètre pour destiner l'output à une autre utilisation. La flexibilité de l'output est donc faible.
  - Représentativité : en raison de sa faible couverture actuelle, JavaSketchIt ne permet pas encore à l'utilisateur de représenter tout type d'interface de

la manière dont il le souhaite, mais la représentativité de cet outil devrait nettement s'améliorer lorsque le nombre de widgets supportés augmentera.

Le projet SketchiXML

Maintenant que nous venons d'analyser JavaSketchIt et d'en identifier les points forts et les faiblesses, il semble intéressant de faire une petite parenthèse sur le projet de développement d'une application du même genre, mais basée sur des spécifications UsiXML. Il s'agit de SketchiXML, développé actuellement par Adrien Coyette au sein de l'unité ISYS de l'UCL. Le principe est en tous points semblable à JavaSketchIt si ce n'est qu'au lieu de générer un output en Java, SketchiXML générera des spécifications UsiXML avec tous les avantages que cela peut comporter, comme nous le détaillerons plus loin dans ce mémoire. Sans entrer dans les détails cependant, nous pouvons d'ores et déjà dire que SketchiXML permettra, sur base d'un dessin sommaire des différents éléments d'une interface graphique, de générer cette dernière dans tous les langages d'outputs supportés par UsiXML (cfr. infra), ce qui engendrera une flexibilité nettement améliorée des outputs en comparaison à une application telle que JavaSketchIt [Coye04].

2.3.5. Silk

Silk est un outil de sketching dont le but est de fournir un support aux développeurs lors de la phase de design créatif préparatoire au développement d'une interface, mais également lors de la communication des idées de design aux autres développeurs. C'est donc un outil de prototypage de basse fidélité (design conceptuel) qui servira essentiellement dans les premiers instants de la création d'interfaces. Analysons-le maintenant sur base de la grille de critères :

- Pour le concepteur :
  - Prix de l'outil : Silk est téléchargeable gratuitement sur le site de l'université de Berkeley.

- Coût moyen / Durée moyenne d'un projet : à nouveau, il s'agit d'un outil de sketching et la durée de développement d'un projet sera donc sensiblement réduite par rapport aux méthodes classiques de développement, tout en restant proche de la durée de développement moyenne d'un projet avec un outil de « drag and drop ».
- Types d'interfaces supportés : Silk étant un outil focalisé sur le design conceptuel, il convient dans le développement de tous types d'interfaces. De plus il prend en compte les « storyboards », dans lesquels l'arrangement des formes montre comment les éléments du design vont se comporter, comme par exemple le fait qu'une dialogbox va apparaître lorsque l'utilisateur appuiera sur un bouton. Ceci lui permet de fournir une représentation de systèmes complexes tels les systèmes d'information ou les systèmes experts.
- Niveau de couverture : Silk prend en compte 12 types de « controls » qui composent la majorité des interfaces actuelles mais, une fois de plus, le niveau de couverture de Silk n'est pas représentatif, car il s'agit plus de générer des spécifications plus ou moins abstraites pour aider les développeurs dans les premières phases du design que d'interfaces concrètes prêtes à l'emploi. On peut voir ci-dessous l'écran « controls » de Silk (Fig.2.7):

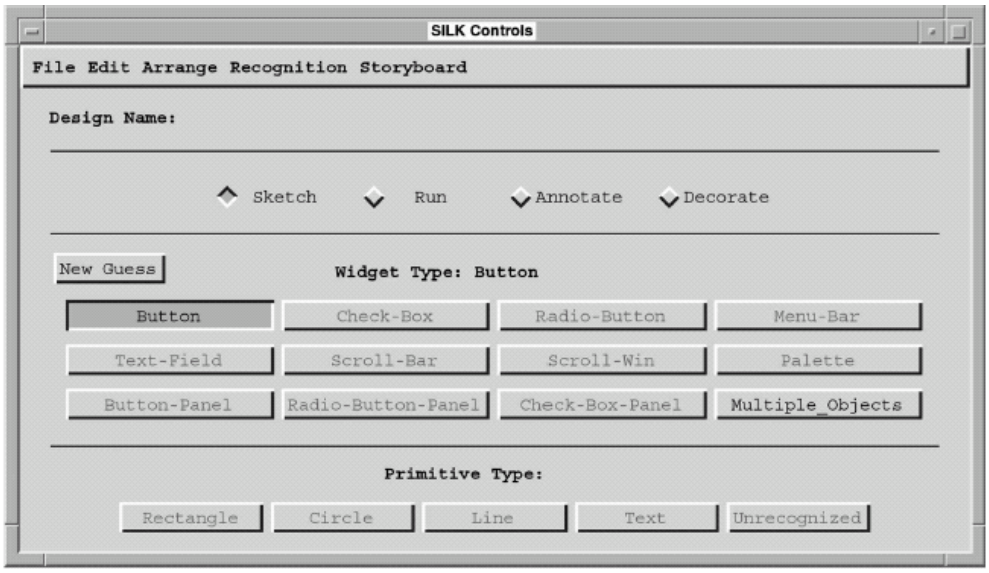


Fig.2.7 : Aperçu de l'écran « controls » de SILK

- Bootstrapping : Silk est bien entendu bootstrappable mais ce critère n'a pas vraiment lieu de s'appliquer ici, puisqu'il sert principalement à juger de la flexibilité de l'output des outils de génération d'interfaces. Silk ne générant pas vraiment d'interfaces, mais plutôt des spécifications abstraites qui sont par définition l'output le plus flexible qui soit, ce critère n'apportera donc aucune information supplémentaire dans l'analyse de Silk.

- Pour le développeur :
  - Aperçu en temps réel : comme expliqué par les auteurs de l'outil : « les développeurs peuvent tester l'interface à n'importe quel moment pendant sa création, pas seulement quand elle est finie. Quand ils sont satisfaits de leurs premiers prototypes, Silk leur permet de transformer leur dessin en une interface utilisant de vrais widgets. » [Land01]
  - Facilité d'utilisation : Silk est un outil simple, destiné à être employé par le plus grand nombre, il ne nécessite aucune connaissance particulière, à part peut-être la faculté de dessiner correctement.
  - Contrôlabilité du processus de production de l'output : le processus de production de l'output n'est pas contrôlable.
  - Niveau de représentation / Niveau d'intervention de l'outil : Silk est un outil spécialisé dans le prototypage de basse fidélité et sa fonction de génération d'interface ne permet pas de paramétrer les objets interactifs outre mesure.
  - Déboguage : il n'y a pas de fonction spéciale pour le déboguage, le seul moyen reste une fois de plus d'effacer les éléments s'ils ne correspondent pas aux attentes de l'utilisateur ; dans Silk cela se fait en dessinant une croix par-dessus l'élément en question.

- Pour l'utilisateur final :
    - Flexibilité de l'output : on pourrait dire de l'output généré par Silk qu'il n'est pas flexible, mais je ne pense pas que ce soit là un critère important pour juger l'outil, puisque son objectif est plus de fournir un moyen de communication entre un développeur et son client au moyen de dessins et d'interfaces très simples permettant de se représenter ce à quoi devrait ressembler l'interface une fois finie.
    - Représentativité : les interfaces générées par Silk correspondent assez bien aux attentes de l'utilisateur, puisqu'elles reprennent les objets interactifs standards avec un niveau de paramétrage faible.

2.3.6. Les environnements de développement d'interfaces orientés modèle

A présent que les différents outils de génération automatique ont été analysés suivant les critères choisis, il semble intéressant de fournir dans cet état de l'art une analyse de deux langages (principes) de spécifications basés sur le concept de modèle, car d'une part, ils ont pour vocation de servir de base à des systèmes de génération automatique de spécifications d'interfaces multi-plateformes et multi-langages (comme VisiXML) et, d'autre part, ils se basent également sur les stencils et la méthode de « drag and drop » de Visio pour ce faire (comme VisiXML également). Ces deux environnements ne seront pas repris dans la grille d'évaluation, car soit ils ne présentent pas encore d'outil de génération spécifique (dans le cas de DiaMODL), soit ces outils de génération qui leur sont spécifiques existent, mais je n'y ai pas eu accès (dans le cas de JUST-UI, des outils existent déjà pour la génération des interfaces en Visual Basic, Java, JSP, ASP et ColdFusion).

2.3.6.1. JUST-UI

JUST-UI est un modèle de spécifications d'interfaces basé sur le principe des « patterns » [Moli04a]. Le but de ce modèle est quasiment identique à celui de VisiXML, c'est-à-dire de capturer et de spécifier de manière abstraite toutes les « conditions » que l'interface devra remplir, pour pouvoir ensuite générer à partir de là l'interface utilisateur dans différents environnements et sur différentes plateformes. Une fois de plus, c'est ici le niveau d'abstraction élevé des spécifications qui facilitera la flexibilité des outputs (cfr. infra pour plus de détails à ce sujet concernant UsiXML).

Le modèle se base sur des patterns simples, qui sont faciles à obtenir, et qui sont ensuite assemblés pour former des patterns complexes. Un pattern simple est défini comme étant un composant qui peut-être directement renseigné par l'utilisateur ou le client au développeur [Moli04a] et tous les types d'interfaces peuvent ensuite être représentés par un assemblage défini de ces patterns. Il existe 5 types de patterns simples :

- le filtre : c'est le premier des patterns et il permet de définir ce que l'utilisateur cherche. Par exemple, dans le cas d'un service de location de voitures, l'utilisateur peut chercher des voitures d'une certaine couleur, bleu par exemple, et le filtre sera alors **color = « blue »**.
- le critère d'ordre : c'est le pattern qui permet de définir comment les éléments doivent être classifiés. Il faut définir sur quel attribut on va se baser pour ordonner les éléments et également s'il faut les classer par ordre croissant ou décroissant. Toujours dans l'exemple du service de location de voitures, le critère d'ordre pourrait être **brand ASC** (ascending), si l'on veut que les voitures soient classées par ordre alphabétique croissant de marques.
- le set d'affichage : maintenant que les objets sont filtrés et ordonnés, il s'agit de déterminer ce qui doit être affiché pour l'utilisateur. Un exemple de set d'affichage pour notre cas de service de location de voitures pourrait être **brand, model, year, colour, state, location.**

- la navigation : les objets étant à présent filtrés, ordonnés et listés, l'utilisateur a donc tout ce qu'il faut pour sélectionner celui qu'il veut. Une fois son objet sélectionné l'utilisateur a deux possibilités ; la première de ces possibilités est la navigation, c'est-à-dire l'affiche des informations relatives à son objet.
- l'action : la seconde possibilité est l'action, i.e. l'exécution d'une méthode ou d'un service qui aura pour effet de changer un ou plusieurs attributs de l'objet.

Sur base de ces 5 patterns simples, les concepteurs de JUST-UI ont donc défini plusieurs patterns complexes, dont voici deux exemples :

- Instance Presentation Pattern : son but est de modéliser la présentation des données d'un des éléments et de supporter l'interaction qui doit se faire entre l'utilisateur et cet élément. Il se compose de 3 patterns simples, à savoir un set d'affichage (permet de définir ce qu'il faut montrer à l'utilisateur), un pattern « action » (permet de définir ce que l'utilisateur peut modifier dans ce qui est affiché) et un pattern « navigation » (permet de déterminer les relations de cet élément avec d'autres qui pourraient être intéressantes à voir pour l'utilisateur).
- Population Presentation Pattern : ce pattern est l'équivalent du précédent, à ce point près qu'il permet d'abord à l'utilisateur de sélectionner son élément. En effet, si dans le premier pattern l'utilisateur n'avait accès qu'à un seul élément prédéterminé, ici il choisit d'abord son élément dans une liste avant de décider les propriétés de celui-ci qu'il veut afficher (set d'affichage), celles qu'il veut modifier (pattern « action ») ou encore les relations que cet élément a avec d'autres éléments dont il voudrait être informé (pattern « navigation »). Le fait de pouvoir d'abord choisir son élément dans une liste implique l'incorporation à ce pattern complexe des deux premiers patterns simples, le filtre et le critère d'ordre. Ceci est donc un pattern complexe composé de tous les types de patterns simples.

Une fois que les patterns complexes nécessaires ont été définis, il reste au développeur à spécifier les points d'entrée de l'utilisateur dans le système et à dessiner le diagramme de navigabilité correspondant aux spécifications de l'interface. Un diagramme de navigabilité est simplement un diagramme directionnel où les nœuds représentent des patterns complexes et sont reliés entre eux par des flèches représentant les liens de navigation existant entre ceux-ci. Un exemple de diagramme de navigabilité est donné ci-dessous [Moli04a]:

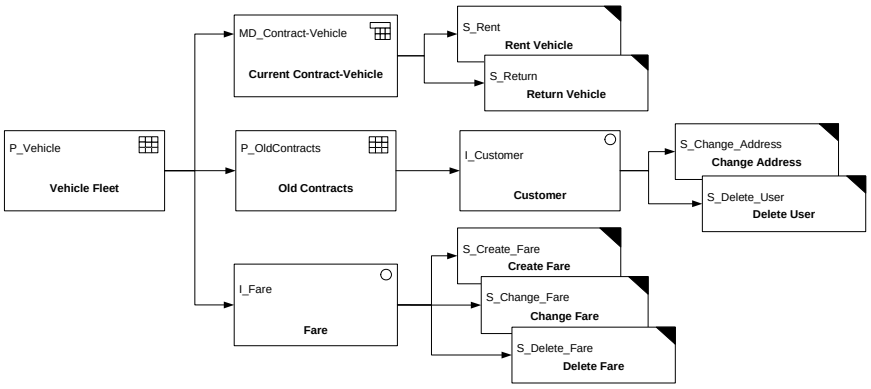


Fig.2.8 : Diagramme de navigabilité de JUST-UI

Ce sont ces types de diagrammes de navigabilité qui peuvent être aisément reproduits dans Visio et par la suite traduits en spécifications d'interfaces finales dans les différents langages précités.

2.3.6.2.DiaMODL

DiaMODL est un langage développé par Hallvard Traetteberg qui a pour but de permettre la modélisation des dialogues entre un système et ses utilisateurs [Trae02]. Il s'agit à nouveau d'un langage abstrait qui part du principe que toute interface, qui a pour rôle de prendre en charge ce type de dialogues, peut-être représentée par un ensemble d'interacteurs, ces derniers étant caractérisés par trois rôles fondamentaux:

- la médiation de l'information : l'interface utilisateur peut être vue à la fois comme un médiateur d'information entre un système et son utilisateur et

comme un outil pour effectuer des actions. Chaque interacteur définit l'information qu'il peut transmettre, ainsi que la direction dans laquelle il la transmet. Comme illustré dans la figure suivante [Trae02] :

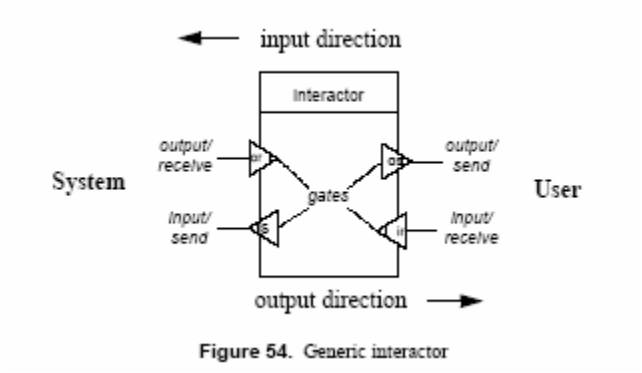


Figure 54. Generic interactor

- le contrôle et l'activation : les interacteurs ont un espace interne d'état qui contrôle l'envoi et la réception d'informations, l'activation d'autres interacteurs et le déclenchement de l'exécution de certaines fonctions spécifiques de l'application.
- la structure compositionnelle : les interacteurs peuvent être assemblés en de nouveaux interacteurs, ce qui permet d'offrir au langage un plus haut niveau d'abstraction en éliminant les détails.

Les différents éléments d'une interface concrète peuvent donc être représentés de manière abstraite sous formes d'interacteurs dans DiaMODL ; c'est le cas notamment d'une checkBox, qui sera représentée de la manière suivante (Fig.2.9) [Trae02] :

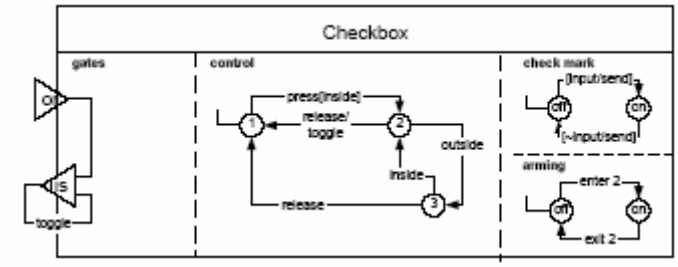


Fig.2.9 : Représentation d'une checkBox sous DiaMODL

L'important n'ést pas ici de détailler la représentation en DiaMODL de tous les objets d'interaction qu'une interface peut comporter, mais bien de savoir qu'une représentation abstraite des interfaces, basée sur la notion d'interacteur, existe et est en cours d'implémentation pour la plateforme Java.

Après cette petite parenthèse sur les environnements de développement orientés modèle, alors que tous les outils ont été analysés dans le détail suivant les critères de la grille, un tableau synthétique récapitulant ceux-ci devrait permettre de dégager certains objectifs importants pour le développement de VisiXML. Ce tableau est fourni en Annexe I pour des raisons de mise en page.

2.3.7. Discussion de la grille d'analyse multicritères

L'analyse de la grille nous permet de nous faire une idée sur la ligne directrice suivie en général par les outils de génération automatique. On peut y lire que tous ces outils présentent des coûts de développement ainsi qu'une durée moyenne de développement de projet qui sont faibles. Ceci est à vrai dire une caractéristique intrinsèque de cette catégorie d'outils et il va donc de soi que VisiXML aura pour but d'égaliser ce qui s'impose comme un standard pour ce type d'applications.

Au niveau des critères pouvant influencer le choix du concepteur, on notera également la faible couverture des outils gratuits ; nous essayerons de pallier cet inconvénient par le recours à UsiXML qui, grâce à ses spécifications abstraites, devrait permettre de générer au final des interfaces assez complètes dans les langages d'output (en effet, l'abstraction permet d'oublier dans un premier temps toute contrainte de traitement de tel ou tel type d'objet induite par telle ou telle petite caractéristique de celui-ci, pour ensuite réincorporer ces dernières une fois que les spécifications et donc le squelette de l'interface ont été générés, cfr. infra).

Le bootstrapping qui, rappelons-le, est un critère supplémentaire pour juger de la flexibilité des outputs d'un outil de génération, sera théoriquement possible dans le cas de VisiXML, l'élément « imageComponent » permettant, en théorie toujours, de représenter tous les objets visibles de l'interface de VisiXML qui ne sauraient être représentés autrement.

En ce qui concerne les types d'interfaces supportés, on peut constater que tous les outils analysés dans ce chapitre permettent la génération d'interfaces web et là, à nouveau, le fait qu'UsiXML soit un langage de spécifications de moyenne fidélité (du moins en ce qui concerne sa partie CUI, cfr. infra) va permettre à VisiXML de générer des spécifications d'interface pour tous types de supports. La nuance à apporter ici est que VisiXML ne fera « que » produire les spécifications UsiXML et ce sont ces spécifications qui pourront ensuite être traduites dans les différents langages supportés. Avec l'adjonction des différents traducteurs (interpreters) VisiXML permettra donc la création de tous types d'interface.

Au niveau des critères pouvant influencer les développeurs, on remarque que tous les outils étudiés permettent à l'utilisateur d'avoir un aperçu en temps réel de son travail. Ce ne sera malheureusement pas le cas avec VisiXML, pour la raison suivante : comme il sera détaillé plus loin dans ce mémoire, la génération des spécifications se passe en deux phases, la création d'une structure hiérarchique des shapes de la surface de travail dans un premier temps et le parcours de cette structure pour en extraire les attributs à intégrer aux spécifications dans un second temps. Il se fait que la manière la plus cohérente de représenter une structure hiérarchique de shapes est un arbre et que le seul composant de ce type disponible lors du développement de VisiXML et qui correspond aux besoins de celui-ci était le Microsoft Treeview Control version 6.0 ; or celui-ci présente la fâcheuse caractéristique de n'être accessible que pendant le runtime, du moins à ma connaissance, ce qui rend beaucoup plus ardue la génération d'un aperçu en temps réel.

Les autres critères au niveau du développeur sont, tout d'abord, le niveau de représentation qui sera, pour VisiXML, de la moyenne fidélité (éventuellement convertie

par la suite en haute fidélité). Un second critère est la facilité d'utilisation ; ici aussi VisiXML s'alignera sur la majorité des outils de génération automatique et sera facile d'utilisation (en partie grâce au « drag and drop »). Il en ira de même pour les deux derniers critères concernant les développeurs ; VisiXML présentera les mêmes caractéristiques que les autres outils analysés, à savoir pas de contrôlabilité (ceci est dû à la complexité beaucoup trop importante qui résulterait de l'implémentation d'un processus de contrôle) et pas de possibilité de déboguage, du moins pas dans un premier temps (pour les mêmes raisons, n'étant pas un professionnel de l'utilisation de VBA, il me semble irréalisable de fournir un système de déboguage en adéquation avec l'outil).

Les deux derniers critères de la grille d'évaluation concernent l'utilisateur final des interfaces générées. La flexibilité de l'output sera forte, et sera donc un des avantages majeurs de VisiXML sur d'autres outils de génération automatique, du fait des spécifications de moyenne fidélité (cfr. supra et infra.). Pour la même raison, la représentativité des interfaces générées par VisiXML se veut être forte, ce qui dans ce cas se trouve dans la moyenne des autres outils analysés (cfr. infra).

3. Spécifications UsiXML

Afin de mieux situer l'approche et le choix d'UsiXML pour le développement de l'outil de génération automatique qui fait l'objet de ce mémoire, il convient de passer brièvement en revue l'intérêt et les avantages du métalangage XML en général, et d'UsiXML en particulier.

3.1. XML

XML (eXtensible Markup Language) est un métalangage qui permet de décrire des données de manière structurée grâce à l'emploi de balises. Il a été défini par le World Wide Web Consortium (W3C) afin d'assurer d'un côté une certaine indépendance des données vis-à-vis des plateformes et des applications, et de l'autre l'homogénéité des données structurées. A l'inverse de son prédécesseur, le HTML, il ne définit pas la manière dont il faut afficher les données, il définit les données elles-mêmes. C'est donc à proprement parler un standard qui régit la manière dont le contenu doit être codé, de façon à obtenir des spécifications uniformes, quelles que soient les plateformes ou les applications dont les données proviennent. Et cela fonctionne aussi dans l'autre sens, c'est-à-dire que sur base de spécifications XML qui décrivent les informations de manière homogène, un affichage adapté à chaque environnement peut être créé. C'est donc un moyen idéal d'uniformisation et de standardisation de la gestion de données hétérogènes telles que peuvent l'être toutes les informations en provenance de la toile.

Le XML permet, comme son nom l'indique, la création de balises spécifiques adaptées au mieux aux besoins de l'environnement dans lequel les informations sont destinées à être affichées. C'est cette propriété d'extensibilité du métalangage qui a été exploitée au sein de l'unité de systèmes d'information (ISYS) de l'IAG pour la définition d'UsiXML.

3.2. UsiXML

UsiXML (User Interface eXtensible Markup Language) est une extension conforme du métalangage XML qui a pour but de décrire les interfaces utilisateur (UI) pour des contextes d'utilisation multiples. Comme nous venons de le voir pour XML, il s'agit d'un langage déclaratif permettant la structuration et l'homogénéisation des données, relatives aux interfaces utilisateur dans le cas présent. Le but d'UsiXML est donc de décrire, à un niveau élevé d'abstraction, les éléments constitutifs des interfaces d'applications tels que champ de saisie de texte, boutons, fenêtres, etc.

De même que son grand frère, UsiXML déclare le contenu des interfaces de manière indépendante vis-à-vis des plateformes sur lesquelles celles-ci sont destinées à être exécutées. En outre, la manière dont les différent(e)s composants (balises) sont déclaré(e)s assure une double indépendance supplémentaire :

- l'indépendance de modalité : i.e. l'interface est décrite de manière autonome vis-à-vis des modalités de l'interaction de l'utilisateur avec elle (graphique, vocale,…)
- l'indépendance de dispositif (device) : i.e. l'interface est décrite de manière autonome vis-à-vis des dispositifs utilisés pour l'interaction entre l'utilisateur et elle-même (souris, clavier, écran tactile, …)

Le principe utilisé pour choisir les objets représentés en UsiXML est la création d'un sous-ensemble d'attributs, événements et primitives commun à la majorité des interfaces. Le but poursuivi par UsiXML n'est pas de fournir un nouveau langage pour l'implémentation des interfaces utilisateur mais bien l'intégration de langages divers déjà existants tels que Java, HTML, XHTML, C++, VRML, etc.

C'est cette dernière caractéristique qui est la plus intéressante dans le cadre de ce mémoire et qui a poussé à la création d'un outil de génération automatique d'interfaces prenant UsiXML comme langage d'output. En effet, comme cela a déjà été brièvement mentionné plus haut, le fait que des spécifications UsiXML valides forment une

description homogène d'interfaces indépendamment des plateformes, des moyens d'interaction et des modalités d'interaction induit la possible multiplicité d'outputs cohérents à peu de frais et à peu d'efforts supplémentaires. C'est cette flexibilité qui est recherchée avant tout dans le cadre du développement de VisiXML.

3.3. Etendue de la couverture d'UsiXML offerte par VisiXML.

Maintenant que nous savons pourquoi UsiXML est très intéressant pour le développement de VisiXML nous pouvons analyser la mesure dans laquelle les possibilités du premier sont exploitées dans le second. A cet effet, je passerai en revue dans ce chapitre l'ensemble des objets graphiques de la couche CUI (Concrete User Interface) d'UsiXML qui pourront être dessinés dans VisiXML et dont le code pourra donc être exporté en UsiXML. J'énumérerai les différents attributs de chaque objet ainsi que les valeurs que ces attributs admettent. Il est à noter que j'ai fait le choix de développement de laisser au concepteur de l'interface la possibilité de saisir la valeur de tous les attributs, qu'ils soient requis ou optionnels. Ce choix implique que tous les attributs auront un champ particulier dans VisiXML et que, pour ne pas induire plus d'erreurs en générant par exemple des attributs facultatifs non désirés, les champs laissés vides dans le programme seront systématiquement ignorés lors de la génération. Il paraît dès lors évident que seuls les attributs requis présenteront une valeur par défaut dans VisiXML (bien que certains attributs non requis en aient une dans UsiXML).

Il convient ici de distinguer les deux types de support possibles de VisiXML pour les objets et leurs attributs. Le premier type de support, que nous pourrions dénommer support des spécifications, concerne les spécifications générées par VisiXML ; on peut dire que l'objet ou l'attribut est supporté dès qu'il y a moyen de le définir dans les spécifications à l'aide de l'outil, c'est-à-dire dès qu'il y a moyen d'en introduire la valeur à un endroit quelconque de VisiXML, en l'occurrence ce sera via la fenêtre de custom properties dans la majorité des cas. C'est ce type de support dont il est question dans le paragraphe précédent, en effet comme nous venons de le dire, le code de l'ensemble des objets graphiques de la CUI pourra être généré dans les spécifications, nous avons donc

un support global des spécifications. Le second type de support est ce que nous pourrions qualifier de support de représentation ou support visuel ; il détermine dans quelle mesure l'utilisateur de VisiXML aura, pour un objet déterminé, un aperçu représentatif de celui-ci et de ses attributs lors du drag and drop sur la surface de travail. Pour ce type de support on peut énumérer 4 cas de figure possibles :

- l'objet UsiXML se retrouve directement représenté dans VisiXML.
- l'objet UsiXML ne peut pas être représenté dans VisiXML, il n'existe pas de support visuel pour lui. On le simule alors au moyen d'une image statique.
- l'objet UsiXML est représenté dans VisiXML mais avec des déformations, des simplifications ou des limitations.
- VisiXML pourrait également dans un quatrième cas de figure comporter des objets qui n'ont pas de contrepartie dans UsiXML.

L'énumération des différents objets comportera donc en plus de leurs attributs une discussion sur le support visuel dont ils font preuve. Toutes les données de cette énumération étant relatives aux spécifications du langage UsiXML, elles proviennent directement des déclarations de celui-ci [Usi04a] et de la documentation draft disponible à son sujet [Usi04b].

3.3.1. Cio (Concrete Interaction Object)

Hérite des attributs de : néant.
Un Cio est toute entité d'une UI (User Interface) que les utilisateurs peuvent percevoir (texte, image, animation,…) et/ou manipuler (bouton, checkbox, …). Il sera caractérisé par les attributs suivants :

- id : est l'attribut qui permet d'identifier un Cio et ne reflète ni son type ni son contenu. *Type : string. Valeur par défaut : néant.*
- name : est le nom qui est donné à un Cio, cet attribut donne un aperçu du type, de la fonction ou du contenu du Cio. *Type : string. Valeur par défaut : néant.*

- icon : est l'image associée au Cio, représentée par son adresse. *Type : uri. Valeur par défaut : néant.*
- content : est le contenu textuel associé à tout Cio, représenté également par son adresse. *Type : uri. Valeur par défaut : néant.*
- help : est l'aide associée à tout Cio. *Type : string. Valeur par défaut : néant.*
- defaultIcon : est une image par défaut pour tout Cio. *Type : string. Valeur par défaut : néant.*
- defaultContent : est la légende par défaut pour tout Cio. *Type : string ou uri. Valeur par défaut : néant.*
- defaultHelp : est l'aide par défaut pour tout Cio. *Type : uri. Valeur par défaut : néant.*

3.3.2. graphicalCio

Hérite des attributs de : Cio.
Un graphicalCio est tout élément composant une GUI (Graphical User Interface). Il peut être soit un container soit un composant individuel et sera caractérisé par les attributs suivants :

- isVisible : a pour valeur « true » si le graphicalCio est visible. *Type : boolean. Valeur par défaut : true.*
- isEnabled : a pour valeur « true » si le graphicalCio est activé. *Type : boolean. Valeur par défaut : true.*
- fgColor : est la couleur de l'avant-plan d'un graphicalCio exprimée en code couleurs HTML. *Type : string. Valeur par défaut : néant.*
- bgColor : est la couleur de l'arrière-plan d'un graphicalCio exprimée en code couleurs HTML. *Type : string. Valeur par défaut : néant.*
- toolTipContent : est le tooltip (texte apparaissant quand on passe la souris au-dessus d'un objet) de tout graphicalCio et est représenté par son adresse. *Type : uri. Valeur par défaut : néant.*
- toolTipDefaultContent : est le tooltip associé à un graphicalCio par défaut. *Type : string. Valeur par défaut : néant.*

- transparencyRate : est l'attribut donnant le taux de transparence d'un graphicalCio exprimé en pourcent. *Type : integer. Valeur par défaut : néant.*
- borderWidth : donne la largeur de la bordure du graphicalCio. *Type : integer. Valeur par défaut : néant.*
- borderType : donne le type de bordure du graphicalCio. *Type : integer. Valeur par défaut : néant.*
- borderTitle : est le titre de la bordure, i.e. le texte apparaissant dans la bordure d'un graphicalCio. *Type : string. Valeur par défaut : néant.*
- defaultBorderTitle : est le titre par défaut de la bordure d'un graphicalCio. *Type : string. Valeur par défaut : néant.*
- borderTitleAlign : donne l'alignement du titre de la bordure du graphicalCio. Les valeurs autorisées sont : « left », « middle » et « right ». *Type : string. Valeur par défaut : néant.*
- borderColor : est la couleur de la bordure exprimée en code couleurs HTML. *Type : string. Valeur par défaut : néant.*

3.3.3. graphicalContainer

Hérite des attributs de : Cio, graphicalCio.

Un graphicalContainer est un élément contenant une collection de Cios qui supportent l'exécution d'un ensemble de tâches sémantiquement ou logiquement connectées. Il sera caractérisé par les attributs suivants :

- width : est la largeur du graphicalContainer exprimée en pixels. *Type : integer. Valeur par défaut : néant.*
- height : est la hauteur du graphicalContainer exprimée en pixels. *Type : integer. Valeur par défaut : néant.*
- bgImage : est l'image d'arrière-plan d'un graphicalContainer, représentée par son adresse. *Type : uri. Valeur par défaut : néant.*
- isAlwaysOnTop : a pour valeur « true » si le graphicalContainer est toujours visible au dessus des autres éléments. *Type : boolean. Valeur par défaut : néant.*

- repetition : indique le nombre de fois qu'un graphicalContainer est répété dans la spécification. *Type : integer. Valeur par défaut : néant.*

3.3.3.1. box

Hérite des attributs de : Cio, graphicalCio, graphicalContainer.
Une box est un container qui permet une structuration non ambigue des graphicalIndividualComponents au sein d'une fenêtre, d'une dialogBox ou d'un tabbedItem. Les boxes sont incorporées les unes aux autres et peuvent être de type « main », « horizontal » ou « vertical ». Leurs attributs sont les suivants :

- type : donne le type de la box. Les valeurs autorisées sont « vertical », « horizontal », « verticalGrid », « horizontalGrid » et « stack ». *Type : string. Valeur par défaut : néant.*
- relativeWidth : exprime en pourcent la largeur relative d'une box. *Type : integer. Valeur par défaut : néant.*
- relativeHeight: exprime en pourcent la hauteur relative d'une box. *Type : integer Valeur par défaut : néant.*
- isSplittable : indique si une box est splittable. *Type : boolean. Valeur par défaut : néant.*
- isDetachable : indique si oui ou non une box peut être détachée de son graphicalContainer. *Type : boolean. Valeur par défaut : néant.*
- isBalanced : indique si tous les graphicalIndividualComponents sont topologiquement équilibrés au sein d'une box. *Type : boolean. Valeur par défaut : néant.*
- isResizableHorizontal : prend la valeur « true » si une box est redimensionnable horizontalement. *Type : boolean. Valeur par defaut : néant.*
- isResizableVertical : prend la valeur « true » si une box est redimensionnable verticalement. *Type : boolean. Valeur par défaut : néant.*
- relativeMinWidth : indique en pourcentage la largeur relative initiale minimale d'une box . *Type : integer. Valeur par défaut : néant.*

- relativeMinHeight : indique en pourcentage la hauteur relative initiale minimale d'une box. *Type : integer. Valeur par défaut : néant.*
- isFlow : indique à l'algorithme d'affichage s'il doit créer ou non une nouvelle ligne dans une box si tous les éléments qui lui appartiennent ne peuvent être contenus dans une seule ligne. *Type : boolean. Valeur par défaut : néant.*
- isScrollable : indique si la box peut être équippée d'une scrollbar si son contenu ne peutêtre disposé entièrement sur sa surface.
- gridWidth : indique la largeur en nombres absolus d'une box dans le cas où elle est de type Grid. *Type : string. Valeur par défaut : néant.*
- gridHeight : indique la hauteur en nombres absolus d'une box dans le cas où elle est de type Grid. *Type : string. Valeur par défaut : néant.*

En termes de support visuel, l'élément box est directement représenté dans VisiXML et l'utilisateur en a donc un bon aperçu lorsqu'il les dépose sur la surface de travail.

3.3.3.2. table

Hérite des attributs de : Cio, graphicalCio, graphicalContainer.
Une table est un container composé de cellules. Elle est cractérisée par sa taille et peut être uni-, bi- ou tridimensionnelle. Ses attributs sont les suivants :

- xSize : désigne le nombre de lignes d'une table. *Type : integer. Valeur par défaut : néant.*
- ySize : désigne le nombre de colonnes d'une table. *Type : integer. Valeur par défaut : néant.*
- zSize : désigne le nombre de couches d'une table. *Type : integer. Valeur par défaut : néant.*

En ce qui concerne le support visuel, l'élément table n'est pas réellement possible à représenter dans VisiXML, notamment à cause de son attribut zSize qui impliquerait, pour avoir une représentation parfaite, d'avoir recours à une vision 3D dynamique grâce à laquelle les cellules pourraient être incorporées au tableau directement dans la bonne

couche. Cela étant malheureusement impossible dans l'état actuel de mes connaissances l'objet est simulé par une image statique de tableau à deux dimensions.

3.3.3.3. cell

Hérite des attributs de : Cio, graphicalCio, graphicalContainer.
Est contenu dans une table et peut en contenir une. L'élément cell est un container en lui-même mais doit toujours faire partie d'une table. Ses attributs sont les suivants :

- xIndex : désigne un numéro de ligne. *Type : integer. Valeur par défaut : néant.*
- yIndex : désigne un numéro de colonne. *Type : integer. Valeur par défaut : néant.*
- zIndex : désigne un numéro de couche. *Type : integer. Valeur par défaut : néant.*

Pour les mêmes raisons que la table, la cellule ne peut pas être directement représentée dans VisiXML, en effet c'est sa position dans la table qui devrait dynamiquement déterminer ses attributs xIndex, yIndex et zIndex. On a donc également recours ici à une simulation de l'objet par une image statique le représentant.

3.3.3.4. dialogBox

Hérite des attributs de : Cio, graphicalCio, graphicalContainer.
L'élément dialogBox est une dialogBox classique telle qu'on peut la voir dans bon nombre d'interfaces. En termes de support visuel la dialogBox est donc directement supportée dans VisiXML. Elle n'a pas d'attributs spécifiques.

3.3.3.5. window

Hérite des attributs de : Cio, graphicalCio, graphicalContainer.
L'élément window représente toute fenêtre dans une interface et ses attributs sont :

- windowLeftMargin : indique la taille de la marge gauche de la fenêtre en pixels. *Type : integer. Valeur par défaut : néant.*

- windowTopMargin : indique la taille de la marge supérieure de la fenêtre en pixels. *Type : integer. Valeur par défaut : néant.*
- isResizable : indique si une fenêtre est redimensionnable ou non. *Type : boolean. Valeur par défaut : true.*

Pour ce qui est du support visuel, l'élément window n'est que partiellement supporté par VisiXML, en effet si son apparence à l'écran correspond bien à la représentation standard d'une fenêtre d'application et bien que son attribut title soit directement éditable à l'écran, certains autres attributs ne le sont pas, c'est notamment le cas de windowTopMargin et windowLeftMargin. On a donc bien ici une représentation simplifiée de l'objet.

3.3.3.6. tabbedDialogBox

Hérite des attributs de : Cio, graphicalCio, graphicalContainer.
Une tabbedDialogBox est un ensemble dialogBoxes ou chaque dialogBox est accessible via un mécanisme d'onglets. Elle est composée de tabbedItems et ne possède aucun attribut spécifique. En termes de support visuel, la dialogBox est directement supportée par VisiXML.

3.3.3.7. tabbedItem

Hérite des attributs de : Cio, graphicalCio, graphicalContainer.
Un tabbedItem est un type de container qui compose une tabbedDialogBox, il y en a un pour chaque onglet dans une tabbedDialogBox. Son attribut spécifique est :

- index : est l'odre relatif d'un tabbedItem vis-à-vis des autres tabbedItems. L'index 0 est placé à l'extrême gauche. *Type : integer. Valeur par défaut : néant.*

Le support visuel est partiel pour le tabbedItem, en effet l'attribut index n'est pas directement généré sur base de la position de l'onglet à l'écran, nous avons donc une représentation simplifiée du tabbedItem.

3.3.4. graphicalIndividualComponent

Hérite des attributs de : Cio, graphicalCio.
Un graphicalIndividualComponent est un graphicalCio contenu dans un graphicalContainer. Ses attributs spécifiques permettent de définir sa position au sein de ce graphicalContainer et sont les suivants :

- glueVertical : est la spécification d'une contrainte d'affichage sur un axe vertical entre le graphicalIndividualComponent et son graphicalContainer. Les valeurs autorisées sont : « top », « middle » et « bottom ». *Type : string. Valeur par défaut : néant.*
- glueHorizontal : est la spécification d'une contrainte d'affichage sur un axe horizontal entre le graphicalIndividualComponent et son graphicalContainer. Les valeurs autorisées sont : « left », « middle » et « right ». *Type : string. Valeur par défaut : néant.*
- mnemonic : est le mnémonique (aide à la mémoire) associé au graphicalIndividualComponent. *Type : string. Valeur par défaut : néant.*
- defaultMnemonic : est le mnemonique associé par défaut au graphicalOndividualComponent. *Type : string. Valeur par défaut : néant.*

3.3.4.1. textComponent

Hérite des attributs de : Cio, graphicalCio, graphicalIndividualComponent.
C'est un composant qui sert à la prise en charge du contenu textuel, il présente els attributs suivants :

- textFont : permet de spécifier le style de police du textComponent. *Type : string. Valeur par défaut : néant.*
- isBold : prend la valeur « true » si le textComponent est en gras. *Type : boolean. Valeur par défaut : néant.*
- isItalic : prend la valeur « true » si le textComponent est en italique. *Type : boolean. Valeur par défaut : néant.*

- isUnderline : prend la valeur « true » si le textComponent est souligné. *Type : boolean. Valeur par défaut : néant.*
- isStrikethrough : prend la valeur « true » si le textComponent est barré. *Type : boolean. Valeur par défaut : néant.*
- isSubscript : prend la valeur « true » si le textComponent est en subscript. *Type : boolean. Valeur par défaut : néant.*
- isSuperscript : prend la valeur « true » si le textComponent est en superscript. *Type : boolean. Valeur par défaut : néant.*
- isPreformatted : si cet attribut a la valeur « true » aucune des caractéristiques de style ne pourra être modifiée en dehors des spécifications. *Type : boolean. Valeur par défaut : néant.*
- textSize : permet de spécifier la taillet en points pour le textComponent. *Type : integer. Valeur par défaut : néant.*
- hyperLinkTarget : désigne un fichier ciblé par hyperlien. *Type : uri. Valeur par défaut : néant.*
- defaultHyperLinkTarget : est l'hyperlien par défaut. *Type : uri. Valeur par défaut : néant.*
- linkVisitedColor : définit la couleur que prend un hyperlien après avoir été visité une fois. *Type : string. Valeur par défaut : néant.*
- activelinkColor : définit la couleur que prend un hyperlien lorsqu'on clique dessus. *Type : string. Valeur par défaut : néant.*
- textMargin : spécifie la largeur de la marge du texte en pixels. *Type : integer. Valeur par défaut : néant.*
- textColor : spécifie la couleur du texte en code couleurs HTML. *Type : string. Valeur par défaut : néant.*
- isEditable : spécifie si le textComponent est éditable, i.e. sumis a l'input de l'utilisateur ou non. *Type : boolean. Valeur par défaut : néant.*
- wordWrapped : définit si le texte est « wrapped » ou non. *Type : boolean. Valeur par défaut : néant.*
- forceWordWrapped : indique si le wrapping du textComponent est forcé ou pas. Si cet attribut a la valeur « true » les mots pourront être divisés. Cet attribut ne

peut avoir de valeur que si wordWrapped a la valeur « true ». *Type : boolean. Valeur par défaut : néant.*

- maxLength : définit la longueur maximale du contenu d'un textComponent exprimée en nombre de caractères. *Type : integer. Valeur par défaut : néant.*
- numberOfColumns : définit le nombre de colonnes du textComponent. *Type : integer. Valeur par défaut : néant.*
- numberOfLines : définit le nombre de lignes du textComponent. *Type : integer. Valeur par défaut : néant.*
- scrollstyle : définit le style de déroulement du textComponent. Les valeurs autorisées sont « scroll », « slide » et « alternate ». *Type : string. Valeur par défaut : néant.*
- scrollDirection : définit la direction du déroulement. Les valeurs autorisées sont « left » et « right ». *Type : string. Valeur par défaut : néant.*
- scrollWidth : indique la largeur en pixels d'un textComponent déroulant. *Type : integer. Valeur par défaut : néant.*
- scrollHeight : indique la hauteur en pixels d'un textComponent déroulant. *Type : integer. Valeur par défaut : néant.*
- scrollHorizSpace : indique la largeur en pixels d'un espace blanc à la droite et à la gauche d'un textComponent déroulant. *Type : integer. Valeur par défaut : néant.*
- scrollVertSpace : indique la hauteur en pixels d'un espace blanc en haut et en bas d'un textComponent déroulant. *Type : integer. Valeur par défaut : néant.*
- scrollDelay : exprime un délai de déroulement en millisecondes. *Type : integer. Valeur par défaut : néant.*
- scrollAmount : exprime en pixels la distance déroulée après chaque scrollDelay. *Type : integer. Valeur par défaut : néant.*
- textVerticalAlign : indique une contrainte d'alignement vertical du texte contenu dans un textComponent. Les valeurs autorisées sont « top », « middle » et « bottom ». *Type : string. Valeur par défaut : néant.*
- textHorizontalAlign : indique une contrainte d'alignement horizontal du texte contenu dans un textComponent. Les valeurs autorisées sont « left », « middle » et « right ». *Type : string. Valeur par défaut : néant.*

- filter : est un expression régulière constraignant la teneur d'un textComponent. *Type : string. Valeur par défaut : néant.*

En termes de support visuel, l'élément textComponent est probablement l'objet indirectement représenté qui admet le plus de simplifications. La liste en est longue, signalons seulement que, si le contenu textuel est directement éditable à l'écran, d'autres attributs tels que l'alignement et le format du texte ne le sont pas. Les attributs d'un composant textuel avec un format de texte riche ne sauront donc pas être directement représentés sur la surface de travail de VisiXML et l'utilisateur aura donc affaire à une représentation simplifiée de celui-ci.

3.3.4.2. imageComponent

Hérite des attributs de : Cio, graphicalCio, graphicalIndividualComponent.
C'est un composant qui sert à la prise en charge du contenu graphique, il présente les attributs suivants :

- imageHeight : donne la hauteur d'un imageComponent exprimée en pixels. *Type : integer. Valeur par défaut : néant.*
- imageWidth : donne la largeur d'un imageComponent exprimée en pixels. *Type : integer. Valeur par défaut : néant.*
- imageHorizSpace : exprime en pixels un offset horizontal à respecter avec le container. *Type : integer. Valeur par défaut : néant.*
- imageBorder : exprime la largeur de la bordure de l'imageComponent en pixels. *Type : integer. Valeur par défaut : néant.*
- hyperlinkTarget : spécifie un fichier cible atteignable à partir d'un imageComponent. *Type : uri. Valeur par défaut : néant.*
- defaultHyperlinkTarget : est l'hyperlien par défaut. *Type : uri. Valeur par défaut : néant.*

En termes de support visuel, l'imageComponent n'est pas supporté dans VisiXML et l'on a recours à une image statique pour le représenter.

3.3.4.3. videoComponent

Hérite des attributs de : Cio, graphicalCio, graphicalIndividualComponent.
C'est un composant qui sert à la prise en charge du contenu vidéo, il admet les attributs suivants :

- alternateImage : définit l'image qui sera visible dans le cas où le composant vidéo est inaccessible, quelle qu'en soit la raison. *Type : uri. Valeur par défaut : néant.*
- autoplay : prend la valeur « true » si la vidéo se lance automatiquement. *Type : boolean. Valeur par défaut : néant.*
- loop : prend la valeur « true » si la vidéo est affichée en boucle. *Type : boolean. Valeur par défaut : néant.*
- builtInControl : spécifie si un videoComponent est équippé des boutons de contrôle standards tels que play, pause et stop. *Type : boolean. Valeur par défaut : néant.*
- subtitle : prend la valeur « true » si le videoComponent est équippé de sous-titres. *Type : boolean. Valeur par défaut : néant.*
- subtitleContent : définit le contenu des sous-titres du videoComponent. *Type : uri. Valeur par défaut : néant.*

Même remarque que pour l'imageComponent en ce qui concerne le support visuel.

3.3.4.4. button

Hérite des attributs de : Cio, graphicalCio, graphicalIndividualComponent.
Ce composant est également appelé bouton de déclenchement car son but est de déclencher n'importe quelle action disponible dans le système. Il ne présente pas d'attributs spécifiques et est directement représenté sous VisiXML en termes de support visuel.

3.3.4.5. checkBox

Hérite des attributs de : Cio, graphicalCio, graphicalIndividualComponent.
Ce composant permet d'effectuer un choix booléen en cochant la case disponible à côté d'un intitulé. Il admet les attributs suivants :

- defaultState : définit l'état de la checkBox par défaut. *Type : boolean. Valeur par défaut : néant.*
- groupName : est le nom du groype de checkBoxes lorsqu'il y en a un. *Type : string. Valeur par défaut : néant.*

L'élément checkbox est également directement représenté sous VisiXML en termes de support visuel.

3.3.4.6.radioButton

Hérite des attributs de : Cio, graphicalCio, graphicalIndividualComponent.
Ce composant permet d'effectuer un choix booléen en cochant le cercle disponible à côté d'un intitulé. Il se différentie des checkBoxes par le fait que le choix effectué dans un groupe de radioButtons est mutuellement exclusif alors qu'un groupe de checkBoxes admettra plusieurs choix simultanés. Il présente les attributs suivants :

- defaultState : définit l'état du radioButton par défaut. *Type : boolean. Valeur par défaut : néant.*
- groupName : est le nom du groype de radioButtons. *Type : string. Valeur par défaut : néant.*

Le radioButton est directement représenté sous VisiXML, la figure suivante nous en offre un aperçu :

3.3.4.7. toggleButton :

Hérite des attributs de : Cio, graphicalCio, graphicalIndividualComponent.
Ce composant permet d'effectuer un choix booléen en enfoncant ou non un bouton multi-états. Il possède l'attribut suivant :

- defaultState : définit l'état du toggleButton par défaut (enfoncé ou non). *Type : boolean. Valeur par défaut : néant.*

Le toggleButton est lui aussi directement représenté sous VisiXML.

3.3.4.8. tree

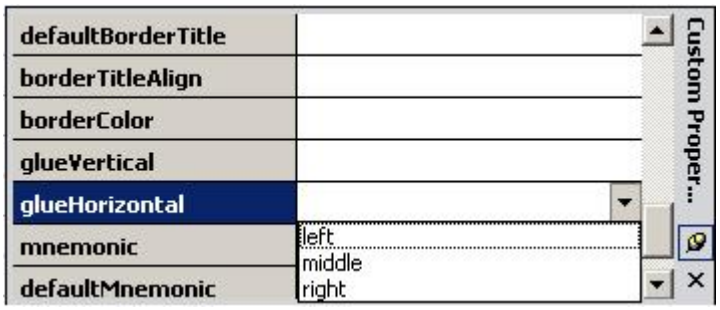
Hérite des attributs de : Cio, graphicalCio, graphicalIndividualComponent.
Le composant tree permet de représenter une structure hiérarchique d'objets. Il n'admet pas d'attributs spécifiques. En raison de sa nature assez complexe, l'élément tree ne peut être représenté sous VisiXML et on donc recours à sa simulation par une image statique.

3.3.4.9. spin

Hérite des attributs de : Cio, graphicalCio, graphicalIndividualComponent.
Le spin est le composant qui permet une navigation séquentielle dans une collection d'objets. Il admet l'attribut suivant :

- orientation : donne l'orientation du spin, les deux valeurs autorisées sont « horizontal » et « vertical ». *Type : string. Valeur par défaut : néant.*

En termes de support visuel, le spin est directement représenté sous VisiXML, on pourra même remarquer sa division en deux éléments distincts en fonction de l'orientation. Ci-dessous, l'élément spin horizontal ainsi qu'une partie des attributs qu'il admet, affichés dans la fenêtre des custom properties sous Visio :



On pourra remarquer au passage le menu déroulant associé aux attributs n'admettant que quelques valeurs distinctes, ce qui restreint le nombre d'erreurs.

3.3.4.10. slider

Hérite des attributs de : Cio, graphicalCio, graphicalIndividualComponent.
Ce composant permet la sélection d'une ou deux valeurs entières dans un ensemble d'entiers ordonnés. Il admet les attributs suivants :

- minValue : définit la limité inférieure des valeurs du slider. *Type : integer. Valeur par défaut : néant.*
- maxValue : définit la limité supérieure des valeurs du slider. *Type : integer. Valeur par défaut : néant.*
- step : sert à préciser les valeurs intermédiaires du slider entre minValue et maxValue. *Type : integer. Valeur par défaut : néant.*
- orientation : sert à définir l'orientation du slider et peut prendre les valeurs « vertical » ou « horizontal ». *Type : string. Valeur par défaut : néant.*

En termes de support visuel, la représentation du slider est simplifiée en ce sens que les limites supérieure et inférieure du slider sont définies via la fenêtre de custom properties et non pas via la shape elle-même.

3.3.4.11. comboBox

Hérite des attributs de : Cio, graphicalCio, graphicalIndividualComponent.

Ce composant permet une sélection directe parmi une collection d'objets ordonnés séquentiellement. Il admet les attributs suivants :

- isDropDown : spécifie si la comboBox prend la forme d'un menu drop down. *Type : boolean. Valeur par défaut : néant.*
- isEditable : spécifie si le contenu de la textBox composante de la comboBox est éditable ou pas. *Type : boolean. Valeur par défaut : néant.*

Pour ce qui est du support visuel, la comboBox est directement représentée sous VisiXML, en voici un aperçu :

Le texte qui se trouve au milieu de l'élément est directement éditable sous VisiXML et se retrouve par la suite dans les spécifications UsiXML générées sous l'attribut « content ».

3.3.4.12. menu

Hérite des attributs de : Cio, graphicalCio, graphicalIndividualComponent.

Ce composant admet la sélection d'un ou plusieurs menuItems dans une liste. Un choix dans la liste de menuItems entraîne toujours un changement dans l'état de l'interface. Il admet les attributs suivants :

- popUpMenu : prend la valeur « true » si le menu est un menu de type pop up. *Type : boolean. Valeur par défaut : néant.*
- toolBarMenu : prend la valeur « true » si le menu est un menu de type barre d'outils. *Type : boolean. Valeur par défaut : néant.*

L'objet menu est directement représenté sous VisiXML, il offre un support visuel complet.

3.3.4.13. menuItem

Hérite des attributs de : Cio, graphicalCio, graphicalIndividualComponent.

C’est le composant dont sont faits les menus, il possède une image, un contenu et un type. Il admet les attributs suivants :

- type : définit le type du menuItem, les valeurs autorisées sont « commande », « dialogue », « sub-menu », « toggle », « radio » et « separator ». *Type : string. Valeur par défaut : néant.*
- keyboardShortcut : définit le raccourci clavier du menuItem. *Type : string. Valeur par défaut : néant.*
- defaultKeyboradShortcut : définit le raccourci clavier par défaut. *Type : string. Valeur par défaut : néant.*

Même remarque que pour le menu en ce qui concerne le support visuel.

3.3.4.14. drawingCanvas

Hérite des attributs de : Cio, graphicalCio, graphicalIndividualComponent.
Ce composant est une surface qui permet le dessin de formes variées par manipulation directe. Il ne possède pas d’attributs spécifiques et est simulé par une image statique dans VisiXML.

3.3.4.15. colorPicker, fontPicker, datePicker, filePicker, hourPicker et progressionBar

Ces composants héritent des attributs de : Cio, graphicalCio, graphicalIndividual-Component.
Ils permettent respectivement la sélection d’une couleur, une police, une date, un fichier, une heure et l’affichage d’un état d’avancement. Aucun d’eux ne présente d’attributs spécifiques et ils sont actuellement tous simulés par des images statiques en ce qui concerne leur support visuel, cependant un support visuel complet devrait leur être affecté dans une version future de VisiXML.

Un petit tableau récapitulatif permettant de jauger de manière globale l’étendue du support de VisiXML pour les composants d’UsiXML est fourni en Annexe IV.

4. Développement du maquetteur d'interfaces.

4.1. Choix de développement

Rappelons tout d'abord que l'objectif de ce mémoire est de fournir un outil de génération d'interfaces accessible au plus grand nombre qui présente des délais de création relativement courts tout en respectant un niveau de fidélité élevé. Le fait que l'outil doive être facile à utiliser a rapidement poussé au choix d'une approche « drag and drop » et il se fait que Microsoft Visio possède tous les atouts requis pour une telle approche. En effet, Visio présente déjà à la base un ensemble conséquent d'objets graphiques et de widgets facilement réutilisables et a la particularité d'intégrer totalement le développement de solutions en VBA (Visul Basic for Applications). Il semblait donc plus facile de développer l'outil au moyen de cette combinaison plutôt que d'employer un éditeur d'interfaces quelconque couplé à un langage de programmation qui ne lui est pas automatiquement intégré. C'est d'ailleurs également l'environnement de développement qui a été utilisé par P.J.Molina et H.Traetteberg pour les outils qui ont été détaillés plus haut, comme la figure suivante le laisse apercevoir (Fig.4.1) :

Fig.4.1 : Stencils Visio pour le dessin de modèles DiaMODL

Les principaux éléments du « drag and drop » sous Visio sont les stencils, ces feuilles reprenant un certain nombre de « masters », c'est-à-dire d'icônes représentant un type particulier de formes (ou shapes). Ces « masters » sont glissés puis déposés sur la

feuille de travail où ils prennent la forme de la shape qu'ils représentent. La figure suivante nous montre le stencil « Windows and Dialogs » dont l'élément « Blank form » est en train d'être déposé sur la feuille de travail, où il abandonne sa forme d'icône pour prendre sa forme définitive.

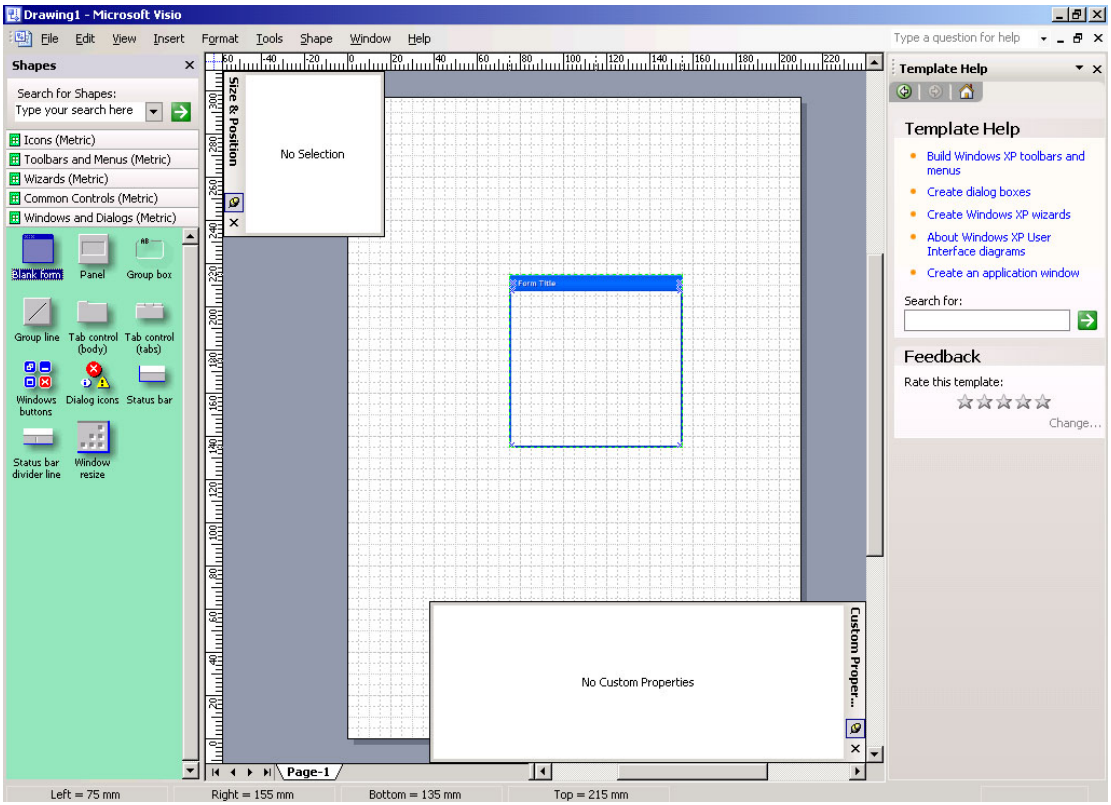


Fig.4.2: Environnement de travail de Microsoft Visio

Tous les objets de Visio présentent un certain nombre de propriétés prédéfinies et le programme offre aux utilisateurs la possibilité de spécifier pour chaque objet un ensemble de « custom properties » qui lui seront spécifiques.

J'ai donc créé un stencil pour les deux types d'objets présents dans la CUI d'UsiXML, reprenant à chaque fois tous les objets de ce type ainsi que leurs attributs sous la forme de « custom properties ». Un objet tel que le « textComponent » d'UsiXML sera donc représenté par une icône sur le stencil des « Individual Components » présentant une cinquantaine de custom properties. Ce choix de représentation des attributs sous Visio est à vrai dire pratique car le simple drop d'une shape sur la surface

de dessin affichera automatiquement dans la fenêtre correspondante la liste de tous les attributs disponibles pour l'objet représenté.

4.2. VisiXML

Maintenant que le cadre est posé (nous avons un utilisateur qui va glisser et déposer des shapes sur une feuille de travail et définir pour celles-ci un certain nombre de « custom properties ») il s'agit de déterminer comment convertir ces actions en spécifications UsiXML.

En y réfléchissant, cette conversion peut être séparée en deux tâches élémentaires distinctes ; tout d'abord la création d'une structure hiérarchique qui permettra de spécifier les différents éléments dans le bon ordre et ensuite la génération des spécifications pour un élément pris à part. La première solution qui a été envisagée était la création de vecteurs reprenant dans une première ligne les différents éléments et dans une seconde les éléments y étant inclus. Il s'avéra que le parcours de ce genre de vecteurs afin d'obtenir le bon ordre de déclaration n'avait rien de facile et qu'il était plus intéressant de passer par la création d'arbres logiques qui seraient par la suite plus intuitivement parcourables.

J'ai donc eu recours à l'élément Treeview de VBA qui permet la représentation hiérarchisée des différents éléments à l'écran mais qui présente le désavantage de n'être accessible que pendant le runtime c'est-à-dire le moment où l'application s'exécute. Cela implique une impossibilité de donner à l'utilisateur de VisiXML un aperçu de sa création en temps réel. Ce désavantage étant, deux boutons sont mis à la disposition de l'utilisateur dans VisiXML ; le premier permet la génération du Treeview comprenant tous les éléments dessinés à l'écran et le second sert à générer les spécifications UsiXML sur base d'un parcours en profondeur du Treeview précédemment créé, par le biais d'un algorithme de parcours récursif (voir infra)

Il est à noter que la hiérarchisation des shapes dans le Treeview se base actuellement sur un test d'inclusion géographique standard, basé sur la minimisation de la somme des différences de coordonnées et d'abscisses. En voici le détail :

- Toutes les shapes sont géographiquement identifiées par les coordonnées de leur coin supérieur gauche et de leur coin inférieur droit (Fig.4.3):

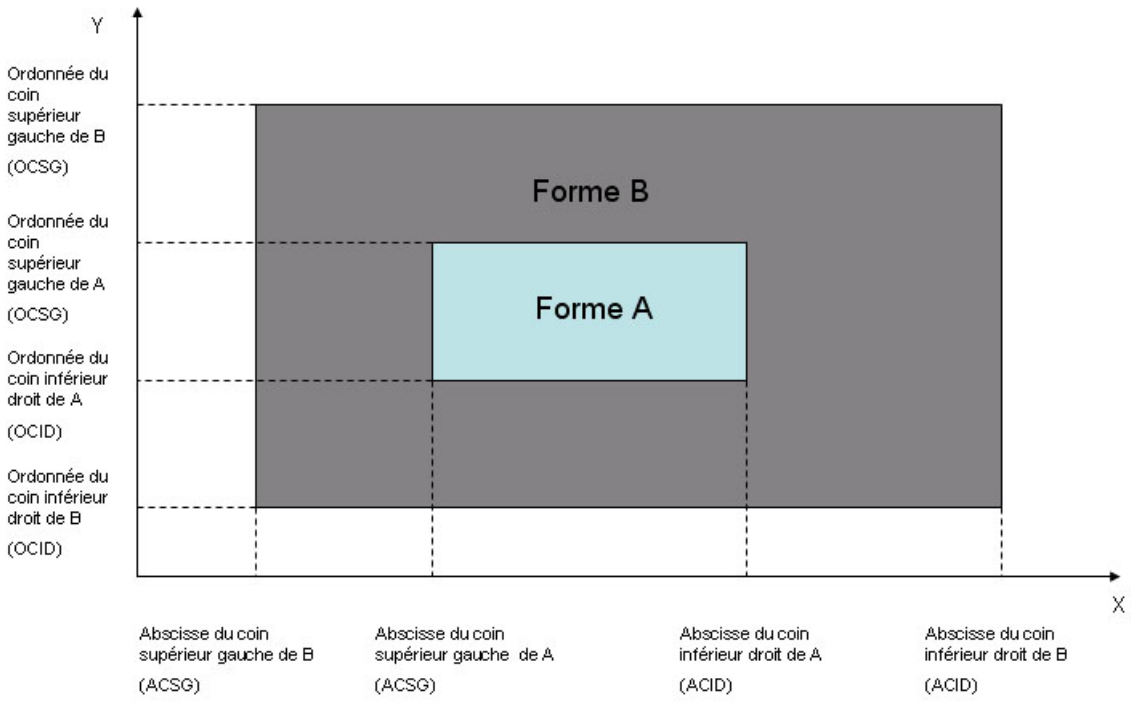


Fig.4.3 : Reconnaissance de la position des shapes via leurs coordonnées.

- La seconde partie du test d'inclusion va ensuite passer toutes les shapes en revue afin de comparer les 4 coordonnées relatives à chacune d'elles :

```
sommeMini = 99999
for i = 1 to shapes.count
shapeActive = shapes (i)
    for j = 1 to shapes.count
            if shapeActive.ACSG >= shapes(i).ACSG then
                    if shapeActive.ACID <= shapes(i).ACID then
                            if shapeActive.OCSG <= shapes(i).OCSG then
                                    if shapeActive.OCID >= shapes(i).OCID then
```

if [shapeActive.ACSG – shapes(i).ACSG + shapes(i).ACID – shapeActive.ACID + shapes(i).OCSG – shapeActive.OCSG + shapeActive.OCID – shapes(i).OCID] <= sommeMini then
sommeMini = [shapeActive.ACSG – shapes(i).ACSG + shapes(i).ACID – shapeActive.ACID + shapes(i).OCSG – shapeActive.OCSG + shapeActive.OCID – shapes(i).OCID]
end if

end if
end if
end if
end if
end if

- Il ne reste plus à la fin de l'exécution qu'à reprendre la shape j dont les coordonnées ont donné la somme la plus petite et c'est le parent le plus proche de notre shape active i et il ne reste donc plus qu'à inclure cette dernière dans notre Treeview comme enfant de j.

Une fois que le test d'inclusion a été effectué sur toutes les shapes de la page on obtient un Treeview complet reprenant toutes les shapes et leurs enfants. Il faut alors parcourir ce Treeview afin d'y effectuer dans le bon ordre la deuxième tâche élémentaire qui a été identifiée : la génération des spécifications pour un élément particulier. Cette tâche se composant d'instructions VBA sans grand intérêt je la représenterai donc dans le détail de l'algorithme récursif par « (generation) ». Voici le détail :

Function recExplore (n : nœud, profondeur : nombre entier)
profondeur = profondeur + 1
set shapeActive = shapes(n)

```
        (generation) pour shapeActive
    If shapeActive a des enfants then
        Set nC = enfant de n
        Do
                Recexplore (nC, profondeur)
                If nC est le dernier enfant de n then
                        Set Parent = n
                        (generation)balise de fermeture  pour n
                        Exit Do
                Set nC = next
        Loop
end if
profondeur = profondeur – 1
end function
```

A la fin de son exécution, la fonction aura généré les spécifications UsiXML de toutes les shapes présentes sur la page dans l'ordre hiérarchique de leur représentation au sein du Treeview.

4.3. Code de VisiXML

Le code de l'application est fourni en Annexe II, il comporte pour le moment 383 lignes et 5 procédures. Les deux premières procédures servent à la customisation de l'environnement de travail sous Microsoft Visio, notamment par l'élimination à l'écran des barres d'outils standard et l'adjonction de la barre d'outils spécifique à VisiXML. Les trois dernières procédures servent, comme nous venons de le détailler, à la génération des spécifications UsiXML sur base des shapes déposées sur la feuille de travail, voici un petit schéma explicatif de leur fonctionnement, les 3 procédures dont il est question sont représentées par des ovales (Fig.4.4) :

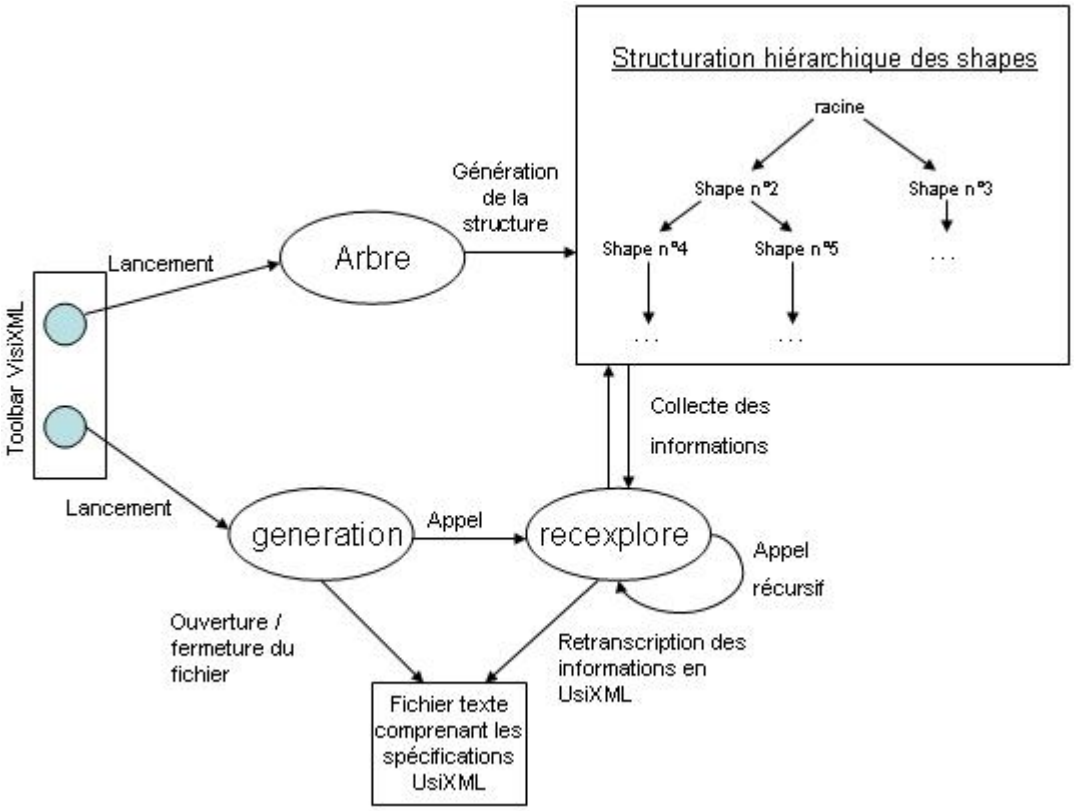


Fig.4.4 : Architecture et fonctionnement des procédures de VisiXML

Le premier bouton de la barre d'outil spécifique de VisiXML va déclencher la procédure « arbre », et celle-ci va générer une structuration hiérarchique de toutes les shapes présente sur la surface de travail sous la forme d'un Treeview. Le second bouton va déclencher la procédure « generation » qui va commencer par ouvrir le fichier d'output en écriture, avant d'appeler la procédure « recexplore » avec la clé de la shape racine comme argument. Cette procédure, sur base de la clé qui lui est fournie, va chercher les enfants de cette shape et se relancer elle-même en prenant à chaque fois comme argument la clé de l'enfant trouvé. Chaque exécution de la procédure « recexplore » génèrera une partie des spécifications en les écrivant dans le fichier d'output. A la fin de sa dernière exécution, la procédure « recexplore » a exploré tout le Treeview, et sa fermeture déclenche la fermeture du fichier d'output par la fonction generation. Les spécifications UsiXML générées sont dès lors prêtes à l'emploi.

5. Etudes de cas et évaluation

Afin d'évaluer VisiXML nous allons à présent essayer de générer des spécifications UsiXML en nous basant sur des formulaires papier, le but est de montrer que l'on peut facilement générer une interface utilisateur répondant à une demande d'information bien précise. Pour ce faire, tous les champs des formulaires papier seront représentés par l'objet interactif concret qui leur convient le mieux, et ces objets sont ensuite introduits dans VisiXML, qui générera les spécifications correspondantes.

5.1. Demande d'affiliation du Guichet des CCI

Le premier exemple choisi est une partie d'un formulaire papier de demande d'affiliation du Guichet des Chambres du Commerce et de l'Industrie, en voici un aperçu (Fig.5.1):

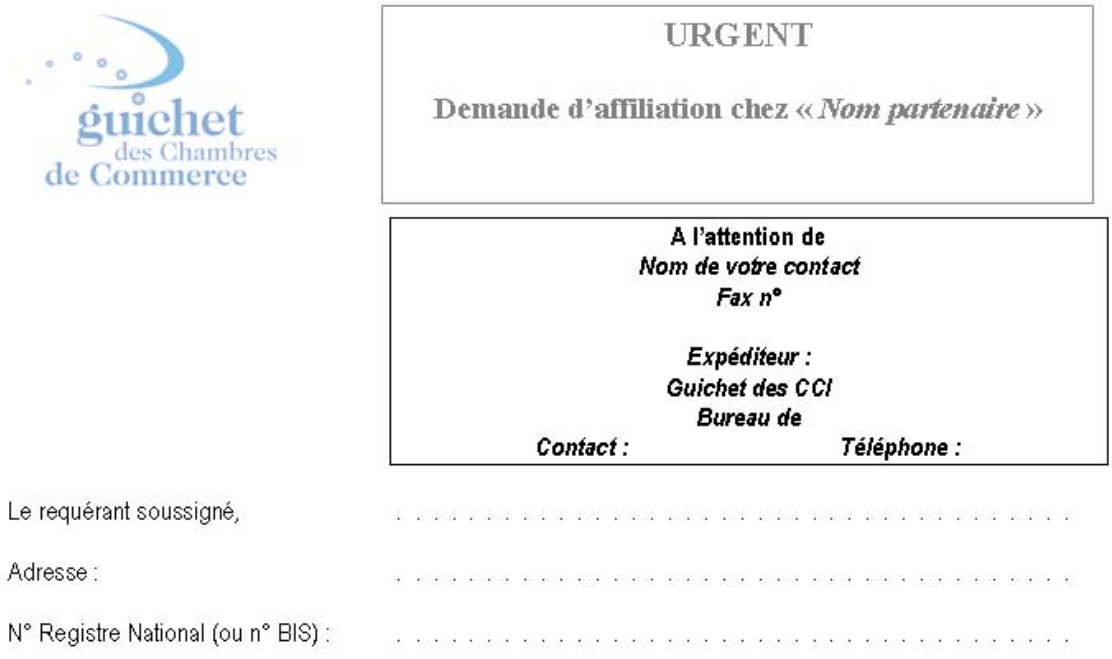


Fig.5.1 : Formulaire papier du guichet des CCI

Le premier élément qui devrait apparaître dans une interface qui pourrait remplacer ce formulaire est le logo en haut à gauche, qui correspond à un composant « imageComponent » en UsiXML. On peut également remarquer que la majorité des éléments qui composent le reste du formulaire sont facilement représentables sous VisiXML, à l'aide des objets « label » et « inputBox » qui sont tous deux des cas particuliers de l'objet de texte, le « textComponent ». Enfin il y a deux cadres avec bordure qui servent à regrouper visuellement les éléments, pour représenter ceux-ci nous avons à notre disposition le composant « box » d'UsiXML. L'ensemble de ces composants devant bien apparaître sur un support, nous aurons recours à un container de type « window » pour les y déposer.

Il nous faut donc à présent faire le « drag and drop » dans VisiXML des différents composants que nous venons d'identifier, en leur définissant les attributs voulus dans la fenêtre de custom properties. Dans le cas présent il faudra par exemple définir une hauteur et une largeur en pixels pour l'imageComponent représentant le logo. On peut voir sur l'image d'après un aperçu de cette étape de « drag and drop », lorsque tous les éléments ont été introduits dans VisiXML (Fig.5.2):

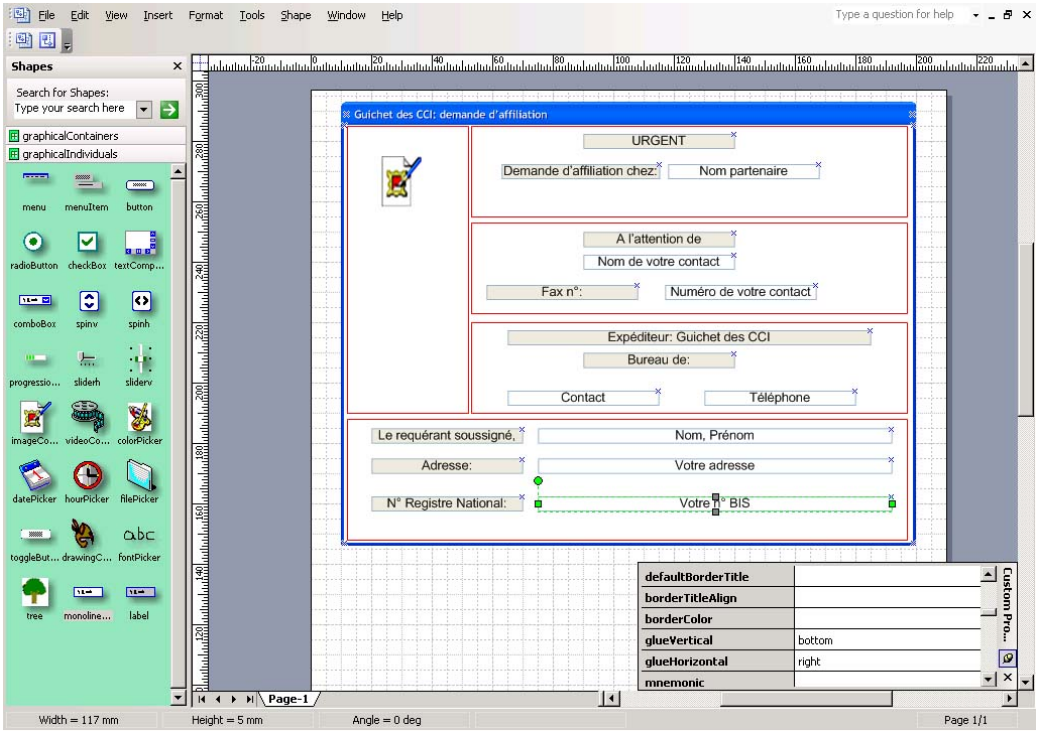


Fig.5.2 : Aperçu visuel de la partie d'interface en préparation

Une fois cette étape finie, il suffit d'appuyer successivement sur les deux boutons de la barre d'outils de VisiXML (ici en haut a gauche) qui lanceront les procédures de création de la structure hiérarchique et de génération des spécifications UsiXML proprement dites en parcourant celle-ci. Un fichier d'output est alors généré et il convient de vérifier que l'UsiXML créé est bien valide. En effet, des spécifications valides sont le gage d'un export réussi de l'interface dans les différents langages dans lesquels UsiXML est destiné à être traduit et sont donc cruciales pour garantir la grande flexibilité de l'output qui reste le principal avantage d'UsiXML (il va de soi que si les balises sont mal placées, mal refermées, mal formulées, ou toute autre erreur, la traduction a peu de chances de s'effectuer correctement).

Afin de vérifier la validité des spécifications générées on peut avoir recours à toute une série de validateurs online, pour ma part j'ai employé « DOM Validator », disponible à l'adresse suivante : http://www.w3schools.com/dom/dom_validate.asp. Ce validateur confirme bien que mon code est « valide », mais cela ne peut constituer une preuve suffisante puisque ce genre d'outils ne peut vérifier « que » la syntaxe, c'est-à-dire si les balises qui sont ouvertes au début sont bien refermées par la suite, s'il n'y a pas d'espaces parasites dans les spécifications, etc.

Dès lors, la meilleure preuve du bon fonctionnement d'un outil comme VisiXML reste la réinjection des spécifications qu'il a fournies dans un éditeur, tel que GrafiXML, ou un traducteur (interpreter), tel que FlashiXML. On pourra alors avoir un rendu visuel de l'interface générée, que nous pourrons comparer au modèle papier pour voir si l'interface correspond bien aux attentes.

Voici le résultat d'une réinjection du code généré dans FlashiXML (Fig.5.3):

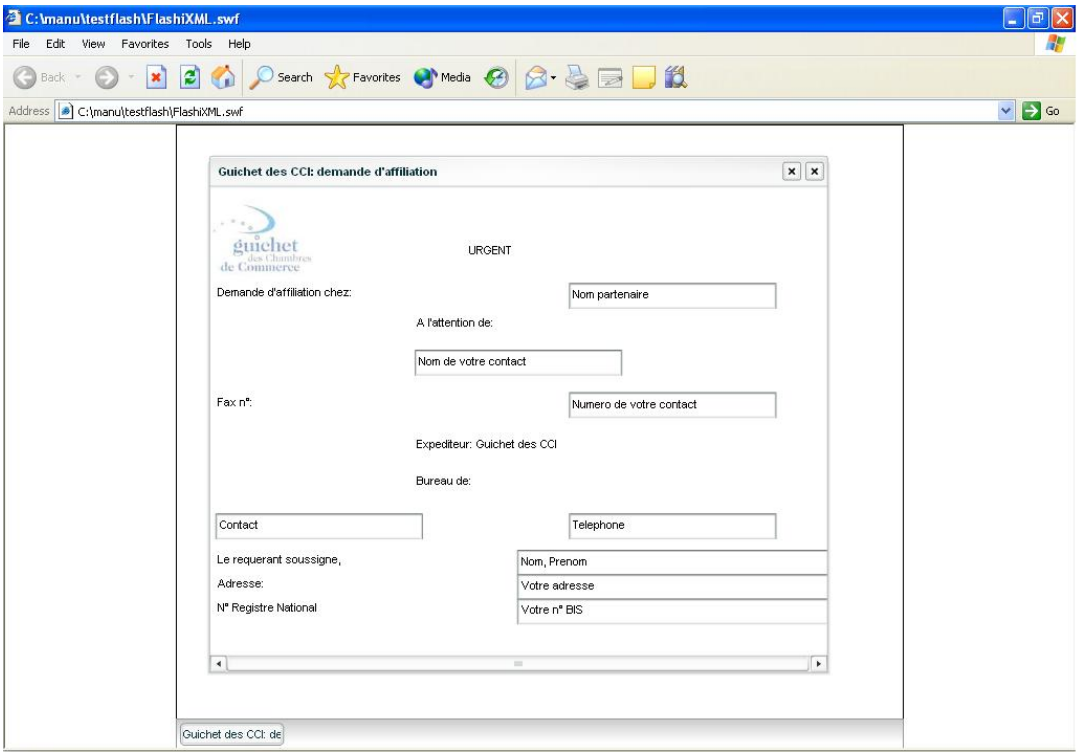


Fig.5.3 : Résultat de la génération après réinjection dans FlashiXML.

Comme on peut le constater l'agencement des différents éléments n'est pas encore parfait sur le résultat final, cela est dû à ma mauvaise connaissance actuelle du système de positionnement des objets box. On peut cependant constater que toutes les informations qui pouvaient être données sur le formulaire papier peuvent bien être saisies sur cette partie d'interface, cette dernière est donc apte à remplir la même fonction que son modèle. De plus tous les composants s'affichent correctement ce qui est la preuve supplémentaire que nous recherchions de la validité des spécifications générées.

5.2. Formulaire TVA

Le second exemple choisi est une partie d'un formulaire de renseignements complémentaires en matière de demande d'identification à la TVA, en voici un aperçu (Fig.5.4) :

5. RENSEIGNEMENTS CONCERNANT L'ACTIVITE DE L'ENTREPRISE

- Date de commencement de l'activité pour laquelle un numéro d'identification à la TVA a été demandé:
- Chiffre d'affaires annuel prévu (hors TVA) en EUR :
- Langue choisie:
  - Néerlandais
  - Français
  - Allemand
- Reprise d'un fonds de commerce:
  - Non
  - Oui

Fig.5.4 : formulaire papier de demande de renseignements complémentaires pour la TVA

A nouveau, on peut constater ici que tous les éléments constitutifs du formulaire peuvent être représentés simplement par des objets UsiXML. En plus des labels et autres textComponents standards nous aurons cette fois-ci recours à des radioButtons afin de représenter les différents choix mutuellement exclusifs de cette partie de formulaire (Fig.5.5) :

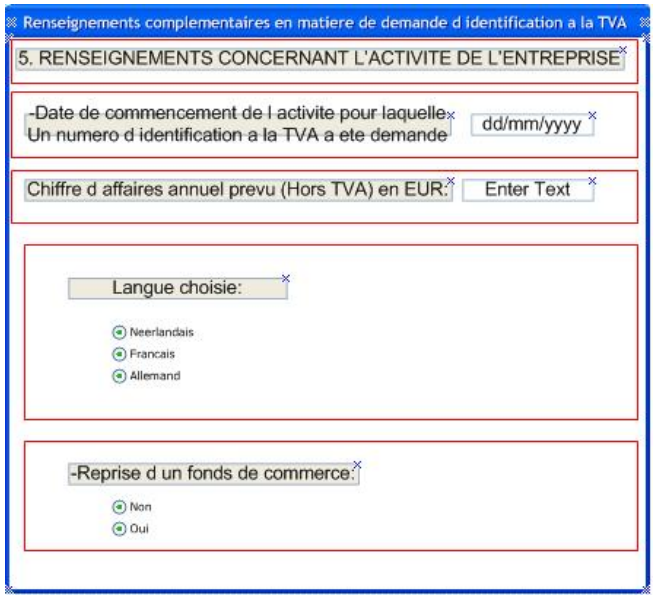


Fig.5.5 : Résultat de la schématisation de la partie d'interface sous VisiXML

Et voici à nouveau le résultat obtenu après réinjection du code généré dans FlashiXML (Fig.5.6) :

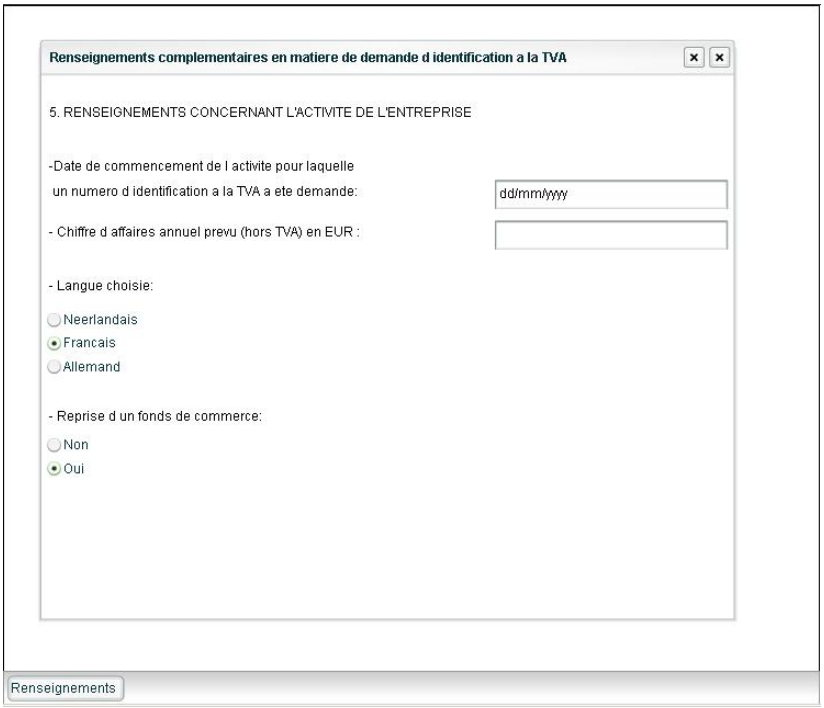


Fig.5.6 : Aperçu du formulaire de TVA sous FlashiXML

Comme on peut le constater, les spécifications générées étaient à nouveau valides, cependant, dans le cas de cet exemple-ci comme dans le cas du premier exemple, on peut noter certaines difficultés rencontrées :

- La taille des labels ne s'adapte pas automatiquement sous VisiXML lorsque le libellé dépasse une ligne. Les spécifications générées restent cependant correctes et présentables si l'on définit un attribut width de taille suffisante pour permettre au label d'afficher le texte au complet.
- Les accents ne sont pas pris en charge dans les spécifications, ou, pour être plus exact, il faudrait que l'utilisateur entre le code HTML correspondant à la lettre accentuée dans VisiXML pour pouvoir faire apparaître le bon caractère dans l'output.
- En ce qui concerne les radioButtons, il n'y a pas encore moyen de préciser directement sur la shape de VisiXML l'état du bouton, il faut pour cela rentrer ce dernier dans les custom properties, et cette solution ne permet pas d'avoir un aperçu complet de l'état des boutons en un simple coup d'œil. Ce problème devrait être réglé dans une version future de VisiXML.

- Une dernière chose, qui n'est pas vraiment une difficulté mais plutôt une contrainte consciemment imposée à l'utilisateur est le fait que la génération des spécifications sous VisiXML ne se fait de manière correcte que moyennent un ordre de drag and drop quasi hiérarchique des différentes shapes, i.e. si l'utilisateur dépose le contenant après le contenu la génération ne marchera pas.

Il ne sert cependant à rien de noircir inutilement le tableau, VisiXML a bien répondu aux attentes de création rapide d'interfaces correspondant à des spécifications bien précises, en effet la création d'un seul de ces exemples ne m'a pris que 10 minutes (20 en réalité car il a fallu que je change certaines parties du code dont les paramètres ont changé depuis la dernière version d'UsiXML sur laquelle je m'étais basé pour les spécifications générées par VisiXML, c'est le cas notamment des types de box dont les valeurs possibles sont passées de « vertical » et « horizontal » à « vert » et « horiz ») alors que cela m'aurait probablement demandé un laps de temps deux fois plus important pour le coder directement en HTML par exemple, ce qui m'aurait également empêché d'avoir la flexibilité des outputs d'UsiXML (en effet les spécifications qui ont été générées peuvent maintenant être traduites, comme cela a été expliqué dans ce mémoire). Le résultat de ces exemples est donc positif, bien que quelques détails puissent encore être améliorés.

6. Conclusion

L'accroissement de la demande actuelle de génération d'interfaces avec des spécifications qui soient les plus complètes possibles est la résultante naturelle de l'intégration de plus en plus poussée des systèmes d'information dans les organisations.

L'objectif central de ce mémoire était de fournir aux développeurs un outil de prototypage répondant au mieux à leurs attentes, attentes qui, dans ce contexte, sont principalement exprimées en termes de rapidité de développement et de niveau de fidélité élevé. Le développement des interfaces en passant par la méthode du prototypage présente, comme nous l'avons vu, des avantages indéniables, en termes de rapidité c'est évident, mais également en termes de fidélité des spécifications, pour lesquelles un niveau élevé est plus rapidement obtenu de par la flexibilité de l'output.

Bien qu'elle ne soit pas encore parfaite, une application comme VisiXML reprend déjà la majorité de ces avantages, et profitera certainement des développements futurs de toutes les applications « satellites » d'UsiXML ; en effet la standardisation des spécifications des prototypes et la multiplication des possibilités de traduction de l'output ne peuvent être que bénéfique pour les habitudes de développement futures.

Certes, il reste encore quelques ombres au tableau, certaines limites de l'environnement de développement choisi se montrant plus contraignantes qu'on eut pu le penser au départ. C'est le cas notamment des custom properties de Visio qui ne sont pas identifiables par leur dénomination sous VBA, ce qui nous oblige à les appeler groupées alors qu'une quantité non négligeable d'exceptions doit être traitée. C'est le cas également de l'accès aux données du Treeview réservé au runtime et le cas, enfin, des dimensions des shapes sous Visio qui sont exprimées en centimètres ou en inches alors qu'UsiXML les définit en pixels, ce qui forcerait, pour une détection automatique des dimensions des objets, le recours à un système de détection des paramètres d'affichage de l'utilisateur couplé à un algorithme de conversion, chose que je suis incapable de réaliser en l'état actuel de mes connaissances VBA. Cependant, VisiXML reste une application

rapide, **facile à utiliser** et qui fournit des **spécifications valides**, ne peut-on pas, dans ce cas, affirmer que les objectifs principaux sont atteints, du moins dans une certaine mesure ?

Notons finalement que, si pour l'instant VisiXML ne permet de représenter que la partie « Concrete User Interface » d'UsiXML, la partie « Abstract User Interface » représente un développement futur intéressant. La création de prototypes de très basse fidélité tels que les AUI devrait en effet se faire plus rapidement encore et ouvrir de nouvelles voies pour la traduction des spécifications dans des langages non encore supportés à l'heure actuelle.

7. Bibliographie

[Ali03] Ali, M.F., Pérez-Quiñones M.A., Abrams M.: Building Multi-Platform User Interfaces with UIML. In: Seffah, A., Javahery, H. (eds.): Multiple User Interfaces: Engineering and Application Framework. John Wiley and Sons, New York (2003)

[Caet02] Caetano, A., Goulart, N., Fonseca, M., Jorge, J., "JavaSketchIt: Issues in sketching the Look of User Interfaces", in AAAI Spring Symposium on Sketch Understanding, 2002.

[Cal03] Calvary, G., Coutaz, J., Thevenin, D., Limbourg, Q., Bouillon, L., Vanderdonckt, J.: A Unifying Reference Framework for Multi-Target User Interfaces. Interacting with Computers 15,3 (2003) 289–308

[Cam04] Campos, P.F., Nunes, N.J., "CanonSketch: a User-Centered Tool for Canonical Abstract Prototyping", in Proceedings of EHCI-DSVIS'2004, 2004.

[Ccm04] http://www.commentcamarche.net/vb/vbintro.php3

[Coye04] Coyette, A., Faulkner, S., Kolp, M., Limbourg, Q., Vanderdonckt, J., « SketchiXML: Towards a Multi-Agent Design Tool for Sketching User Interfaces Based on USIXML , in proceedings of 3rd International Workshop on TAsk MOdels and DIAgrams for user interface design - TAMODIA'2004, November 15-16 2004**.**

[Eis01] Eisenstein, J., Vanderdonckt, J., Puerta, A.: Model-Based User-Interface Development Techniques for Mobile Computing. In: Lester, J. (ed.), Proc. of 5th ACM Int. Conf. on Intelligent User Interfaces IUI'2001 (Santa Fe, January 14-17, 2001). ACM Press, New York (2001) 69–76

[Land01] Landray, J.A., Myers, B.A., "Sketching Interfaces: Toward More Human Interface Design", in IEEE Computer Society Press vol.34, pp.56-64, 2001.

[Lau01] Laudon, K.C., Laudon, J.P., "Management Information Systems – Organization and Technology in the Networked Enterprise", Prentice Hall, pp.436-512, 2001.

[Limb04a] Limbourg, Q., Vanderdonckt, J., Michotte, B., Bouillon, B.: TOMATOXML, a General Purpose XML Compliant User Interface Description Language, TOMATOXML V1.2.0. Working Paper n°105. Institut d'Administration et de Gestion (IAG), Louvain-la-Neuve, 2004.

[Limb04b] Limbourg, Q., Vanderdonckt, J.: Transformational Development of User Interfaces with Graph Transformations. In: Jacob, R., Limbourg, Q., Vanderdonckt, J. (eds.): Proc. of 5th Int. Conf. on Computer-Aided Design of User Interfaces CADUI'2004 (Madeira, January 14-16, 2004). Kluwer Academics Pub., Dordrecht (2004)

[Gree98] Greenberg, S., "Prototyping for Design and Evaluation", 1998. http://pages.cpsc.ucalgary.ca/~saul/681/1998/prototyping/survey.html

[Mari04] C. Mariage, J. Vanderdonckt, J. Eisenstein, A. Puerta, Design Knowledge for Better User Interface Design Assistants, Interacting With Computers, à paraître.

[Moli02] Molina, P.J., Meliá, S., Pastor, O., "User Interface Conceptual Patterns", in Proceedings of the 9th International Workshop on Interactive Systems. Design, Specification, and Verification, pp.159-172, 2002.

[Moli04a] Molina, P.J., Meliá, S., Pastor, O., "The Just-UI approach: Conceptual modeling of device independent user interfaces", in Revue d'Interaction Homme-Machine, vol.5 n°1, 2004.

[Moli04b] Molina, P.J., Traetteberg, H., "Analysis and Design of Model based User Interfaces: An approach to refining specifications towards implementation", In pre-proceedings of CADUI'2004, pp.211-222, 2004.

[Mori03] Mori, G., Paternò, F., Santoro, C.: Tool Support for Designing Nomadic Applications. In: Proc. of 7th ACM Int. Conf. on Intelligent User Interfaces IUI'2003 (Miami, January 12-15, 2003). ACM Press, New York (2003)141–148

[Nune04] Nunes, N.J., Campos, P., "Towards Usable Analysis, Design and Modeling Tools", in MBUI Workshop papers, 2004.

[Ols00] Olsen, D.R., Jefferies, S., Nielsen, T., Moyes, W., Fredrickson, P.: Cross Modal Interaction using XWEB. In: Proc. of the 13th Annual ACM Symposium on User Interface Software and Technology UIST'2000 (San Diego, November 5-8, 2000). ACM Press, New York (2000) 191–200

[Trae02] Traetteberg, H., "Model-based User Interface Design", Ph.D. Thesis, pp. 83 ss., 2002.

[Uqam96] http://www.comm.uqam.ca/~GRAM/C/term/tcm/tcmt126.html

[Usi04a] http://www.UsiXML.org/spec/UsiXML-cui.xsd

[Usi04b] http://www.UsiXML.org/pub/UsiXML-documentation-draft.pdf

[Vand99] Vanderdonckt, J., Berquin, P.: Towards a Very Large Model-Based Approach for User Interface Development. In: Paton, N.W., Griffiths, T. (eds.): Proc. of 1st IEEE Int. Workshop on User Interfaces to Data Intensive Systems UIDIS'99 (Edinburgh, September 5-6, 1999). IEEE Computer Society Press, Los Alamitos (1999) 76–85

[Wong02] Wong, C., Chu, H.H., Katagiri, M.A., Single-Authoring Technique for Building Device-Independent Presentations. In: Proc. of W3C Workshop on Device Independent Authoring Techniques (St. Leon-Rot, 15-26 September 2002), accessible at http://www.w3.org/2002/ 07/DIAT/posn/docomo.pdf

Annexes

ANNEXE I:

	Dreamweaver	Visual Basic	CanonSketch	JavaSketchIt	Silk
Au niveau du concepteur:					
coût de l'outil	479€ - 1129€	109$	gratuit	gratuit	gratuit
coût moyen d'un projet	faible	faible	faible	faible	faible
durée moyenne d'un projet	faible	faible	faible	faible	faible
bootstrapping	non	oui	non	oui	oui - N/A
niveau de couverture	élargi	élargi	faible (en amélioration)	faible (en amélioration)	N/A
types d'interfaces supportés	graphiques, web et multimédia	web, systèmes experts	Web	Graphiques, web	graphiques, web, systemes experts
Au niveau du développeur:					
aperçu en temps réel	oui	oui	Oui	oui	oui
niveau de représentation	haute fidélité	haute fidélité	basse, moyenne et haute fidélité	moyenne fidélité	basse fidélité
niveau d'intervention de l'outil	haute fidélité	haute fidélité	basse, moyenne et haute fidélité	moyenne fidélité	basse fidélité
facilité d'utilisation	facile	facile	Difficile	facile	facile
Déboguage	oui	non	N/A	non	non
contrôlabilité	non	non	Non	non	non
Au niveau de l'utilisateur final:					
flexibilité de l'output	faible	faible	Forte	faible	forte
représentativité	forte	forte	Forte	faible (en amélioration)	forte

ANNEXE II:

```
Option Explicit

'=====
'variables pour document_documentCreated
'=====
Dim cBar As Office.CommandBar
Dim cbButton As Office.CommandBarButton

'=====
'variables pour Arbre
'=====
Dim strGlobal() As String
Dim strActif() As String
Dim i, j As Integer
Dim shpActif As Visio.Shape
Dim strNodeKey As String
Dim strRootKey As String
Dim Noeud As Node
Dim strGrand As String
Dim strKeyParent As String

'=====
'variables pour Generation
'=====
Dim CrochOpen As String
Dim CrochClose As String
Dim Espace As String
Dim Attrib As String
Dim Guil As String
Dim Slash As String
Dim longLenKey As Long
Dim strShapeKey As String
Dim shpEnCours As Visio.Shape
Dim nrows As Long
Dim celObj As Visio.Cell
Dim strValName As String
Dim strLabelName As String
Dim intShapeKey As Integer

Private Sub Document_BeforeDocumentClose(ByVal doc As IVDocument)
Application.commandBars("VisiXML").Delete
End Sub

Private Sub document_documentCreated(ByVal doc As IVDocument)
```

```
    Application.ActiveWindow.Windows.ItemFromID(visWinIDCustProp).Visible = True
    Set cBar = Application.commandBars.Add(Name:="VisiXML")
    cBar.Context = Str(visUIObjSetDrawing) & "*"
    If cBar.Enabled = True Then
       cBar.Visible = True
    End If

    Set cbButton = cBar.Controls.Add(Type:=msoControlButton)
    With cbButton
       .Caption = "Tree"
       .TooltipText = "Click here to generate the hierarchical structure of the shapes."
       .Tag = "cbbVBaMacro"
       .FaceID = 7072
       .OnAction = "ThisDocument.Arbre"
    End With
    Set cbButton = cBar.Controls.Add(Type:=msoControlButton)
    With cbButton
       .Caption = "usiXML"
       .TooltipText = "Click here to generate usiXML specifications. A hierarchical
structure must have been generated previously."
       .Tag = "cbbVBaMacro"
       .FaceID = 7076
       .OnAction = "ThisDocument.Generation"
    End With


    Application.commandBars("Picture").Visible = False
    Application.commandBars("Clipboard").Visible = False
    Application.commandBars("Envelope").Visible = False
    Application.commandBars("Developer").Visible = False
    Application.commandBars("Formatting").Visible = False
    Application.commandBars("Standard").Visible = False
    Application.commandBars("Web").Visible = False
    Application.commandBars("View").Visible = False
    Application.commandBars("Stencil").Visible = False
    Application.commandBars("Reviewing").Visible = False
    Application.commandBars("Task Pane").Visible = False
    Application.commandBars("Snap & Glue").Visible = False
    Application.commandBars("Picture").Visible = False
    Application.commandBars("Layout & Routing").Visible = False
    Application.commandBars("Ink").Visible = False
    Application.commandBars("Format Text").Visible = False
    Application.commandBars("Format Shape").Visible = False
    Application.commandBars("Drawing").Visible = False
    Application.commandBars("Action").Visible = False
```

```
Set cBar = Application.commandBars("VisiXML")
If cBar.Position <> msoBarTop Then
cBar.Position = msoBarTop
End If

End Sub

Sub recExplore(N As Node, intProf As Integer)

CrochOpen = Chr(60)
CrochClose = Chr(62)
Espace = Chr(32)
Attrib = Chr(61) & Chr(34)
Guil = Chr(34)
Slash = Chr(47)

intProf = intProf + 1
Dim nC As Node
Dim Parent As Visio.Shape
Dim contNoChildren As Visio.Shape
Dim longLenKey2 As Long
Dim strShapeKey2 As String
Dim intShapeKey2 As Integer
Dim strID As String

longLenKey = Len(N.Key) - 1
strShapeKey = Right(N.Key, longLenKey)
intShapeKey = strShapeKey
Set shpEnCours = Visio.ActivePage.Shapes(intShapeKey)
nrows = shpEnCours.RowCount(Visio.visSectionProp)

Print #2, CrochOpen; shpEnCours.Data1; Espace;

strID = shpEnCours.Index
strID = Trim(strID)

Print #2, "ID"; Attrib; strID; Guil;

Select Case shpEnCours.Data1

Case "textComponent", "comboBox"

   Print #2, Espace; "content"; Attrib; shpEnCours.Characters; Guil;

Case "window", "dialogBox"
```

```
    Print #2, Espace; "name"; Attrib; shpEnCours.Characters; Guil;

End Select

For j = 0 To nrows - 1

    Set celObj = shpEnCours.CellsSRC(Visio.visSectionProp, j, 0)
    strValName = celObj.ResultStr(Visio.visNone)
    If strValName = "TRUE" Then
    strValName = "true"
    End If
    If strValName = "FALSE" Then
    strValName = "false"
    End If
    Set celObj = shpEnCours.CellsSRC(Visio.visSectionProp, j, 2)
    strLabelName = celObj.ResultStr(Visio.visNone)

    If strValName <> "" Then
       If strValName <> "0,0000" Then
          If Right(strValName, 5) = ",0000" Then
             strValName = Left(strValName, Len(strValName) - 5)
          End If
          Print #2, Espace; strLabelName; Attrib; strValName; Guil;
       End If
    End If

Next j



Select Case shpEnCours.Data1

Case "menu", "menuItem", "button", "radioButton", "checkBox", "textComponent",
"comboBox", "spin", "progressionBar", "slider", "imageComponent",
"videoComponent", "colorPicker", "datePicker", "hourPicker", "filePicker",
"toggleButton", "drawingCanvas", "fontPicker", "tree"

    Print #2, Slash;

End Select


Print #2, CrochClose

If N.Children Then
```

```
   Set nC = N.Child
   Do
   recExplore nC, intProf
   If nC.Index = N.Child.LastSibling.Index Then
      longLenKey2 = Len(N.Key) - 1
      strShapeKey2 = Right(N.Key, longLenKey2)
      intShapeKey2 = strShapeKey2
      Set Parent = Visio.ActivePage.Shapes(intShapeKey2)
      Print #2, CrochOpen; Slash; Parent.Data1; CrochClose
   End If
   If nC.Index = N.Child.LastSibling.Index Then Exit Do
   Set nC = nC.Next
   Loop



Else

   longLenKey2 = Len(N.Key) - 1
   strShapeKey2 = Right(N.Key, longLenKey2)
   intShapeKey2 = strShapeKey2
   Set contNoChildren = Visio.ActivePage.Shapes(intShapeKey2)

   Select Case contNoChildren.Data1

   Case "tabbedDialogBox", "tabbedItem", "window", "box", "dialogBox", "cell", "table"

      Print #2, CrochOpen; Slash; contNoChildren.Data1; CrochClose

   End Select

End If

intProf = intProf - 1
End Sub

Public Sub Arbre()

UserForm1.TreeView1.Nodes.Clear
ReDim Preserve strGlobal(1 To 7, 1 To Visio.ActivePage.Shapes.Count)

For i = 1 To Visio.ActivePage.Shapes.Count

   Set shpActif = Visio.ActivePage.Shapes(i)

   Select Case shpActif.Data2
```

```
    Case "x"

    'MsgBox ("x")

    strGlobal(1, i) = shpActif.Index
    strGlobal(2, i) = shpActif.ID
    strGlobal(3, i) = shpActif.Data1
    strGlobal(4, i) = shpActif.Cells("pinx").Result(visMillimeters)
'abscisse du coin supérieur gauche
    strGlobal(5, i) = shpActif.Cells("piny").Result(visMillimeters) +
shpActif.Cells("height").Result(visMillimeters) / 2  'ordonnée du coin supérieur gauche
    strGlobal(6, i) = shpActif.Cells("pinx").Result(visMillimeters) +
shpActif.Cells("width").Result(visMillimeters)       'abscisse du coin inférieur droit
    strGlobal(7, i) = shpActif.Cells("piny").Result(visMillimeters) -
shpActif.Cells("height").Result(visMillimeters) / 2  'ordonnée du coin inférieur droit


    Case Else

    If shpActif.Name = "sliderv" Then

            strGlobal(1, i) = shpActif.Index
            strGlobal(2, i) = shpActif.ID
            strGlobal(3, i) = shpActif.Data1
            strGlobal(4, i) = shpActif.Cells("pinx").Result(visMillimeters) -
shpActif.Cells("height").Result(visMillimeters) / 2     'abscisse du coin supérieur gauche
            strGlobal(5, i) = shpActif.Cells("piny").Result(visMillimeters) +
shpActif.Cells("width").Result(visMillimeters) / 2  'ordonnée du coin supérieur gauche
            strGlobal(6, i) = shpActif.Cells("pinx").Result(visMillimeters) +
shpActif.Cells("height").Result(visMillimeters) / 2     'abscisse du coin inférieur droit
            strGlobal(7, i) = shpActif.Cells("piny").Result(visMillimeters) -
shpActif.Cells("width").Result(visMillimeters) / 2  'ordonnée du coin inférieur droit

    Else

    strGlobal(1, i) = shpActif.Index
    strGlobal(2, i) = shpActif.ID
    strGlobal(3, i) = shpActif.Data1
    strGlobal(4, i) = shpActif.Cells("pinx").Result(visMillimeters) -
shpActif.Cells("width").Result(visMillimeters) / 2     'abscisse du coin supérieur gauche
    strGlobal(5, i) = shpActif.Cells("piny").Result(visMillimeters) +
shpActif.Cells("height").Result(visMillimeters) / 2  'ordonnée du coin supérieur gauche
    strGlobal(6, i) = shpActif.Cells("pinx").Result(visMillimeters) +
shpActif.Cells("width").Result(visMillimeters) / 2     'abscisse du coin inférieur droit
```

```
   strGlobal(7, i) = shpActif.Cells("piny").Result(visMillimeters) -
shpActif.Cells("height").Result(visMillimeters) / 2  'ordonnée du coin inférieur droit

End If

End Select

Next i

For j = 1 To Visio.ActivePage.Shapes.Count

   Set shpActif = Visio.ActivePage.Shapes(j)
   ReDim strActif(1 To 7, 1 To 1)

   Select Case shpActif.Data2

   Case "x"

   'MsgBox ("x")

   strActif(1, 1) = shpActif.Index
   strActif(2, 1) = shpActif.ID
   strActif(3, 1) = shpActif.Data1
   strActif(4, 1) = shpActif.Cells("pinx").Result(visMillimeters)
'abscisse du coin supérieur gauche
   strActif(5, 1) = shpActif.Cells("piny").Result(visMillimeters) +
shpActif.Cells("height").Result(visMillimeters) / 2  'ordonnée du coin supérieur gauche
   strActif(6, 1) = shpActif.Cells("pinx").Result(visMillimeters) +
shpActif.Cells("width").Result(visMillimeters)       'abscisse du coin inférieur droit
   strActif(7, 1) = shpActif.Cells("piny").Result(visMillimeters) -
shpActif.Cells("height").Result(visMillimeters) / 2  'ordonnée du coin inférieur droit


   Case Else

   If shpActif.Name = "sliderv" Then

         strActif(1, 1) = shpActif.Index
         strActif(2, 1) = shpActif.ID
         strActif(3, 1) = shpActif.Data1
         strActif(4, 1) = shpActif.Cells("pinx").Result(visMillimeters) -
shpActif.Cells("height").Result(visMillimeters) / 2     'abscisse du coin supérieur gauche
         strActif(5, 1) = shpActif.Cells("piny").Result(visMillimeters) +
shpActif.Cells("width").Result(visMillimeters) / 2  'ordonnée du coin supérieur gauche
         strActif(6, 1) = shpActif.Cells("pinx").Result(visMillimeters) +
shpActif.Cells("height").Result(visMillimeters) / 2     'abscisse du coin inférieur droit
```

```
        strActif(7, 1) = shpActif.Cells("piny").Result(visMillimeters) -
shpActif.Cells("width").Result(visMillimeters) / 2  'ordonnée du coin inférieur droit

    Else



    strActif(1, 1) = shpActif.Index
    strActif(2, 1) = shpActif.ID
    strActif(3, 1) = shpActif.Data1
    strActif(4, 1) = shpActif.Cells("pinx").Result(visMillimeters) -
shpActif.Cells("width").Result(visMillimeters) / 2     'abscisse du coin supérieur gauche
    strActif(5, 1) = shpActif.Cells("piny").Result(visMillimeters) +
shpActif.Cells("height").Result(visMillimeters) / 2  'ordonnée du coin supérieur gauche
    strActif(6, 1) = shpActif.Cells("pinx").Result(visMillimeters) +
shpActif.Cells("width").Result(visMillimeters) / 2     'abscisse du coin inférieur droit
    strActif(7, 1) = shpActif.Cells("piny").Result(visMillimeters) -
shpActif.Cells("height").Result(visMillimeters) / 2  'ordonnée du coin inférieur droit

End If

End Select

    If j = 1 Then

        '=====
        'Si c'est la première shape de la page, elle est identifiée et indexée comme racine de
l'arbre.
        '=====

        strNodeKey = "k" & shpActif.Index
        strRootKey = strNodeKey
        Set Noeud = UserForm1.TreeView1.Nodes.Add(, , strNodeKey, shpActif.Data1)
        Noeud.EnsureVisible

    Else

        strNodeKey = "k" & shpActif.Index
        strGrand = 99999
        For i = 1 To Visio.ActivePage.Shapes.Count
            If strActif(4, 1) - strGlobal(4, i) >= 0 Then
                If strActif(5, 1) - strGlobal(5, i) <= 0 Then
                    If strActif(6, 1) - strGlobal(6, i) <= 0 Then
                        If strActif(7, 1) - strGlobal(7, i) >= 0 Then
                            If strActif(1, 1) <> strGlobal(1, i) Then
```

```
                    If strActif(4, 1) - strGlobal(4, i) - strActif(5, 1) + strGlobal(5, i) -
strActif(6, 1) + strGlobal(6, i) + strActif(7, 1) - strGlobal(7, i) <= strGrand Then
                    strGrand = strActif(4, 1) - strGlobal(4, i) - strActif(5, 1) +
strGlobal(5, i) - strActif(6, 1) + strGlobal(6, i) + strActif(7, 1) - strGlobal(7, i)
                    strKeyParent = "k" & strGlobal(1, i)
                    Else
                    End If
                  Else
                  End If
                Else
                End If
              Else
              End If
            Else
            End If
          Else
          End If
      Next i

   Set Noeud = UserForm1.TreeView1.Nodes.Add(strKeyParent, tvwChild, strNodeKey,
shpActif.Data1)
   Noeud.EnsureVisible

   End If
Next j

MsgBox ("Hierarchical Treeview has been succesfully generated, please click the
generation button to generate the specifications.")

End Sub

Public Sub Generation()

Open "output.txt" For Output Shared As #2
Print #2, "<?xml version=""1.0"" encoding=""UTF-8"" standalone=""yes""?>"
Print #2, "<uiModel creationDate=""2004-07-16T13:39:23.366+02:00""
schemaVersion=""1.6.0"" id=""test_16"" name=""test"" xsi:schemaLocation="""";
Print #2, "http://www.usixml.org/spec http://www.usixml.org/spec/usiXML-
ui_model.xsd";
Print #2, "http://www.usixml.org/spec http://www.usixml.org/spec/usiXML-aui.xsd
http://www.usixml.org/spec http://www.usixml.org/spec/usiXML-context.xsd
http://www.usixml.org/spec http://www.usixml.org/spec/usiXML-cui.xsd
http://www.usixml.org/spec http://www.usixml.org/spec/usiXML-cui_auditory.xsd
http://www.usixml.org/spec http://www.usixml.org/spec/usiXML-cui_behavior.xsd
http://www.usixml.org/spec http://www.usixml.org/spec/usiXML-cui_graphical.xsd
http://www.usixml.org/spec http://www.usixml.org/spec/usiXML-cui_relationship.xsd
```

```
http://www.usixml.org/spec http://www.usixml.org/spec/usiXML-domain.xsd
http://www.usixml.org/spec http://www.usixml.org/spec/usiXML-mapping.xsd
http://www.usixml.org/spec http://www.usixml.org/spec/usiXML-resource.xsd
http://www.usixml.org/spec http://www.usixml.org/spec/usiXML-rule_term.xsd
http://www.usixml.org/spec http://www.usixml.org/spec/usiXML-task.xsd
http://www.usixml.org/spec http://www.usixml.org/spec/usi";
Print #2, "XML-transformation.xsd""
xmlns:xsi=""http://www.w3.org/2001/XMLSchema-instance""
xmlns=""http://www.usixml.org"">"
Print #2, "<version modifDate=""2004-07-16T13:39:56.198+02:00""
xmlns="""">1</version>"
Print #2, "<cuiModel id=""test_16-cui"" name=""test-cui"">"

recExplore UserForm1.TreeView1.Nodes(strRootKey), 0

Print #2, "</cuiModel>"
Print #2, "</uiModel>"

Close #2

MsgBox ("USIXML specifications have been succesfully generated. You will find them
in the VisiXML folder.")

End Sub
```

ANNEXE IV:

Elément	Support de spécification	Support visuel
box	total	complet
table	total	simulation
cell	total	simulation
dialogBox	total	complet
window	total	simplification
tabbedDialogBox	total	complet
tabbedItem	total	simplification
textComponent	total	simplification
imageComponent	total	simulation
videoComponent	total	simulation
button	total	complet
checkBox	total	complet
radioButton	total	complet
toggleButton	total	complet
tree	total	simulation
spin	total	complet
slider	total	simplification
comboBox	total	complet
menu	total	complet
menuItem	total	complet
drawingCanvas	total	simulation
colorPicker	total	simulation
fontPicker	total	simulation
datePicker	total	simulation
filePicker	total	simulation
hourPicker	total	simulation
progressionBar	total	simulation