Université Catholique de Louvain
Ecole Polytechnique de Louvain
Département d'ingénierie informatique

Développement d'un simulateur d'interface graphique à distance

Promoteur : Jean Vanderdonckt

Mémoire présenté par Michaël Delhaye

En vue de l'obtention du grade de master en informatique

Louvain-La-Neuve
Année académique 2010-2011

Table des matières

Introduction

Contexte du problème

Lors de la conception d'une interface graphique, on passe généralement par une phase de prototypage. Le prototype ne doit pas nécessairement être réalisé sous forme de code exécutable mais peut se concevoir sous forme descriptive dans un langage tel qu'usiXML. En effet usiXML permet dans sa dernière section « concrete interface » de décrire une interface concrète indépendamment de tout langage de programmation et de plateforme.

Le problème est que le prototype ainsi développé reste lié à la station sur laquelle il a grandi. On voudrait pouvoir en faire la démonstration aux futurs utilisateurs afin de récolter leurs impressions sans les obliger à se déplacer ou à faire des installations complexes.

On pense alors à Internet. Mais les navigateurs sont incapables de représenter les interfaces décrites en usiXML sous forme graphique. Tout au plus ils peuvent afficher le code XML, parfois de manière structurée ce qui est bien sûr insuffisant.

L'idée est donc de développer un plugin pour un navigateur Internet permettant de lire ces spécifications à distance et d'effectuer leur rendu. On pense par exemple au format graphique SVG également en XML et pris en charge nativement par les navigateurs modernes.

Mais le développement d'un plugin n'est pas chose aisée car il doit être développé en langage C et ne permet pas d'utiliser les moteurs de rendu des navigateurs. Peu de bibliothèques existantes peuvent être réutilisées et donc beaucoup de choses doivent être recommencées à zéro. De plus un plugin est lié à un navigateur et à une plateforme donc il faudrait développer des variantes pour chaque combinaison possible de ceux-ci. Enfin cela impose une installation côté client et donc des mises à jour ce qui peut passer à côté de l'objectif initial qui est de rendre la démonstration très facile. Tout de même il faut reconnaître cette approche serait optimale d'un point de vue de l'intégration au navigateur.

Les extensions de navigateur sont elles programmées en JavaScript et peuvent utiliser les fonctionnalités du navigateur. Mais par ce biais là on ne peut pas

prendre en charge de nouveaux formats de données tels qu'usiXML. La solution n'est donc pas à chercher de ce côté là non plus.

Motivations

Afin de pouvoir aller assez loin dans la démarche, le choix sera fait d'effectuer un prétraitement de la description usiXML côté serveur ce qui permettra de pouvoir livrer du code HTML, CSS et JavaScript au navigateur qui pourra être interprété directement. De cette manière on utilise les nombreuses possibilités du navigateur sans réinventer la roue.

Par un simple téléchargement du fichier XML sur un serveur prévu à cet effet on pourra ainsi permettre à toute personne de consulter l'interface développée et ce via un navigateur Internet sans aucune installation.

Le développeur d'une application pourra alors facilement soumettre des prototypes d'interfaces aux parties-prenantes et récolter leurs commentaires assez tôt dans le cycle de développement, ce qui réduira fortement les coûts.

Objectif central de la thèse

L'objet de ce mémoire est de développer un simulateur permettant d'afficher des interfaces décrites à l'aide d'usiXML dans un navigateur Internet. Une analyse des patterns d'architecture logicielle pour applications graphiques sera effectuée et on tentera d'en utiliser les enseignements afin de développer ce simulateur.

Le but est d'obtenir un interpréteur usiXML pour le navigateur qui supporte un échantillon représentatif des possibilités d'usiXML et qui puisse être facilement étendu. On veillera aussi à maximiser la réutilisabilité du code afin de pouvoir bénéficier du travail effectué pour une application dans d'autres contextes, par exemple pour les navigateurs de smartphones.

Les outils et techniques utilisés lors du développement seront détaillés. Signalons déjà que le langage de programmation utilisé côté serveur sera le PHP car un tel environnement est courant et facile à mettre en place pour le développeur qui souhaite faire une démonstration d'interface. Malgré ces avantages on se rendra compte au long de ce travail de quelques limites d'un tel environnement.

Hypothèses de travail

Quelques hypothèses sont posées. On suppose d'abord que les fondements du langage usiXML sont connus. Pour le détail de la structure du langage on renvoie à la documentation officielle disponible sur www.usiXML.org

On suppose aussi que l'on est capable de générer des descriptions usiXML valides à l'aide de logiciels graphiques[1] ou de façon manuelle à l'aide d'un éditeur de texte. Seule la partie « concrete interface » sera considérée par l'interpréteur avec la description d'une et une seule fenêtre par fichier.

D'autre part on suppose que l'on a accès à un serveur web avec le support de PHP5. C'est sur cette plateforme que sera développée l'interpréteur.

Enfin l'interpréteur sera en théorie compatible avec tous les navigateurs web récents mais on fera le choix de ne le tester que sur Google Chrome. Il ne s'agit pas du navigateur ayant le plus de parts de marché mais il fait partie de ceux prenant le mieux en charge les derniers standards. Il est disponible sur toutes les plateformes et est subjectivement le plus apprécié dans la communauté informatique. Afin d'offrir une expérience similaire sur tous les navigateurs, de légères adaptations pourraient devoir être effectuées.

Plan de lecture

Ce mémoire est divisé en 2 parties. La première fait une analyse théorique des modèles d'architecture pour logiciels avec interface graphique. Après une analyse du poste de travail où on verra que les applications web sont une tendance de l'industrie on s'intéressera plus particulièrement à celles-ci, d'autant plus que c'est ce qui nous concerne pour l'implémentation de l'interpréteur. On y traite enfin de quelques rappels technologiques concernant le web.

En deuxième partie on trouve les détails de l'implémentation se basant sur les fondements posés dans la première partie. L'architecture y est longuement décrite et on y donne quelques explications quant à la conversion en essayant de ne pas trop rentrer dans les détails du code. Les lecteurs avertis peuvent passer à cette section directement.

---

[1] Il en existe plusieurs comme GrafiXML, VisiXML, SketchiXML,…

Partie 1 : Analyse théorique

1 Patterns d'architecture logicielle pour applications graphiques

Commençons par une étude théorique des patterns d'architecture logicielle adaptés aux applications graphiques. Nous pourrons l'utiliser par la suite comme cadre d'analyse afin de mieux appréhender l'univers du web. Les enseignements nous seront aussi utiles afin de développer l'interpréteur dans la deuxième partie. Mais tout d'abord qu'est-ce qu'un pattern d'architecture logicielle ? Voici donc tout d'abord 2 définitions.

Definition : **An architectural style is a named collection of architectural design decisions that (1) are applicable in a given development context, (2) constrain architectural design decisions that are specific to a particular system within that context, and (3) elicit beneficial qualities in each resulting system.**[2]

Definition : **An architectural pattern is a named collection of architectural design decisions that are applicable to a recurring design problem, parameterized to account for different software developemnt contexts in which that problem appears.**

On voit qu'un style d'architecture intervient à un niveau plus global et conditionne l'implémentation à un plus bas niveau. Un pattern est une solution générique plus locale à un problème et qui peut être appliqué à des problèmes similaires.

Des styles d'architecture sont par exemple : client-serveur, muli-tier, basé sur les évènements, orienté objet, pipe&filter,…

Des patterns d'architecture sont : Modèle-Vue-Contrôleur (MVC), Modèle-Vue-Présentateur (MVP), Présentation-Abstraction-Contrôle (PAC), le pattern Observateur,…

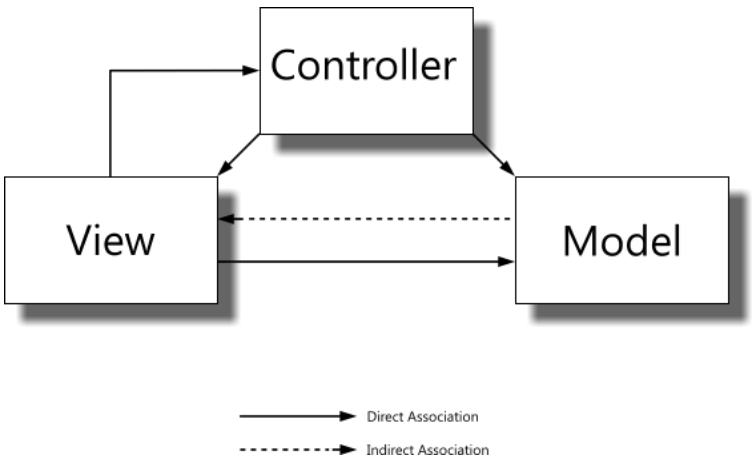
[2] *Software architecture : foundations, theory, and practice*, page 73

Nous allons nous intéresser aux patterns d'architecture logicielle les plus courants et plus particulièrement ceux adaptés aux applications graphiques : MVC et MVP dans sa version initiale et simplifiée.

1.1 Modèle-Vue-Contrôleur

MVC est un des plus anciens patterns. Il a été inventé dans les années 70 à un moment où les interfaces graphiques n'étaient pas encore très courantes. Il s'agit d'un des premiers efforts pour rationnaliser le développement d'applications graphiques. Cette méthode a été implémentée pour la première fois pour le langage de programmation Smalltalk. On parle encore beaucoup de cette méthode aujourd'hui mais la version originale est très souvent mal comprise à cause de ses évolutions

Comme l'acronyme le laisse penser, l'objectif est de séparer les données du monde réel (modèle) gérées par l'application de l'interface graphique (vue) qui les représente. Ce faisant on gagne en clarté et en maintenabilité. Mais on permet aussi implicitement d'avoir plusieurs interfaces qui représentent les mêmes données, même simultanément. On maximise aussi ainsi la réutilisabilité du code de la vue et du modèle. Seul le contrôleur reste très spécifique à l'application.



Pattern MVC

Le modèle est indépendant de la vue et se suffit à lui-même pour représenter le monde réel. Il comprend l'état, la structure et le comportement de ce qu'on veut modéliser. Le modèle n'accède pas directement à la vue ni au contrôleur. Si le modèle est modifié la vue ou le contrôleur peut en être informé grâce au pattern « Observer ». Les objets qui observent le modèle sont notifiés lorsque des modifications y sont apportées.

Le rôle du contrôleur est de gérer les actions de l'utilisateur effectuées sur l'interface graphique à l'aide du clavier, de la souris ou de tout dispositif d'interaction utilisateur[3] et de les convertir en modifications à apporter au modèle ou à la vue. Il faut savoir qu'il existe en réalité un contrôleur par widget de l'interface graphique. Chaque contrôleur prend en charge les événements générés par l'élément de l'interface graphique dont il est responsable afin d'agir de façon adéquate sur le modèle ou sur la vue. Toute la logique de l'application est répartie dans les différents contrôleurs.

Le contrôleur doit être compris dans le pattern MVC comme un intermédiaire entre l'utilisateur et l'application, pas comme l'intermédiaire entre la vue et le modèle. C'est une nuance subtile mais importante si on veut se replacer dans l'esprit initial. Le contrôleur a un rôle de médiateur entre l'utilisateur et l'application mais la séparation du modèle et de la vue est réalisée par le pattern Observer.

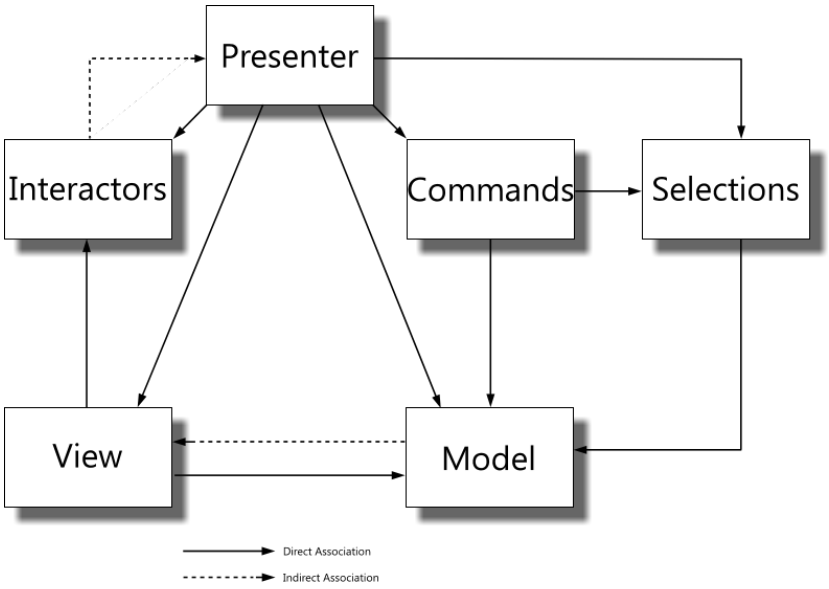
La vue, réduite à un widget d'interface graphique, possède quant à elle en plus du lien avec son contrôleur associé, un lien avec le modèle d'où elle va directement chercher les données sans passer par le contrôleur. C'est tout un ensemble de couples vue-contrôleur qui va composer l'interface graphique complète.

1.2 Modèle-Vue-Présentateur

Cette architecture vient initialement des bureaux d'IBM et a été formalisée par Mike Potel en 1996 lorsqu'il travaillait pour Taligent, une joint venture entre Apple et IBM. MVP signifie Modèle-Vue-Présentateur. C'est une adaptation de MVC beaucoup utilisée pour le développement sur Microsoft Windows (exemples ?) On voit que le contrôleur de MVC est remplacé par un présentateur. C'est bien sûr plus qu'un changement de nom. Nous allons voir cela.

L'approche consiste à se rendre compte qu'il y a dans toute application, une partie présentation et une partie gestion des données. L'ensemble de ce qui était vue et contrôleur dans le pattern MVC forme ici la partie présentation, le modèle correspond à la gestion des données. L'ensemble est alors raffiné d'avantage afin d'aider le développeur à construire des applications complexes.

[3] Ecran tactile, système de synthèse vocale,…



Pattern MVP (Taligent)

Les concepts de Sélections et de Commandes sont ajoutés entre le contrôleur et le modèle. Les sélections permettent de spécifier précisément des sous-ensembles de données du modèle. Les commandes définissent les opérations que l'on peut effectuer sur les données du modèle ou sur les sélections.

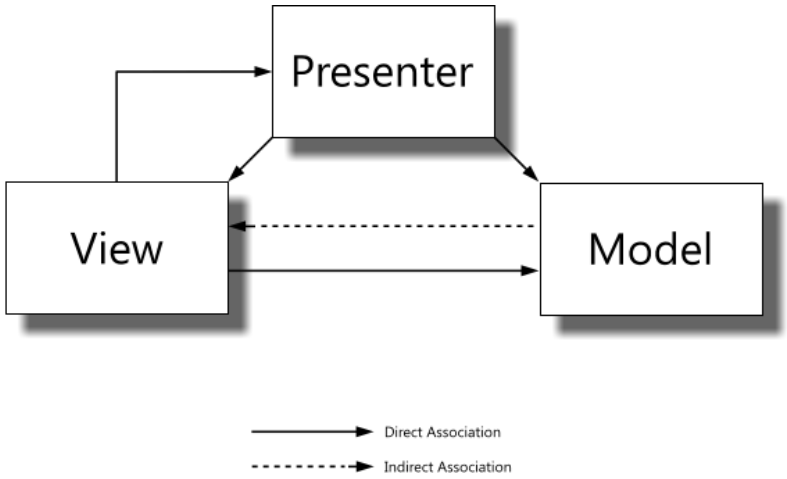
Pour répondre aux actions de l'utilisateur ou événements de l'interface graphique on ajoute au pattern le concept d'Interacteur entre la vue et le contrôleur. Ce sont eux qui vont se charger de convertir ces actions effectuées sur l'interface en modifications à effectuer sur les données.

Le contrôleur quant à lui est élevé au niveau application et devient Présentateur. Il prend en charge une vue complète voire un ensemble de vues. Il s'occupe de gérer la coordination de la vue, des interacteurs, des commandes, des sélections et du modèle. La tâche du présentateur devient la mise à jour du modèle, moins d'être l'intermédiaire entre l'utilisateur et l'application. Il est pour cela assisté des autres composants.

1.3 Modèle-Vue-Présentateur simplifié

Il existe une version simplifiée de MVP plus proche de MVC appelée le « Dolphin Smalltalk Model-View-Presenter ». Les sélections, commandes et interracteurs n'existent plus. La vue se suffit ici à elle-même pour tous les aspects liés à l'interface graphique. Et le présentateur a le rôle de gérer toute la logique applicative et donc de faire le lien entre la vue et le modèle. A la différence du contrôleur MVC il prend en charge des ensembles complets de vues et pas seulement des widgets individuels. La deuxième grande différence est qu'il n'a

pas comme objectif principal de gérer les évènements utilisateur, cette tâche étant confiée à la vue.



Pattern MVP (Dolphin)

Il existe 2 variantes en fonction du mode de transmission des données au présentateur. Pour la première appelée « vue passive » c'est le présentateur qui vient chercher les données introduites dans la vue par l'utilisateur. L'autre variante est le « supervising controller ». Celle-ci suppose que c'est la vue elle-même qui transmet les données au contrôleur-présentateur.

2 Analyse du poste de travail

Les applications traditionnelles sont des logiciels installés et exécutés directement sur les postes de travail. En entreprise elles sont maintenant rares, la plupart des applications fonctionnent désormais en mode client-serveur. Cela permet surtout de gérer des données centralisées dont les droits d'accès sont définis et soumises à des procédures de sauvegardes. On trouve d'autres avantages comme l'administration facilitée, l'augmentation de la sécurité et l'architecture rendue évolutive au nombre d'utilisateurs.

Mais aujourd'hui un nouveau mode de livraison des logiciels est très courant : SAAS pour software-as-a-service. Le logiciel reste sur le serveur du prestataire et le client y accède grâce à une interface web. La facturation se fait généralement à l'utilisateur. C'est une évolution à l'heure d'Internet du modèle ASP (Application service-provider) où un véritable logiciel natif était utilisé comme client. Les avantages sont multiples : pas d'installation côté client, pas de mises à jour à faire côté client, support facilité car tout le monde utilise la même version, accès à ses applications dans l'entreprise comme à domicile,…

Cette tendance est accompagnée par des évolutions dans le matériel. Dans les entreprises de plus en plus souvent les postes utilisateur ne sont plus que de simples clients légers[4]. Le concept n'est pas nouveau mais il semble qu'on ne puisse vraiment l'appliquer que depuis peu. En effet, de l'espace de stockage n'est souvent plus nécessaire au niveau du poste de travail puisque toutes les données se trouvent sur un serveur. Les modes de stockage amovibles ne sont eux plus proposés depuis longtemps étant donné le risque du point de vue de la sécurité qu'ils représentent[5]. Et du point de vue des ressources de calcul, les processeurs dernier-cri ne sont généralement plus nécessaires étant donné que les applications sont déportées côté serveur.

En dehors de certains domaines où les applications natives[6] restent indispensables comme la publication assistée par ordinateur et les logiciels dédiés au montage audio et vidéo, on assiste donc à une transition vers des applications web ne demandant pas d'installation. Afin d'imiter le comportement et la souplesse d'utilisation d'applications natives les applications web sont

---

[4] Ordinateurs de petit format, dont les composants sont réduits à leur strict minimum et parfois même sans disque dur mais à différencier des terminaux qui ne font que déporter l'interface d'un serveur.
[5] En terme de confidentialité, de virus,…
[6] Installées sur le poste de travail

dotées d'interfaces graphiques de plus en plus évoluées. On assiste même à une tendance un peu inverse à celle exposée précédemment en ce sens qu'on ramène de plus en plus de logique côté client, au départ uniquement pour ce qui concerne l'interface graphique mais aussi de plus en plus de logique applicative. Ces applications sont appelées Rich Internet applications. On peut citer Gmail[7] comme le précurseur de cette tendance. D'autres applications ont suivi comme des suites bureautiques en ligne[8], des applications de cartographie[9], de gestion de projets[10]. Les réseaux sociaux comme Facebook ont de plus en plus l'allure d'applications également. Mais les éditeurs d'applications professionnelles suivent cette tendance de prêt.

L'attitude des éditeurs de navigateurs confirme cette tendance. Dans la concurrence qu'ils se livrent actuellement, ils ont en effet trouvé le moyen de se démarquer dans l'amélioration des performances du langage de script utilisé côté client : JavaScript. Des machines virtuelles indépendantes ont été écrites dans le langage de bas niveau C améliorant les performances de façon très significative.

Mais des tests effectués[11] sur Firefox 3.5 à l'aide de l'outil de benchmarking SunSpider montrent que ses performances sont plus de 2 fois supérieures à celles de Firefox 3. L'éditeur Apple annonce lui le 24 février 2009 que la nouvelle version de son navigateur, Safari 4, exécute le code Javascript 4,2 fois plus vite que la version précédente. Les performances ont continué d'évoluer depuis.

Notons aussi que Google expérimente actuellement le moyen de permettre aux applications web d'exécuter du code natif, dans un environnement sécurisé. Portant le nom de « Native Client », cette technologie est intégrée dans la dernière version beta de Google Chrome. Cela ouvre la voie à des applications web à la réactivité identique à leurs homologues installés sur le poste de travail.

---

[7] Application de messagerie de Google
[8] Google Docs, Zoho Docs, Microsoft Office Live, iWork.com d'Apple (seulement consultation)
[9] Google Maps
[10] Basecamp, GoPlan, ActiveCollab
[11] Par les sites d'actualités Technologizer et PCPro

3 Analyse des modèles d'application web existantes

Après ce deuxième chapitre montrant l'importance des applications web dans un avenir proche, concentrons-nous maintenant sur celles-ci. Voyons comment elles sont organisées et quelles sont les approches possibles pour la création d'interfaces utilisateur.

Tentons d'analyser le web actuel sous l'angle des patterns d'architecture logicielle vus précédemment. On verra que le style client-serveur conditionne les possibilités d'implémentation. Commençons par le web classique, les pages HTML statiques, pour aller jusqu'aux applications AJAX dynamiques voire totalement autonomes.

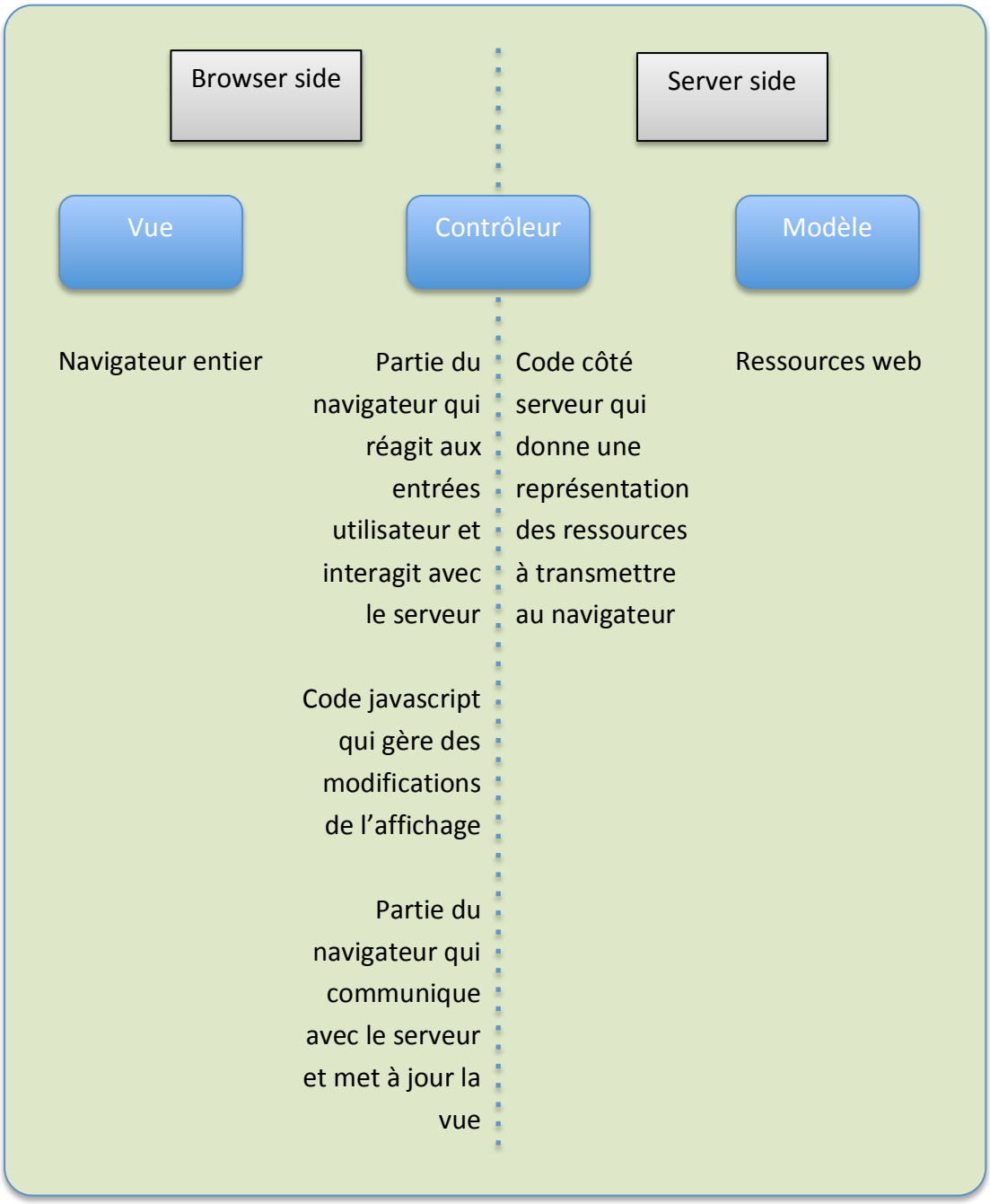
3.1 Le Web comme application MVC

Le web peut être interprété comme une application du pattern MVC. Posons-nous d'abord la question des éléments qui composent le contrôleur. Qui répond aux entrées utilisateur ? On voit qu'il y a le navigateur Internet lui-même. C'est lui qui prend en charge l'entrée d'URL dans la barre d'adresse, les boutons précédent/suivant, l'ouverture d'onglets, le défilement vertical et horizontal, le clic sur les liens et boutons, l'impression,… Notons aussi que l'éventuel code JavaScript des pages visitées qui ajoute des comportements aux pages ou modifie des comportements par défaut fait aussi partie du contrôleur.

La vue est alors ce qui reste du navigateur à savoir toute l'interface graphique ainsi que le contenu des pages visitées.

Le modèle quant à lui consiste en les différentes ressources web disponibles sur le serveur. Une ressource est tout élément pouvant être identifié à l'aide d'une adresse URL[12]. Cela peut être des pages statiques/dynamiques, des images, des feuilles de style CSS, des fichiers JavaScript, des documents de traitement de texte,…

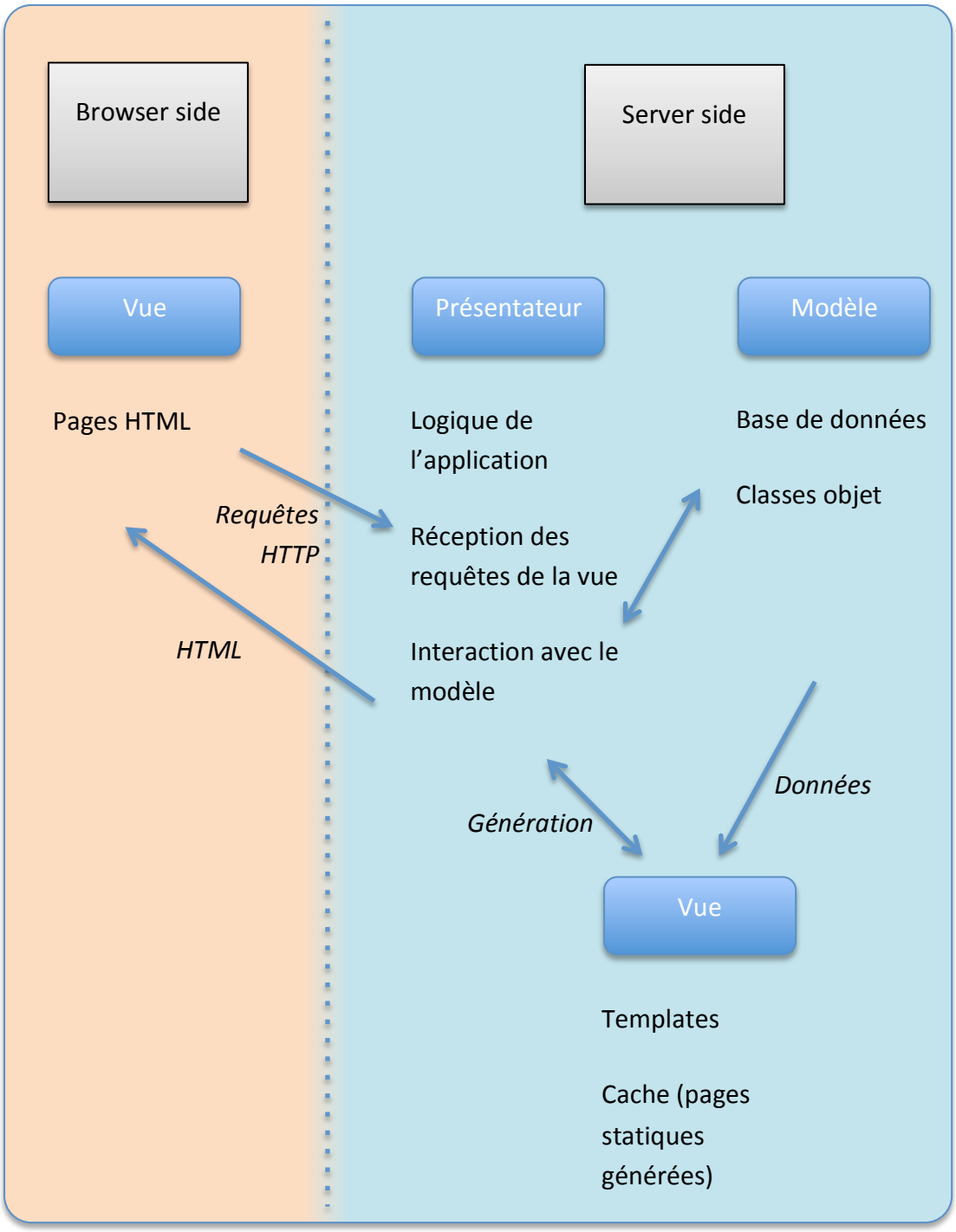
[12] Universal Ressource Locator



Modèle-Vue-Contrôleur

Il y a encore une partie du contrôleur située côté serveur qui est chargée de donner une représentation des ressources à transmettre au navigateur et une partie côté navigateur qui se charge de communiquer avec le serveur et de mettre à jour la vue.

Sur le web il n'y a pas d'état[13] côté serveur (mis à part mécanisme de session). Tout objet est recréé à chaque invocation. On a donc pas de pattern observer permettant de mettre à jour la vue automatiquement lors d'une modification de données côté serveur. Bien sûr on peut arriver à imiter ce fonctionnement en mettant en place des rafraîchissements automatiques côté navigateur.



Application web simple

[13] conservation de données temporaires, pointeur d'instruction

3.2 Le Web comme application du pattern MVP simplifié

Si on fait abstraction du navigateur lui-même et qu'on ne regarde que ce qui se passe au niveau du contenu on peut voir le web comme une application MVP. En effet ce ne sont plus alors les événements d'entrée/sortie utilisateur qui sont déterminant mais les requêtes HTTP effectuées. On voit alors que le contrôleur se trouve tout entier côté serveur. C'est là que sont traitées les requêtes issues du navigateur. Il a maintenant plutôt le rôle de présentateur car il ne traite plus des actions utilisateur mais des événements de la vue.

La vue consiste en les pages HTML et les fichiers de style CSS. Ils sont récupérés tels quels du serveur.

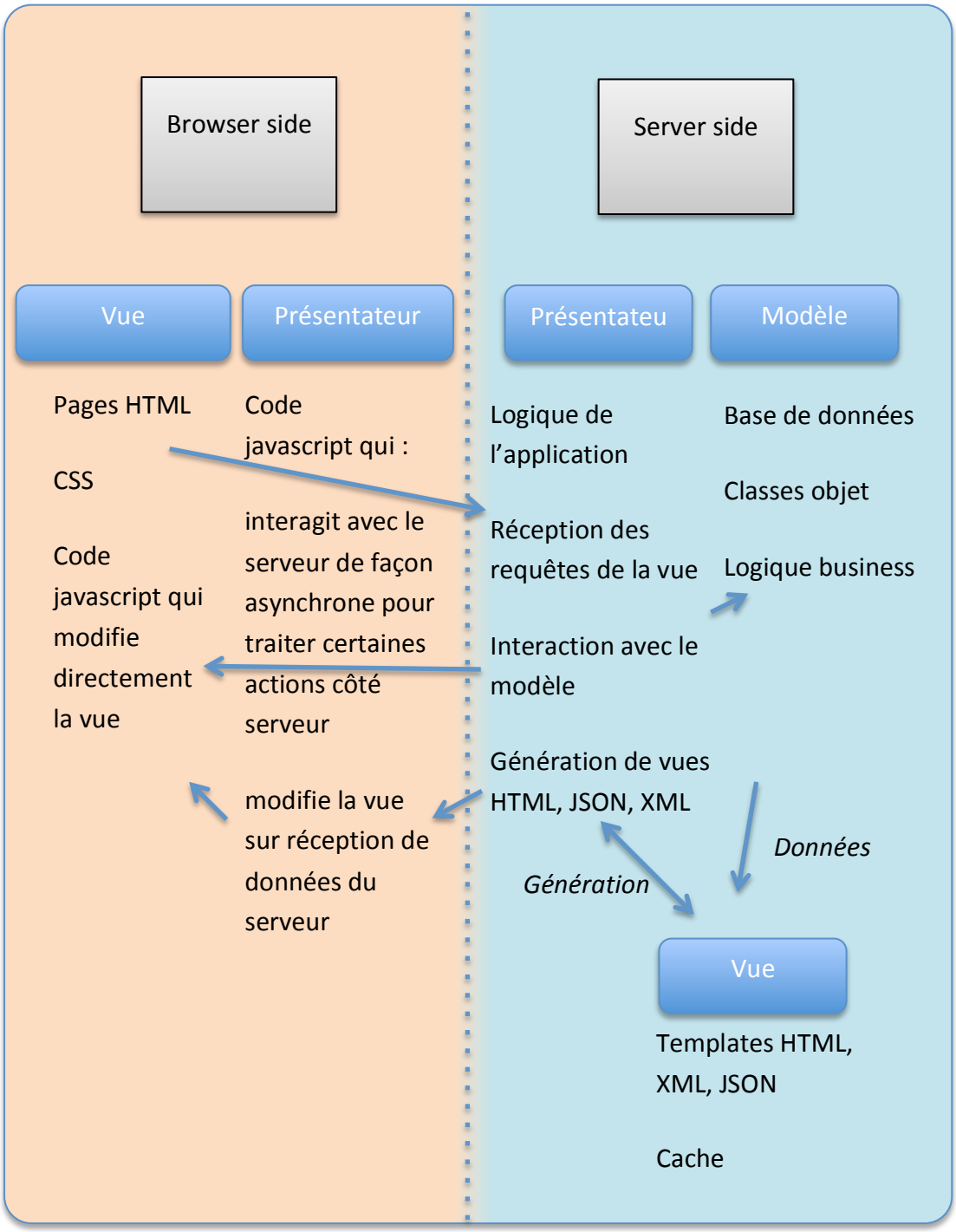
Le modèle est côté serveur et consiste en bases de données, classes objets et toute la logique business.

Toute la logique de l'application se trouve dans le présentateur côté serveur. En plus de répondre aux requêtes HTTP il dialogue avec le modèle et génère des vues à transmettre au navigateur. En ce sens on peut dire qu'une partie de la vue est côté serveur, surtout si des templates sont utilisés. Dans certains cas on aura même des vues entières gardées en cache côté serveur et retransmises telles quelles au navigateur si les données du modèle n'ont pas changé entre-temps.

3.3 Le Web comme application MVP avec vue intelligente

Si une partie de la logique applicative est déportée côté client en JavaScript[14] avec des requêtes HTTP asynchrones (AJAX) alors le schéma ci-dessus est légèrement modifié. Une partie du présentateur prend place du côté navigateur et la partie vue côté serveur n'est plus que du HTML mais aussi des données en format plus brut JSON ou XML qui seront traitées et affichées côté client. Le code JavaScript qui ne s'occupe que de modifier la vue est lui compris dans la vue.

[14] Voir chapitre 2, Analyse du poste de travail

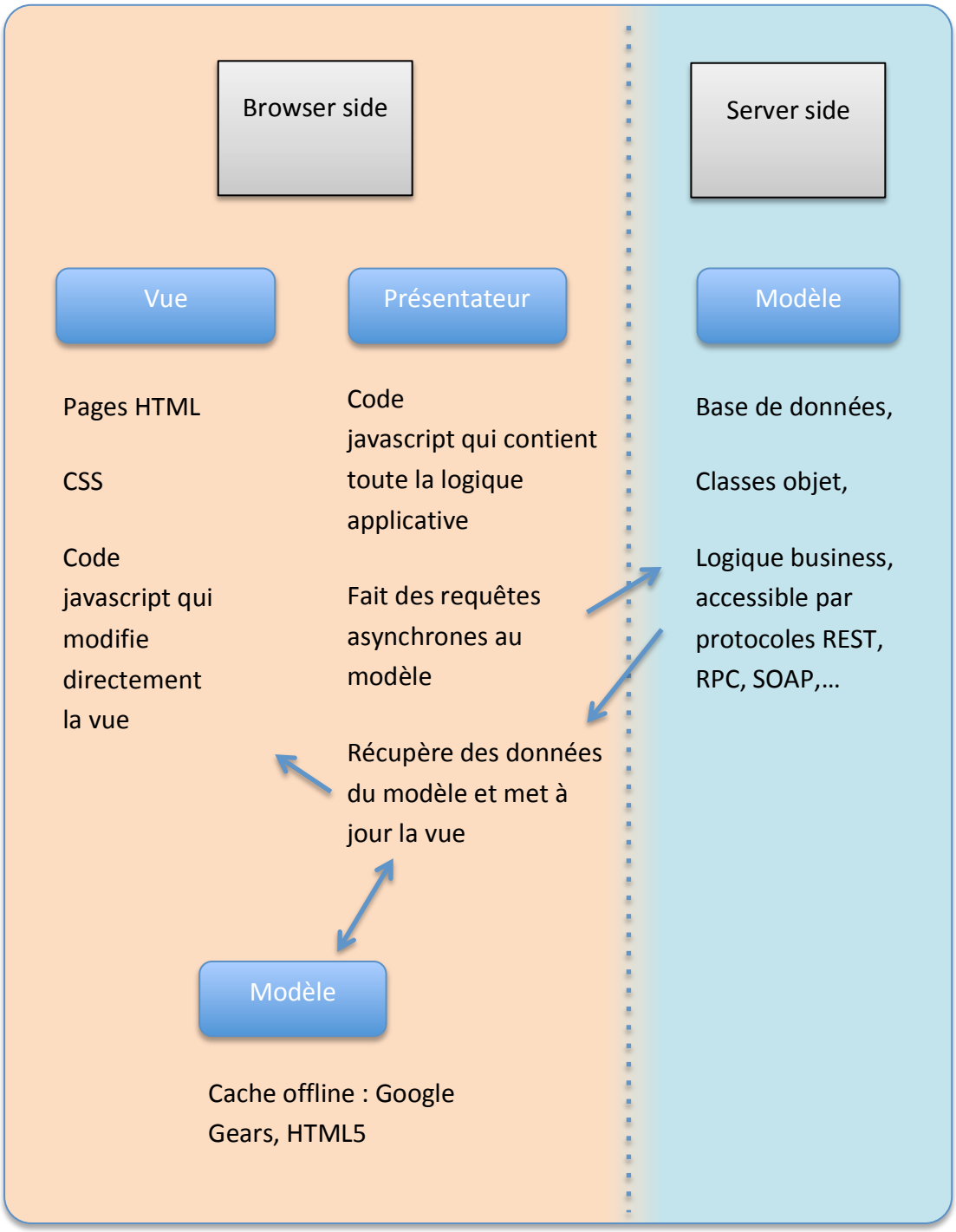


Application web avec vue intelligente

3.4 Application web autonome sous forme MVP

En poussant la réflexion un peu plus loin on se rend compte qu'on tend de plus en plus à déplacer toute la logique applicative côté client. Il ne reste alors côté serveur que le modèle qui lui possède une interface REST, RPC ou SOAP afin de permettre une communication depuis le navigateur. Mais même le modèle peut

aussi se trouver côté navigateur avec des systèmes de cache offline tels que Google Gears et ce qu'il sera possible de faire avec HTML5[15]



Application web autonome

[15] grâce au Local Storage

4 Technologies du web

Plusieurs langages informatiques et technologies sont utilisés dans le cadre de ce projet. Des livres entiers ont été écrit à leur sujet mais voici donc tout d'abord quelques éléments jugés essentiels pour bien appréhender la suite.

4.1 HTML

HTML est un langage de description de documents en mode texte destiné à la publication sur Internet. Couplé aux liens hypertextes qui permettent de faire des relations entre pages, il forme ce qu'on appelle le web[16]. De nombreuses versions du langage se sont succédées la première datant du début des années 90.

Le langage a une structure de type XML mais permet une mise en œuvre plus souple que des langages XML qui ont une syntaxe définie de façon très stricte. De nombreuses balises ne doivent par exemple pas nécessairement être fermées dans le cas du HTML. Certaines balises sont implicites comme html, head, body. Les noms de balises et attributs ne sont pas sensibles à la casse. Les guillemets ne sont pas obligatoires pour les valeurs d'attributs. Enfin il n'y a pas de syntaxe pour fermer une balise qui ne contient pas d'élément.

Une tentative entre 2000 et 2006 de faire évoluer le HTML en une version XML, le XHTML n'a pas rencontré beaucoup de succès. Elle était surtout motivée par un groupe de travail du World Wide Web Consortium (W3C)[17] qui voulait donner une dimension plus rigoureuse au HTML mais ne représentait pas toute l'industrie. Il n'empêche que l'intérêt s'est vu grandissant pour les standards du web et la nécessité de se mettre d'accord sur une norme commune, surtout suite à la perte du monopole du navigateur de Microsoft, Internet Explorer, qui a beaucoup limité l'innovation dans le secteur du web et son interopérabilité.

S'en est suivi la création par les concepteurs de navigateurs web d'un autre groupe de travail, le Web Hypertext Application Technology Working Group (WHATWG). La motivation était de capitaliser sur l'existant tout en faisant évoluer le HTML aux nouveaux besoins. En effet le web ne sert plus uniquement à la publication de documents mais sert de plus en plus de plateforme pour le

---

[16] "web" signifie toile d'araignée et est le nom qui a été donné à cet ensemble de pages HTML liées entre elles.
[17] Organisme de standardisation de technologies du web qui a notoriété

développement d'applications riches. De nouvelles balises et attributs deviennent nécessaires tandis que d'autres disparaissent parce qu'elles ne sont plus utilisées, sujettes à confusion ou ne sont pas suffisamment interopérables. Ces travaux ont été repris par le W3C et ont servi de base à l'élaboration du format HTML5.

Mais malgré ces évolutions le HTML reste insuffisant pour créer des interfaces d'applications web complexes. Ainsi, des contrôles auxquels on est habitués sur le poste de travail comme des onglets, des curseurs, des barres de progression, ne sont pas pris en charge nativement. HTML évolue prudemment en fonction des nouveaux usages (pas encore clairement définis) et de la volonté des concepteurs de navigateurs. Il ne permet, pour encore longtemps probablement, que de réaliser des descriptions très générales. Mais il a l'avantage grâce à cela d'offrir une énorme souplesse qu'on ne trouve dans aucun autre langage.

4.2 CSS

Parallèlement à l'évolution du HTML s'est développé un autre langage destiné à la mise en forme du contenu. En effet le besoin s'est vite fait ressentir par les développeurs web de séparer le contenu de la présentation comme dans d'autres contextes informatiques. Les premières feuilles de style CSS comme on les appelle sont ainsi apparues au milieu des années 90. Les spécifications de ce formats sont publiées par le World Wide Web Consortium (W3C) tout comme le HTML. Et au fur et à mesure de leur prise en charge par les navigateurs elles devinrent naturellement de plus en plus utilisées. Au début des années 2000 les développeurs consciencieux réalisent des sites au contenu HTML purement structurel et dédiant la présentation à CSS.

CSS signifie Cascading Style Sheets ou feuilles de style en cascade. Pourquoi en cascade ? Cela vient du fait qu'elles peuvent être appliquées à différents niveaux. Une feuille de style peut prendre la forme d'un fichier indépendant et être commune à tout un site web, à un ensemble de pages formant un ensemble cohérent ou appliquée à une seule page. La feuille de style peut aussi être intégrée directement au document html voire être incluse directement comme attribut des balises html dans le code source. CSS prévoit aussi l'application de feuilles de style différentes selon le mode de consultation (écran, impression,…), l'application de feuille de style utilisateur et des styles par défauts inclus dans les navigateurs. Ces différents niveaux et modes ont des priorités spécifiques mais

en règle générale on peut retenir que le concepteur du site contrôle (heureusement) la mise en forme de ce qui est fourni à l'utilisateur.

La syntaxe CSS est simple car elle a été conçue initialement afin de pouvoir être éditée intuitivement par un être humain. Elle est néanmoins puissante grâce à la quantité des options disponibles et la richesse des sélecteurs. Les sélecteurs sont le moyen de spécifier quels éléments html sont concernés par les différentes propriétés de style regroupées en règles de style. Afin de montrer la puissance de CSS et parce que plusieurs de ses concepts seront utilisés par la suite, illustrons son fonctionnement sur quelques exemples.

On peut écrire une règle de style concernant un élément html spécifique (tous les éléments du document html qui portent ce nom), par exemple « p » (les paragraphes). Voici un exemple de règle css avec 2 déclarations mettant les paragraphes en gras et en couleur rouge :

```
p {font-weight: bold; color: red;}
```

Pour cibler l'élément (unique) ayant un identifiant spécifique on utilise cette syntaxe :

```
#identifiant {…}
```

Enfin on peut atteindre tous les éléments ayant une certaine classe définie de cette façon :

```
.classe {…}
```

Ce sont les sélecteurs de base. Ceux-ci peuvent être combinés afin d'obtenir des expressions plus précises. On peut par exemple cibler les éléments du document html ayant un certain identifiant ou une certaine classe :

```
p#identifiant {…}
```

```
p.classe {…}
```

Ensuite on peut faire jouer le contexte afin de cibler un élément spécifique du document html contenu dans un autre qu'on peut aussi caractériser par un sélecteur. Si on souhaite cibler les éléments « p » avec une classe « classe » contenus dans un « div » avec identifiant « conteneur », il suffit d'ajouter un espace entre les 2 sélecteurs de cette façon afin de former un nouveau sélecteur :

```
div#conteneur p.classe {…}
```

Mais de cette manière on sélectionne tous les p.classe descendants de div#conteneur. Si on souhaite uniquement ceux qui sont enfants c'est la syntaxe suivante qu'il faut utiliser :

```
div#conteneur > p.classe {…}
```

Enfin il est possible de regrouper des sélecteurs indépendants afin de leur attribuer des règles communes en les séparant par des virgules :

```
div#identifiant, p.classe  {…}
```

Se pose alors la question de la priorité de ces expressions étant donné que de mêmes éléments du document html peuvent être ciblés à plusieurs reprises, de façon plus ou moins précise. C'est justement la précision du ciblage de ces éléments qui définira la priorité.

Si la règle CSS est directement intégrée dans la balise html via l'attribut « style » elle aura la plus grande priorité. Ensuite il faut considérer le nombre d'identifiants présents dans le sélecteur. Plus il y a d'identifiants plus on est spécifique car on impose une hiérarchie. En cas d'égalité pour le précédent critère on considère le nombre de classe, de la même façon. Et enfin on regarde le nombre d'éléments html dans le sélecteur. En cas d'égalité à ce stade c'est la règle qui apparaît en dernier qui a la priorité, l'ordre pouvant apparaître au niveau même d'un fichier CSS ou dans le document HTML.

4.3 Elements de formatage HTML & CSS

Afin de bien appréhender la conception d'une interface graphique en HTML & CSS il est primordial de bien comprendre les modèles de formatage utilisés par les navigateurs pour effectuer le rendu.

4.3.1 Formatage en flux

Attardons-nous d'abord sur le modèle de formatage en flux qui est le plus courant et qui affiche les éléments dans l'ordre où ils apparaissent dans le code source HTML. Nous n'aborderons pas les positionnements absolus ni flottants qui permettent de faire sortir des éléments du flux normal.

Il existe 2 grandes catégories d'éléments HTML qui diffèrent par leur rendu par défaut[18]. Ce sont les éléments de type **block** et ceux de type **inline**.

Exemples d'éléments de type block : h1, h2,…,p, ul, fieldset,…

Exemples d'éléments de type inline : a, em, img, input, label, strong,…

Il y aussi les éléments génériques div et span. Div est de type block et span est de type inline. Ils ont pour unique rôle de créer des conteneurs du type correspondant.

Un élément de type **block** s'affiche sur toute la largeur disponible de son conteneur. Plusieurs éléments de type block successifs s'affichent donc l'un en dessous de l'autre. Si un élément de type block est précédé ou suivi d'un élément de type inline il y aura un retour à la ligne entre les 2 éléments. Il est possible de lui donner des dimensions auquel cas il ne prendra plus nécessairement toute la largeur disponible mais conservera la propriété d'être seul sur sa ligne. On peut y inclure des éléments de type block ou inline.

Les éléments de type **inline** s'affichent les uns à côté des autres horizontalement et passent à la ligne lorsque cela est nécessaire. Ils s'intègrent dans le texte. A la différence des éléments de type block il n'est en principe pas prévu de leur donner des dimensions et ils ne peuvent contenir que des éléments du même type (inline).

Il existe une catégorie hybride appelée **inline-block**. Ces éléments se comportent comme les inline mais il est possible de les dimensionner. Tout élément inline auquel on donne des dimensions en CSS devient un inline-block. Et il est possible de convertir des éléments block en inline-block à l'aide de la propriété CSS « display ».

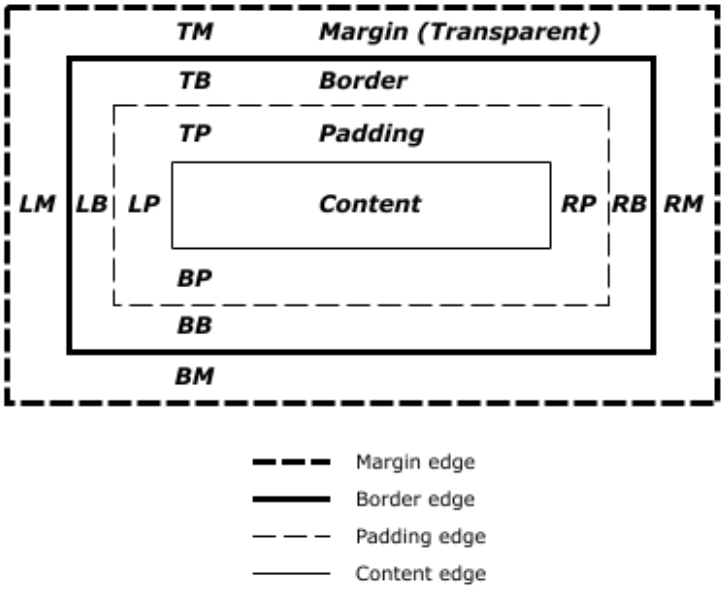
4.3.2 Taille des boîtes, bordures et marges

Un autre principe qu'il est utile de commenter est le fonctionnement du calcul des tailles des boîtes, des « paddings », bordures et marges. Il s'agit ici d'une description en toute généralité (il y a des exceptions). Lorsqu'une boîte a une taille fixée celle-ci est en réalité la taille du contenu uniquement. Pour obtenir la « taille totale » il faut y ajouter la taille des paddings, des bordures et des marges.

---

[18] Modifiable à l'aide de règles CSS.

Les paddings permettent d'ajouter un espacement entre le contenu et les bordures qui signent le contour visuel de la boîte. Ceux-ci sont dans la même couleur que l'arrière-plan de la boîte. Les marges sont elles transparentes par contre et prennent donc la couleur de l'arrière-plan du conteneur de la boîte. De plus 2 marges voisines se combinent. On a ainsi que l'espace entre les bordures de 2 boîtes voisines vaut la taille de la plus grande marge les séparant.



4.3.3 Effets visuels : débordement et visibilité

Un autre aspect important concerne le sort réservé aux contenus qui sont trop grands pour tenir dans l'espace réservé par un conteneur. Cela peut se produire si le conteneur a une taille fixe et que l'élément contenu ou la juxtaposition des éléments contenus a une taille supérieure.

Ceci est géré par la propriété « overflow » en CSS. Elle définit que faire du contenu qui sort du conteneur. Par défaut sa valeur est « visible », ce qui signife que le contenu sera affiché en dehors du conteneur par débordement ! Si ce n'est pas ce que l'on souhaite il faut le spécifier autrement. On peut demander de simplement rogner le contenu, d'afficher des barres de défilement de façon permanente ou seulement lorsqu'il y a débordement.

Au cas où le conteneur n'a pas de taille fixe le problème ne se pose pas car celui-ci s'adaptera généralement à ce qu'il contient. Pour les cellules de tableaux ceci n'intervient pas non plus car même en spécifiant des tailles les cellules s'adaptent à leur contenu. Ca paraît étrange mais c'est généralement ce que l'on souhaite pour un tableau.

Un autre aspect concerne la possibilité de masquer des éléments en CSS. Ici il faut constater la présence de 2 déclarations similaires : « display :none » et « visibility :hidden ». Mais elles n'ont bien sûr pas le même effet. La première aura pour conséquence de masquer l'élément concerné mais en plus de le traiter comme s'il n'existait plus dans la structure. Il ne fera plus partie du flux d'affichage. La deuxième instruction, elle, ne fait que masquer l'élément tout en conservant sa place dans le flux ce qui signifie qu'il continue de jouer son rôle dans la présentation. Il faut faire attention d'utiliser la bonne instruction en fonction du contexte.

4.4 PHP

PHP est un langage de programmation interprété destiné au web. Il a une syntaxe empruntée aux langages C et Java mais est beaucoup moins strict que ceux-ci. La grosse différence est que le langage n'est pas typé[19]. Cela facilite l'apprentissage aux débutants mais malheureusement cela limite quelque peu les possibilités et il faut faire attention à la sécurité étant donné que n'importe quelle valeur peut être donnée aux variables à l'exécution.

PHP est surtout utilisé, et c'est ce qui nous concerne, afin de générer côté serveur du code HTML, CSS et parfois JavaScript de façon dynamique. L'interpréteur PHP est interfacé avec un serveur web tel que Apache (généralement) ou IIS de Microsoft. C'est le serveur web qui reçoit les requêtes HTTP des clients (navigateurs web). Le serveur agrège alors les ressources demandées. Une partie peut être livrée directement, les ressources statiques (fichiers ou parties de code HTML, CSS, JavaScript). Le code PHP doit être envoyé au moteur PHP pour interprétation et le résultat sera combiné aux ressources statiques.

Une requête HTTP contient toujours l'adresse d'une ressource mais la requête peut être de plusieurs types dont les plus utilisés sont « GET » et « POST ». Le mode « GET » est le mode par défaut. Il permet de simplement demander de fournir la ressource située à l'adresse indiquée. Ce mode peut aussi être utilisé pour transmettre des données au programme situé à cette adresse sur le serveur. Les données sont alors transmises à la fin de l'URL, après un « ? » sous forme de paires « clé=valeur » séparés par des « & ». L'autre mode HTTP est le mode POST. C'est le mode prévu initialement par le protocole HTTP pour l'envoi de données à une ressource côté serveur. PHP peut utiliser nativement ces 2 modes pour recevoir des données des clients.

---

[19] Les variables et leur type ne doivent pas être déclarées avant utilisation.

Le langage était destiné à la base aux débutants et en voulant leur faciliter la tâche certaines techniques peu recommandables du point de vue la sécurité notamment avaient été mises en place. Mais des avancées ont été réalisées récemment rendant le langage progressivement plus professionnel. Il y a d'abord la suppression d'une extension facilitant la récupération de données en GET et POST en définissant les variables issues des formulaires de façon globales, « register_global ». Ce n'est heureusement plus possible maintenant. A l'inverse une extension aussi destinée à aider le débutant mais augmentant la sécurité a elle aussi été supprimée. Il s'agit de « magic_quotes ». Elle avait pour rôle d'échapper automatiquement toutes les chaines de caractères reçues de l'utilisateur afin d'éviter des failles d'injection SQL au cas où ces données seraient introduites dans une base de données. C'est maintenant au programmeur d'y penser et c'est mieux car cet échappement n'était pas toujours nécessaire[20] et pouvait rendre le programme non portable sur des environnements où cette extension n'était pas activée.

PHP est de plus devenu dans sa dernière version un langage orienté objet assez complet avec le support de classes (attributs, méthodes et instances de classes), l'héritage, la visibilité dans les classes, les classes abstraites, les interfaces. Deux limitations importantes sont qu'on ne peut pas faire de l'héritage multiple et que la surcharge de méthodes avec des paramètres de type différent n'est pas possible puisque le langage n'est pas typé.

Le langage PHP est livré avec une bibliothèque très riche ce qui le rend très efficace. Il est très facile par exemple de l'interfacer avec des bases de données grâce aux connecteurs inclus par défaut. Dans notre cas on pourra bénéficier de l'API SimpleXML qui converti un fichier xml en objet PHP qu'on peut parcourir facilement.

4.5 JavaScript & jQuery UI

JavaScript est un autre langage de programmation mais utilisé côté client au sein du navigateur afin d'ajouter de la logique à la partie présentation HTML+CSS. Le langage date de 1995 et est standardisé par l'ECMA. Sa mise en œuvre dans les navigateurs a induit quelques différences mais il s'agit aujourd'hui d'une technologie mature et bien supportée. Ces dernières années on a assisté à une augmentation très importante des performances JavaScript par les navigateur

[20] Au cas où le programme n'interragit pas avec une base de données

grâce à l'utilisation de moteurs indépendants écrits en langage C et effectuant la compilations des instructions JavaScript à la volée.

jQuery UI est une bibliothèque JavaScript permettant de combler certains manques de HTML pour la création d'applications riches en ajoutant des widgets importants comme la barre de progression, le « date picker » ou le « slider ». Il s'agit aussi d'une surcouche à JavaScript introduisant un niveau d'abstraction facilitant la mise en œuvre de certaines opérations comme l'accès aux éléments html, à leurs propriétés et permettant de créer des boîtes redimensionnables par exemple. jQuery UI est un package additionnel de la bibliothèque de base jQuery sur laquelle il repose.

jQuery propose une fonction multi-usage essentielle qu'il est utile de détailler ici. Cette fonction porte le nom « jQuery » ou son alias « $ ».

Si on passe comme argument à cette fonction une définition d'une autre fonction celle-ci est considérée comme un callback et sera exécutée une fois que le document html sera complètement chargé par le navigateur. Elle n'attend par contre pas le chargement d'éventuelles images. Cette fonction nous sera très utile car c'est par ce biais que nous pourrons insérer dans le document des widgets non-pris en charge par HTML aux endroits que nous aurons réservés. Cela ne peut se faire qu'une fois que la structure du document est complètement chargée.

La fonction jQuery accepte aussi en paramètre un sélecteur CSS. Son rôle est alors de rechercher tous les éléments HTML qui correspondent à ce critère et de les convertir en objets jQuery. On pourra alors utiliser l'abstraction jQuery pour manipuler ces éléments via l'objet retourné. Les sélecteurs classiques CSS[21] sont pris en charge : recherche par balise, par classe, par identifiant, par type ou une combinaison. Exemple : $('div.foo') retourne un objet jQuery représentant tous les div's avec pour classe « foo ». Cette fonction nous permettra notamment de localiser les emplacements que nous aurons réservés dans le code html pour des widgets non-pris en charge afin de les convertir.

Enfin on peut appeler la fonction jQuery avec du code html. L'intérêt est de créer de nouveaux morceaux de code html qui pourront être manipulés en jQuery et éventuellement intégrés dans le document en cours.

---

[21] Voir chapitre 4.2, CSS

Partie 2 : Implémentation

Venons-en à l'objet de ce mémoire, le développement d'un simulateur d'interface graphique. Il s'agira en réalité d'un interpréteur usiXML ou plus précisément d'un interpréteur de la dernière section du modèle usiXML : « concrete interface ». Elle permet de décrire une interface indépendamment de toute plateforme et de tout langage de programmation, c'est ce qui nous intéresse. D'autres sections aux niveaux d'abstraction plus élevés existent comme la description des tâches utilisateur, la description d'interface indépendamment des modalités,… Nous n'en tiendrons pas compte pour notre interpréteur.

Celui-ci sera implémenté en utilisant judicieusement les 4 technologies décrites brièvement ci-dessus : HTML, CSS, PHP et jQuery. En effet aucune ne suffit à elle-même pour obtenir ce que nous souhaitons. HTML se limite à la définition d'une structure sémantique d'éléments mais il est insuffisant pour créer tous les éléments d'interface prévus par usiXML. CSS permet d'en gérer la présentation. jQuery nous viendra en aide pour apporter les contrôles manquants dans le standard HTML ainsi que leur présentation CSS. Il gèrera aussi certains comportements comme le redimensionnement de boîtes. PHP sera le langage de programmation central de notre interpréteur. La conversion sera donc effectuée côté serveur.

Le fonctionnement sera le suivant. L'utilisateur pourra se connecter à l'aide de son navigateur au serveur sur lequel l'interpréteur aura été installé. En écran d'accueil il trouvera la possibilité de télécharger un fichier usiXML sur le serveur ou consulter la liste des fichiers déjà présents. Il pourra au départ de cette liste demander l'interprétation d'un document. Une fenêtre qui y est décrite sera alors interprétée comme une fenêtre « pop-up »[22] du navigateur aux dimensions précisées.

Le téléchargement d'un fichier sur le serveur pourra se faire de 2 façons, soit par l'envoi d'un fichier présent sur le poste de travail, soit en indiquant une adresse URL où se trouve un fichier distant qui peut être copié. Il sera aussi possible de demander l'interprétation directe d'un fichier distant, sans copie dans l'espace de stockage du serveur.

---

[22] Il s'agit d'une fenêtre secondaire sans barre de menus ni barres d'outils qu'on peut ouvrir en javascript.

1 Architecture

Le choix a été fait d'implémenter l'interpréteur en adoptant une structure de type MVP, ceci afin d'illustrer les concepts vus dans la première partie mais aussi et surtout afin d'obtenir un code compréhensible et donc moins sujet aux erreurs, facilement maintenable et permettant des évolutions futures. Cela est d'autant plus critique que dans notre cas 4 langages informatiques vont se trouver entremêlés dans un seul code source.

Voyons d'abord grossièrement de quelle façon la découpe dans les 3 composantes MVP sera effectuée pour notre interpréteur.

La **vue** consistera en :

1°) la représentation au sein du navigateur de descriptions usiXML préalablement fournies à l'application, c'est-à-dire téléchargées sur le serveur. Il s'agira donc du code HTML, CSS et jQuery générés par l'interpréteur, ceux-ci étant interprétés à leur tour par le moteur de rendu du navigateur afin de former l'interface graphique. Un élément window présent dans la structure usiXML correspondra à un document HTML.

2°) Une autre vue sera la page d'accueil de l'application qui permettra de lister les fichiers usiXML disponibles sur le serveur et pour lesquels on peut demander une interprétation. Cette page d'accueil permettra aussi de télécharger sur le serveur de nouveaux fichiers usiXML.

3°) Une partie de la vue se situera également côté serveur. On y trouvera les templates HTML & PHP utilisés pour générer les vues. En effet certaines parties ne dépendent pas de la description usiXML et peuvent être réutilisées à chaque interprétation. De cette manière on ne doit pas générer de code à chaque fois. De plus on peut se baser sur ces canevas et y introduire aux endroits voulus les parties changeantes. On procède à l'envers. Au lieu de générer du code HTML par un script PHP c'est en quelque sorte un document HTML dans lequel on intègre des données PHP. On gagne ainsi en lisibilité et en maintenabilité. De la même façon et pour les mêmes raisons on aura aussi un template pour la page d'accueil de l'application.

4°) Les règles de styles CSS générales propres à l'interpréteur ainsi que la librairie jQuery seront aussi hébergées à cet endroit.

En ce qui concerne le **modèle** c'est ici un peu particulier dans ce cas car les données de notre application interpréteur seront les représentations informatiques des interfaces graphiques à interpréter. Le modèle comprendra donc :

1°) Les représentations sous forme de classes des éléments usiXML. La double structure hiérarchique devra y être représentée :

- hiérarchie des éléments des plus abstraits aux plus concrets (héritage). Exemple : inputText hérite de 2DgraphicalIndividualComponent qui hérite de 2DgraphicalCio qui hérite à son tour de cio.

- hiérarchie des contenus : 1 window contient une ou plusieurs boîtes de type quelconque, qui peuvent en contenir à leur tour ou contenir des éléments finaux.

2°) D'autres classes héritant des précédentes mais représentant les objets dans les technologies prévues. Ces classes doivent permettre de générer le code HTML, CSS et jQuery rendant les widgets et les conteneurs opérationnels. Il faudra faire attention à ce que les propriétés des conteneurs ayant un impact sur les éléments contenus soient correctement gérées. On imagine déjà que la génération des différents codes ne pourra se faire qu'une fois que toute la hiérarchie des objets sera créée.

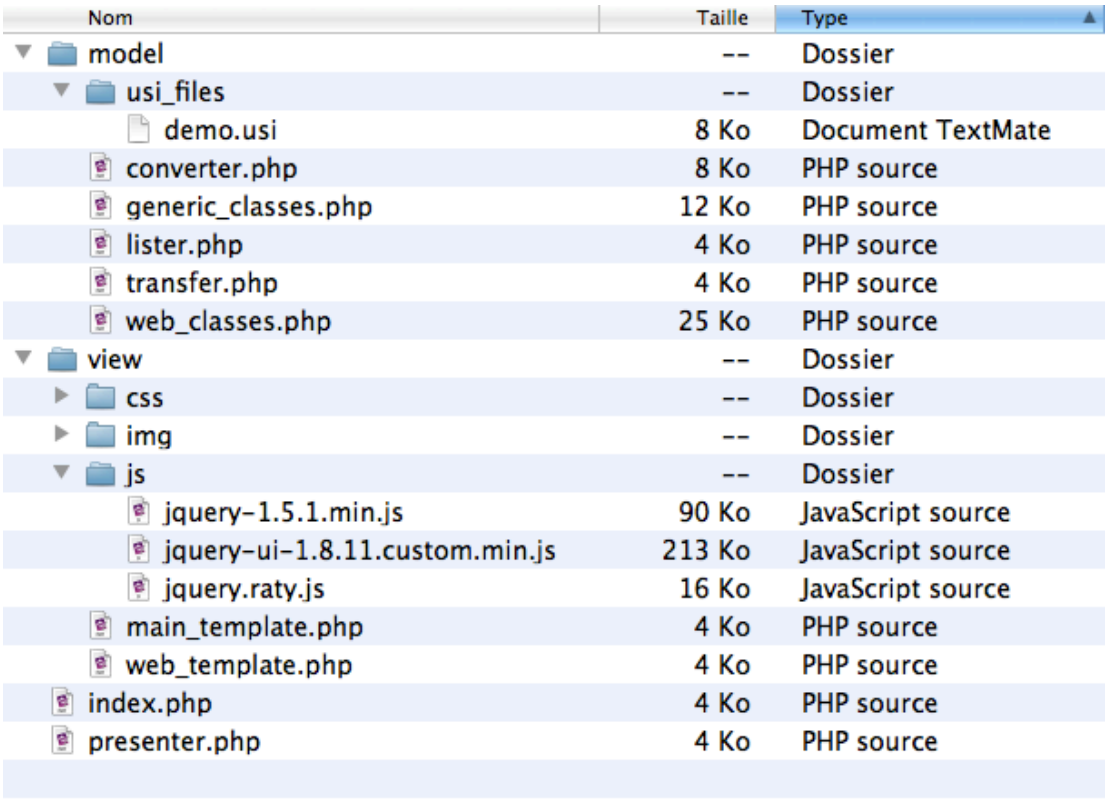
3°) Une classe Converter servira de source de données pour le template d'interface graphique. On pourra ainsi dans le template faire des appels statiques à des méthodes de Converter afin de compléter les emplacements prédéfinis.

4°) Les fichiers sources usiXML téléchargés sur le serveur font aussi partie du modèle. On y ajoute une classe Lister qui servira de source de données (comme au point précédent) qui sera utilisée par le template principal de l'application.

5°) Enfin on prévoit une classe Transfer dont le rôle de se charger du téléchargement de fichiers usiXML vers le serveur, de vérifier leur conformité et de les placer le cas échéant dans le dossier adéquat. Cette classe servira aussi de source de données pour le template principal afin d'indiquer les éventuelles erreurs lors du traitement du fichier envoyé.

Enfin le **présentateur** qui est le cœur logique de l'application prendra la forme d'un seul fichier source. Ce n'est pas lui qui s'occupera directement de l'interprétation, cette tâche étant confiée à une classe du modèle. Il aura pour unique rôle d'analyser les requêtes HTTP soumises par l'utilisateur par

l'intermédiaire de son navigateur afin d'effectuer les bonnes actions sur le modèle et de soumettre les vues adéquates. Vu l'existence de templates et de sources de données au niveau du modèle on tentera de faire intervenir le moins possible le présentateur pour générer les vues. On tentera de faire en sorte que les templates puissent directement aller chercher les données dans le modèle.



Structure de l'application

2 Détails du modèle

2.1 Classes génériques

Les classes représentant directement les éléments de concrete interface usiXML sont regroupés dans un fichier appelé « generic_classes.php ». Ces classes ne permettent rien d'autre que cela. Il est prévu de les étendre afin de permettre une représentation de ces éléments dans un contexte et une technologie précise.

Ces classes comprennent donc des attributs dont les noms sont calqués directement sur les attributs des éléments usiXML correspondants. La hiérarchie des éléments usiXML est matérialisée par des liens d'héritage entre les classes. La classe InputText hérite ainsi de la classe abstraite TwoDgraphicalIndividualComponent qui hérite elle-même hérite de la classe abstraite TwoDgraphicalCIO qui hérite enfin la classe abstraite Cio. Les classes ne représentant pas des objets finaux sont déclarées abstraites. Il faut nécessairement les étendre pour les utiliser. Vous vous demandez peut-être pourquoi les noms des classes sont TwoD[…] et pas 2D[…]. C'est simplement dû à une limitation de PHP qui impose que les noms de classes commencent par une lettre.

Les constructeurs prennent tous en argument un tableau d'attributs de la forme clé->valeur, un numéro d'identifiant et une référence vers un parent. Le tableau d'attributs permettra de donner des valeurs aux attributs de la classe. Ce sont les attributs de la description usiXML. Le numéro d'identifiant donné en paramètre sera utilisé au cas où un identifiant n'est pas fourni dans les attributs. Dans ce cas l'identifiant « myId_ » suivi du numéro d'attribut sera défini pour cet élément. Enfin la référence vers le parent permettra de parcourir la hiérarchie des éléments dans le sens ascendant. Cela pourra être utile pour la conversion de l'interface dans un autre langage.

Les attributs de classe sont marqués « protected » par sécurité. Cela signifie qu'on ne peut y accéder qu'à l'intérieur de la classe ou de ses descendants (héritage). Il n'est en principe pas nécessaire d'y accéder dans d'autres contextes. Si cela devait être nécessaire il est toujours possible de mettre en place des « Getters » voire des « Setters » comme on les appelle.

Le langage PHP n'est pas typé mais on veillera tout de même à définir les attributs dans le format le plus courant pour ce langage. Tous les attributs reçus en paramètres seront en format texte mais ce n'est pas le format le plus pertinent pour représenter des valeurs booléennes ou numériques dans un

langage de programmation car ils ne permettent pas de tester facilement leur valeur ou de les modifier. Les attributs booléens de usiXML seront donc convertis en valeurs booléennes et les attributs numériques en valeurs numériques.

Un attribut important est ajouté par rapport aux attributs usiXML pour ce qui concerne les conteneurs c'est-à-dire la classe TwoDgraphicalContainer. Il s'agit de l'attribut « inside ». Il permet de matérialiser l'autre niveau de hiérarchie, la hiérarchie des contenus. En effet on doit pouvoir indiquer quels éléments sont contenus dans les différents types de boîtes. L'attribut est un tableau car il peut y avoir plusieurs éléments contenus. Il est marqué « protected » également mais une méthode « put($comp) » est prévue afin d'ajouter des éléments aux conteneurs lors du parsing du fichier XML.

Le même type de procédé est utilisé pour les éléments comboBox et slider mais étant donné qu'il ne s'agit pas de conteneurs et qu'ils n'éritent donc pas de TwoDgraphicalContainer, il a fallu le spécifier de façon particulière pour ces composants. Un comboBox contient des « items », c'est le nom choisi pour l'attribut de classe couplé à une méthode « addItem » qui permettra d'ajouter des éléments au tableau. Un slider peut contenir un ou plusieurs « cursors ». On a donc prévu un attribut du même nom qui les contiendra et une méthode « addCursors ».

2.2 Classes web

Les classes web sont celles qui contiennent les instructions nécessaires pour générer le code HTML, CSS et JavaScript représentant la description d'interface. Elles ont été distinguées des classes génériques afin de permettre une évolution de l'interpréteur supportant d'autres modes de conversion telle qu'une version mobile.

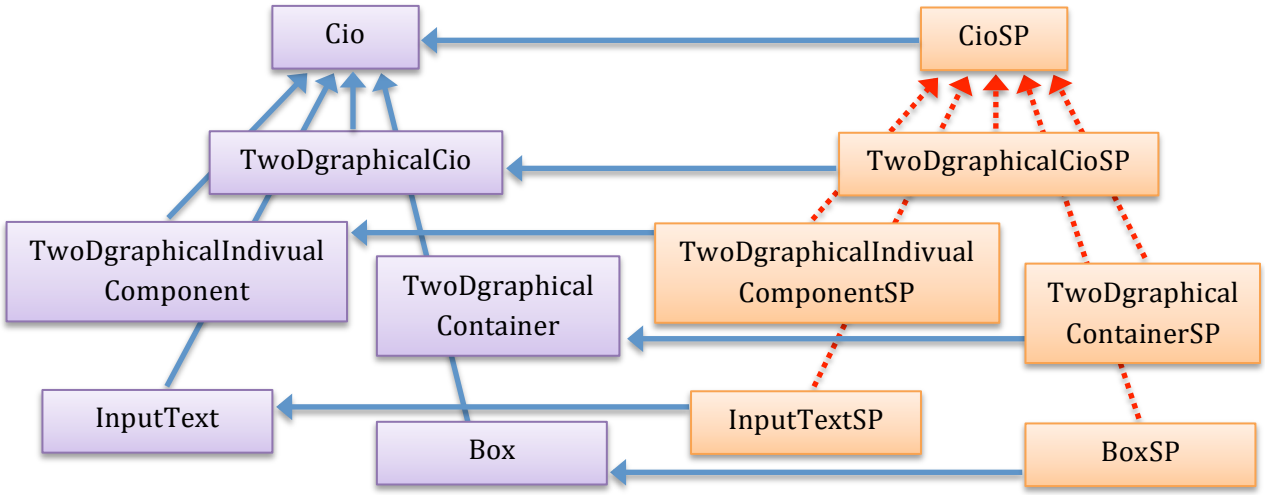
Détaillons ici quelques notions utiles pour comprendre le fonctionnement de la conversion.

Parallèle avec les classes génériques

Ces classes correspondent exactement aux classes génériques et héritent une-à-une de celles-ci. On aurait voulu pouvoir faire de l'héritage multiple et hériter et des classes génériques et des classes web plus abstraites afin d'avoir une véritable symétrie. Cela aurait pu éviter de répéter des parties de code de conversion pour des éléments finaux dépendant de mêmes classes abstraites et

partageant certaines propriétés de rendu. Mais ce n'est malheureusement pas possible en PHP.

Seules les classes représentant des éléments finaux sont donc implémentées puisque les autres sont inutiles ici. Attention on utilise bien les attributs des classes génériques abstraites.



Parallèle entre classes génériques (gauche) et classes web (droite)

Méthodes à implémenter

Pour chaque classe concrète on s'impose d'implémenter 3 méthodes html(), script() et style() qui définiront respectivement le code html, le code JavaScript (jQuery) et les règles de style CSS nécessaires pour représenter correctement l'élément. Dans certains cas il n'y aura peut-être pas besoin de définir de règles CSS ou de code JavaScript. Dans ces cas les méthodes renverront simplement la chaîne vide. On s'impose d'implémenter ces 3 méthodes car ici aussi on aurait bien voulu définir une interface ou une classe abstraite dont les classes devraient hériter et pour lesquelles l'interpréteur PHP imposerait que ces 3 méthodes soient implémentées. Mais, on l'a vu, ce n'est pas possible.

Valeurs par défaut

Systématiquement on doit au sein de ces 3 méthodes utiliser les attributs de classe afin de les traduire en syntaxes adéquates. Il n'y aura pas nécessairement de valeur pour tous les attributs et il est donc important de faire attention aux valeurs par défaut supposées du langage usiXML afin qu'elles correspondent aux valeurs par défaut des attributs de nos classes. Le code généré devra également tenir compte des valeurs par défaut des langages de destination utilisés et

écraser ces valeurs le cas échéant car elles peuvent être en complète contradiction avec ce que l'on souhaite.

En règle générale, dans le doute, il est préférable de donner des valeurs par défaut afin d'éviter de laisser au navigateur la liberté d'interpréter le code comme il le souhaite.

Tous les attributs booléens ont reçu une valeur par défaut car, par définition, ils ne peuvent prendre que 2 valeurs et la valeur « non-défini » ne signifie rien ou plutôt peut être mal interprétée. On préfère donc donner une valeur qui correspond vraisemblablement au souhait de celui qui a réalisé la description d'interface. Par exemple l'attribut « isVisible » recevra la valeur « true » s'il n'est pas spécifié autrement. Autre exemple : l'attribut « isPassword » d'un élément inputText a pour objectif de rendre celui-ci plus spécifique afin qu'il soit adapté à la prise en charge de mots de passe. Sa valeur par défaut doit être « false » car si rien n'est précisé on doit garder le contexte d'utilisation le plus large possible.

Une autre raison pour donner des valeurs aux attributs booléens est que si l'on effectue un test conditionnel sur un attribut qui n'est pas défini le test renverra systématiquement false. Or ce n'est pas nécessairement la valeur par défaut souhaitée, bon nombre d'attributs devant avoir la valeur par défaut « true ». Si des valeurs par défaut n'étaient pas donnés à tous les attributs il faudrait avant d'effectuer les tests conditionnels faire un appel à la méthode « isset() » de PHP afin de s'assurer que les variables-attributs sont définis. Ce serait fastidieux étant donné que ce type d'opération est très courant dans le code.

Foreach

La structure XML du document usiXML est composée d'éléments imbriqués formant une hiérarchie. On l'a vu une « window » contient une « box » qui peut en contenir d'autres etc… Cette structure hiérarchique est matérialisée par des tableaux (attributs « inside ») contenant des références vers les éléments inclus.

Le langage principal généré, HTML, étant également un langage balisé, le même type de hiérarchie doit être reproduit. Le résultat de la méthode « html() » d'un conteneur ne peut donc être fourni qu'une fois que le code html de tous les éléments enfants aura été généré. Et à fortiori c'est le cas aussi pour l'élément racine, « window ».

Pour tous les conteneurs il est donc nécessaire d'assembler d'une part la balise ouvrante avec tous ses attributs, le code html de tous les enfants, et terminer par la balise fermante. Le code html de tous les enfants est obtenus grâce à une

boucle « foreach » effectuée sur le tableau « inside » et à un appel de la méthode html() sur tous les éléments.

Pour le code jQuery et les règles CSS cet aspect n'intervient pas étant donné que les éléments sont ciblés de manière directe grâce à leur id. Il n'y a pas besoin de respecter de hiérarchie ni l'ordre d'ailleurs. Mais dans le cas d'un conteneur c'est tout de même aussi à lui de faire appel aux méthodes correspondantes des éléments contenus afin d'obtenir une représentation complète.

2.3 Converter

Converter est la classe qui s'occupe de la conversion de la description usiXML en objets des classes web et qui permet d'obtenir les différentes composantes de l'interface convertie, au départ de cette structure d'objets.

La description usiXML est fournie sous forme d'objet SimpleXMLElement à cette classe. Le parsing XML de base n'est donc pas pris en charge par cette classe et il n'y aura donc pas d'erreur de format XML. On peut par contre avoir à faire à des descriptions usiXML non prises en charge telles que celles qui ne contiennent pas de « cuiModel » ou pas de « window ». Cette classe se doit de générer des exceptions dans ces cas.

La conversion proprement dite est effectuée par une méthode effectuant un grand « switch-case » sur tous les éléments d'un même niveau dans la hiérarchie des contenus. Les noms des éléments constituant la description usiXML sont testés par rapport à une liste des éléments pris en charge. Lorsqu'un élément est reconnu l'objet d'une des classes web correspondant est créé et ajouté à la liste des objets à retourner. S'il s'agit d'un élément conteneur la méthode est appelée récursivement afin d'ajouter à l'objet (dans son attribut « inside ») les éléments enfants. C'est aussi le cas pour les éléments « comboBox » et « slider » qui ne sont pas des conteneurs dans le sens qu'ils n'héritent pas de « 2DgraphicalContainer » mais ils ont aussi des éléments enfants XML qu'il est nécessaire d'ajouter à nos objets. Il s'agit des « items » pour le premier et des « cursors » pour le second. Au final, après avoir effectué tous les appels récursifs, la méthode retourne un tableau contenant les objets situés à la racine de la description usiXML. Il n'y a en principe que des éléments « window ». En tout cas c'est ce qu'on attend.

« Converter » est une classe aux membres statiques. Dans le cas où la conversion se passe correctement il est prévu de l'utiliser comme source de données pour les templates. Pour cela les méthodes « getBody() », « getScript() » et « getStyle() » permettent d'obtenir respectivement le code HTML, les

instructions jQuery et les règles de style CSS représentant une fenêtre décrite en usiXML. On aussi prévu une méthode « getTitle() » qui permet d'obtenir le titre de la fenêtre car celui-ci n'est pas compris dans le corps (body) du document HTML. Il devra apparaître au sein de la barre de titre de la fenêtre du navigateur. Cette information doit donc être disponible séparément afin d'être utilisée au bon endroit dans le template. De plus, il est nécessaire de proposer dans cette interface un moyen d'obtenir la taille de la fenêtre. Les dimensions seront utilisées par la page d'accueil de l'interpréteur afin d'ouvrir la fenêtre à interpréter dans les bonnes dimensions. Enfin on rend accessible 2 métadonnées importantes de la description usiXML : le nom de l'auteur et le commentaire.

Cette classe est utilisée dans 3 contextes : Lors du téléchargement d'un fichier vers le serveur afin de vérifier sa conformité ; sur la page d'accueil lors du listage des descriptions usiXML disponibles afin de vérifier une nouvelle fois si elles sont conformes au cas où des fichiers auraient été déposés sur le serveur par un autre moyen que l'interface graphique et afin d'obtenir les informations utiles au listage ; enfin Converter est bien sûr aussi utilisé pour l'affichage des fenêtres converties. Notons qu'on ne conserve aucun résultat, la conversion est effectuée à chaque fois. On peut se le permettre car cela est très rapide. De plus cela permet d'ajouter et supprimer manuellement des fichiers usiXML dans le répertoire du serveur.

Enfin signalons que cette classe a été conçue d'une façon générique dans le sens qu'elle peut supporter d'autres modes de conversion, une version mobile par exemple. Il suffit pour cela de lui fournir d'autres classes spécialisées représentant les éléments usiXML d'une autre façon. On doit cependant rester dans le modèle du web car le présupposé est que l'on garde la découpe dans les 3 composantes HTML, CSS et JavaScript. Ces éléments doivent pouvoir être obtenus sur l'élément racine qui reste de type « window ».

2.4 Transfer

La classe « Transfer » est une classe utilisée de façon statique comme le Converter. Son rôle est de réaliser le téléchargement de fichiers usiXML vers un dossier pré-défini du serveur mais seulement après avoir vérifié qu'il était conforme à ce qu'on attend.

Des erreurs peuvent se produire à plusieurs niveaux. Le fichier pouvant être distant, l'URL spécifiée peut déjà ne pas être valide. Ensuite que le fichier soit distant ou sur le serveur en attente d'être transféré vers son emplacement définit, un problème peut survenir lors de l'ouverture ou de la lecture du fichier.

Une erreur peut encore apparaître lors du parsing XML par SimpleXMLElement, si le document ne respecte pas une syntaxe XML valide. C'est le cas par exemple si des balises ne sont pas correctement fermées, si les éléments ne sont pas correctement emboîtés, si les attributs ne sont pas indiqués de façon adéquate,… Pour chacune de ces erreurs pouvant être rencontrée une exception est lancée et traitée afin de ne pas arrêter toute l'application.

Si ce premier parsing s'est effectué correctement on passe alors à la conversion, qui peut aussi générer des exceptions comme on l'a vu au chapitre précédent ; Enfin la dernière opération critique concerne la copie du fichier spécifié par l'utilisateur dans son emplacement final au cas où il est valide. Cette opération nécessite les droits d'accès adéquats au fichier et dépend de l'espace disponible sur le disque dur du serveur et peut donc être la source d'erreur également.

Tous ces risques doivent être pris en compte et il faut informer l'utilisateur avec des messages pertinents au cas où une opération s'est mal passée. C'est la responsabilité de cette classe.

3 Détails de la Vue

Concernant la vue, décrivons un peu le schéma des templates puisque la partie côté navigateur est basée sur ceux-ci.

3.1 Template de conversion

Commencons par le template utilisé pour représenter une fenêtre. Il s'agit d'un template HTML. On doit donc commencer par indiquer le doctype. Il s'agit d'une ligne qui permet de spécifier au navigateur quelle version de HTML est utilisée. Celui-ci adaptera alors son mode de rendu en conséquence. Le doctype est nécessaire car sans lui les navigateurs supposeront que le code ne respecte pas les standards du web et interprèteront celui-ci dans un mode de compatibilité. Cette technique appelée « doctype switching » a dûe être mise en place au moment du regain d'intérêt pour les standards afin que les navigateurs puissent les prendre en charge pleinement sans casser l'affichage d'anciens sites web développés pour Internet Explorer qui avait certaines interprétations erronées. On utilise ici le doctype HTML5 qui est simplement :

```
<!DOCTYPE html>
```

Commence ensuite le document HTML proprement dit dont la structure de base comprend une section d'en-tête et le corps du document :

```
<html>
<head></head>
<body></body>
</html>
```

Dans la section d'en-tête on spécifie le type MIME du contenu et le mode d'encodage des caractères. C'est ici du « text/html » encodé en UTF-8, le plus standard actuellement et évitant les problèmes avec les caractères accentués. Nous ne devrons pas nous occuper de donner leur équivalent en entités HTML. On ajoute donc la ligne suivante :

```
<meta http-equiv="Content-Type"
content="text/html;charset=utf-8" />
```

On ajoute ensuite une instruction qui ne sera considérée que par Internet Explorer et qui a pour objectif de le rendre compatible HTML5, les autres navigateurs le supportant déjà très largement. Il s'agit d'un bout de code

javascript qui crée les éléments HTML manquants. Nous pourrons donc les utiliser à souhait. On importe ce fichier JavaScript depuis un dépôt de Google afin d'avoir une version toujours à jour :

```
<!--[if IE]>
<script type="text/javascript"
src="http://html5shiv.googlecode.com/svn/trunk/html5.js">
</script>
<![endif]-->
```

Suivent ensuite les définitions de style. On commence par les règles nécessaires à jQuery. On les importe telles-quelles depuis l'emplacement où elles ont été sauvegardées :

```
<link type="text/css" href="view/css/ui-lightness/jquery-
ui-1.8.11.custom.css" rel="stylesheet" />
```

Viennent alors nos propres règles CSS définies de façon générale. Elles concernent la mise en forme par défaut à accorder à tous les éléments HTML, nos boîtes et widgets. Celle-ci sont introduites entre balises « style ». Voir le code source pour plus de détails.

Les dernières règles de style seront spécifiques à l'interprétation courante. On introduit donc un bloc de données variables qui devra être complété par PHP. Les données seront à aller chercher dans le Converter, utilisé de façon statique. On suppose ici que le Converter a été défini avant l'appel du template et que la conversion a été effectuée au préalable. Le bloc de données est défini de cette façon, toujours entre les balises « style » :

```
<?php echo Converter::getStyle() ?>
```

Suit alors le titre de la fenêtre. Celui-ci correspond au titre de la fenêtre dans la description usiXML. On introduit donc encore un appel au Converter :

```
<title><?php echo Converter::getTitle() ?></title>
```

On doit encore importer la bibliothèque jQuery. Il s'agit de 2 fichiers séparés, l'un pour la bibliothèque de base, l'autre pour la partie interface utilisateur de jQuery :

```
<script type="text/javascript" src="view/js/jquery-
1.5.1.min.js"></script>
```

```
<script type="text/javascript" src="view/js/jquery-ui-
1.8.11.custom.min.js"></script>
```

Enfin pour finir avec la section d'en-tête on introduit nos instructions jQuery issues de la conversion qui seront exécutées dès que la structure HTML aura été chargée par le navigateur :

```
<script type="text/javascript">
<?php echo Converter::getScript() ?>
</script>
```

Il ne reste plus qu'à introduire le corps du document HTML, issu lui aussi de la conversion :

```
<?php echo Converter::getBody() ?>
```

3.2 Template d'accueil de l'interpréteur

Ce template ne sera pas décrit en détail. Les mêmes principes sont d'application que pour le précédent. Mais celui-ci intègre des blocs conditionnels et des boucles. Attardons-nous donc un peu sur ces nouveaux éléments.

Un bloc conditionnel est nécessaire afin d'afficher un message d'erreur sur la page d'accueil si on a tenté d'envoyer un fichier et qu'une erreur s'est produite lors de la tentative de conversion ou lors du transfert vers le dossier de stockage. On utilise pour se faire une simple instruction conditionnelle PHP au sein du template HTML. On ouvre alors un bloc d'instruction à l'aide d'une accolade et on poursuit avec le code HTML qu'on souhaite afficher au cas où la condition est vérifiée. L'accolade fermante de fin de bloc PHP est ensuite ajoutée. C'est une astuce PHP très efficace qui permet de rester dans un cadre de template HTML sans devoir écrire du code HTML à l'intérieur d'instructions PHP. Le schéma de principe est le suivant :

```
<?php if($condition){ ?>
  <p>Contenu à afficher si la condition est vérifiée</p>
<?php } ?>
```

Ce genre de technique doit aussi être utilisée pour mettre en place des boucles notamment afin de lister les fichiers usiXML disponibles sur le serveur. Ceux-ci sont disponibles sous forme de tableau via la classe Lister mais ce tableau doit être parcouru dans le template. On procède de la même façon que pour le bloc

conditionnel mais on utilise une instruction « foreach ». Le schéma de principe est le suivant :

```
<ul>
<?php foreach($filenames as $filename){ ?>
  <li><?php echo $filename ?></li>
<?php } ?>
</ul>
```

Ce template comprend enfin un formulaire destiné au téléchargement de fichiers vers le serveur. Celui-ci est défini de la façon suivante :

```
<form method="post" action="." enctype="multipart/form-
data">
```

Cette ligne signifie que le formulaire sera envoyé via une requête HTTP de type « POST »[23]. C'est indispensable pour l'envoi de fichiers, le mode « GET » ne convient pas pour cela. « Action » spécifie à quelle ressource côté serveur le formulaire doit être envoyé. Par un point on indique ici que le formulaire doit être envoyé à la ressource actuelle. Ce sera le présentateur, point d'entrée unique de notre application. Enfin « enctype » défini le mode d'encodage des données à transmettre. « multipart/format-data » est le mode préconisé par le W3C lorsqu'il y a des fichiers à transmettre, des données non-ASCII ou binaires.

On introduit dans ce formulaire un champ « input » de type « file » qui permettra de spécifier le fichier à télécharger. Le nom donné au champ doit correspondre au nom attendu par notre classe « Transfer ». Un bouton de type « submit » permettra finalement d'envoyer le formulaire selon les dispositions prévues.

[23] Voir Partie 1 chapitre 4.4, PHP

4 Détails du Présentateur

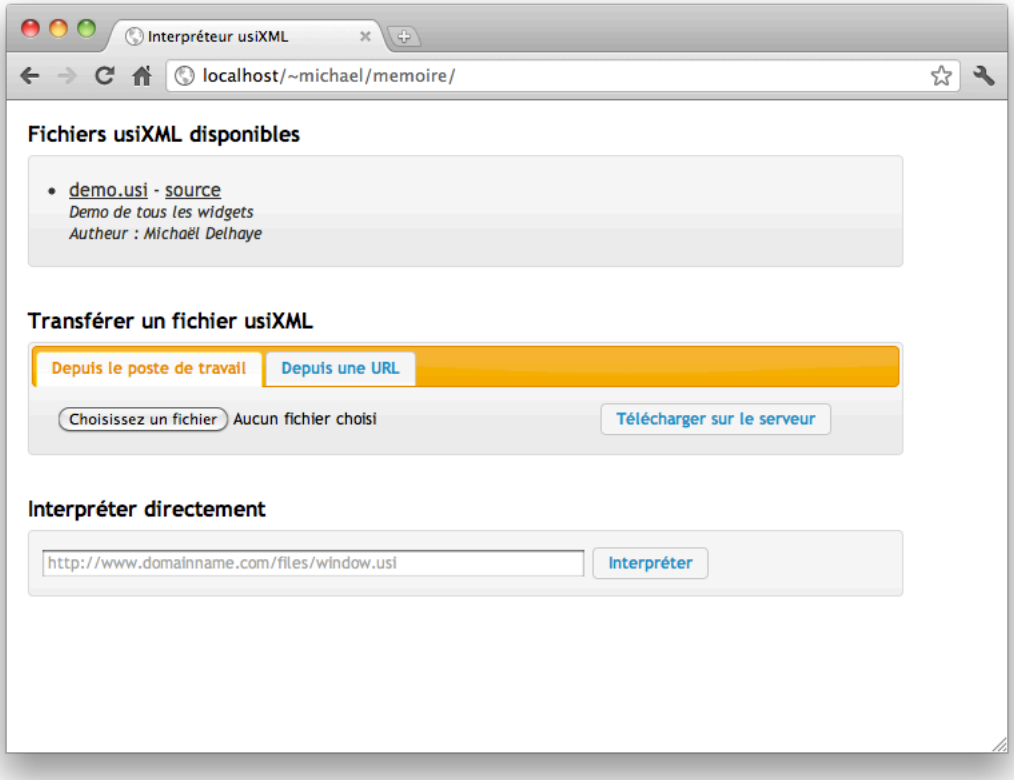
Le présentateur est l'élément central de notre petite architecture MVP. Du point de vue du serveur c'est le point d'entrée principal de l'application. C'est lui qui s'occupe de charger les autres composants. Il existe pour cela l'instruction « require() » de PHP. Celle-ci a pour effet à l'exécution de se remplacer elle-même par le contenu du fichier spécifié en paramètre. Elle fourni donc un moyen simple de découper une application en différents morceaux sans contrainte spécifique.

Il faut cependant être conscient de quelques conséquences de cette inclusion simpliste. Tout d'abord les références relatives au système de fichier (par rapport au dossier contenant le fichier) sont gardées intactes. Ce qui signifie que si on importe un fichier source présent à un autre niveau du système de fichier, un autre dossier ou sous-dossier, toute référence relative au système de fichier dans ce fichier source ne sera plus correcte. Les chemins relatifs sont malgré tout préférables car ils permettent une portabilité de l'application et évitent de dépendre de la structure de fichiers du serveur. La solution choisie dans notre cas sera de spécifier les chemins de tous les fichiers à inclure de façon relative mais par rapport au script principal de l'application, c'est-à-dire le présentateur ici. En effet puisque tout le code sera introduit à ce niveau à l'exécution, les références seront correctes. Mais il faut dès lors être conscient que les autres fichiers source ne peuvent pas être utilisés tels quels de façon indépendante dans d'autres contextes.

Un autre élément à prendre en compte est la possibilité de référence aux mêmes fichiers à inclure à plusieurs endroits dans l'application. Et c'est le cas ici. Si on inclut plusieurs fois la même classe il y aura une redéfinition de classe ce qui en plus d'être inutile n'est pas autorisé par PHP. La solution à ce problème n'est pas difficile à trouver car il suffit d'utiliser l'instruction « require_once() » au lieu de « require() ». Cette instruction s'assure que le fichier à inclure n'a pas déjà été inclus précédemment et ne l'inclut qu'une seule fois. C'est donc cette version qui sera utilisée.

Le présentateur gère toute la logique de l'application. Il doit donc gérer les différents cas de figure pouvant se présenter. Il y en a 3. L'utilisateur peut d'abord demander le téléchargement d'un fichier vers l'emplacement de stockage du serveur. Il peut le faire de 2 façons : par envoi du fichier depuis son ordinateur ou en précisant une URL où le fichier peut être récupéré. L'autre action explicite possible est la demande d'interprétation d'un fichier particulier. Cela peut être un fichier présent sur le serveur ou un fichier distant. Enfin il peut

n'y avoir aucune action particulière. Dans ce cas c'est la page d'accueil de l'application qui doit être présentée avec la liste des fichiers présents sur le serveur et les formulaires utiles aux fonctionnalités précédentes. Chaque fois on aura l'appel à une de nos classes statiques du modèle, suivi de l'invocation d'un template qui en utilisera les données.



Page d'accueil de l'application

Si l'utilisateur demande le téléchargement d'un fichier vers le serveur au départ de son ordinateur, son navigateur transmet le fichier au serveur dans un dossier temporaire dans un premier temps. Le présentateur a la possibilité de détecter cet événement en consultant le tableau global $_FILES. S'il demande le téléchargement depuis une URL, le formulaire utilisé aura transmis celle-ci dans une variable « usi_http » en mode POST, le présentateur a donc la possibilité de le détecter en vérifiant si l'entrée « $_POST['usi_http'] » est définie. Si l'un de ces 2 cas ce produit, le présentateur effectue le branchement vers la méthode de la classe « Transfer » qui se chargera du traitement. Une fois le fichier vérifié on affiche la liste mise à jour des fichiers disponibles à l'interprétation en invoquant le template principal.

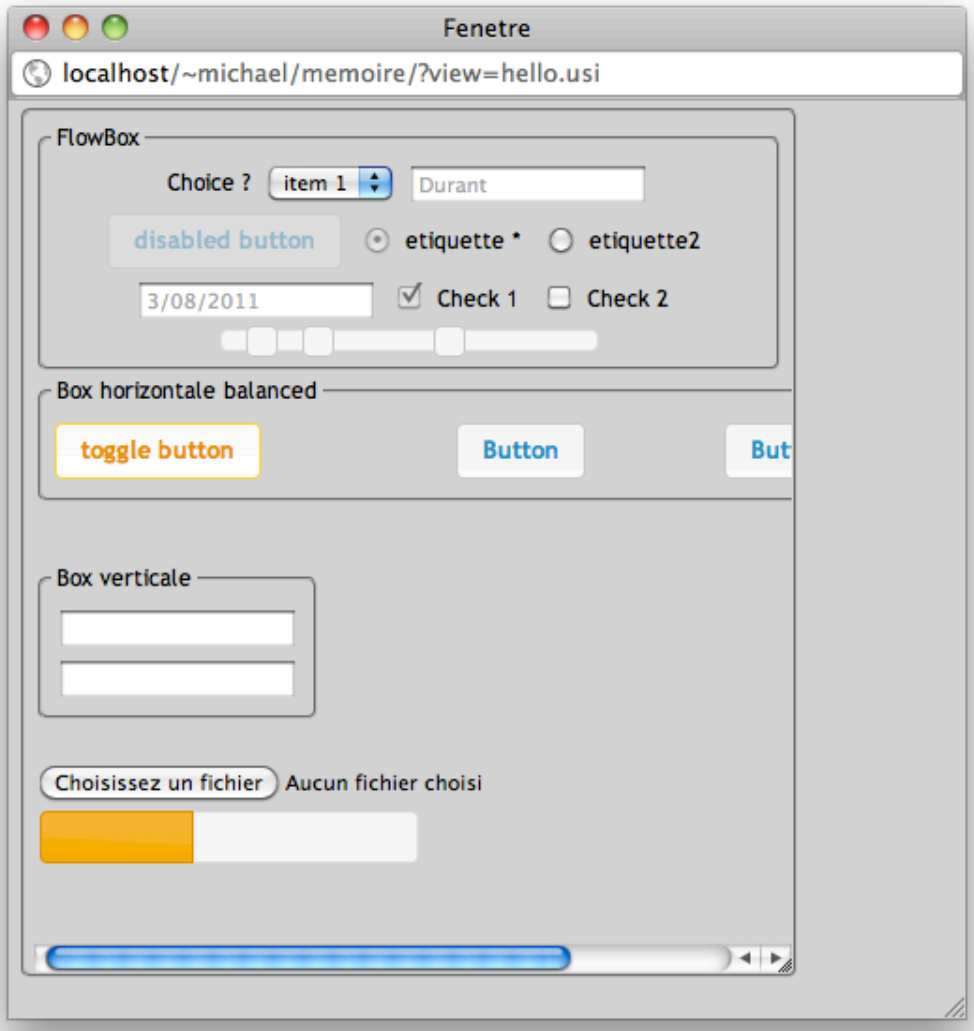
L'utilisateur peut aussi demander l'interprétation d'un fichier usiXML présent sur le serveur. Il est prévu dans la vue principale de l'application listant les

fichiers disponibles qu'un clic sur un lien représentant un fichier usiXML provoque un appel au présentateur avec une variable « view » passée en mode « GET » via la barre d'adresse du navigateur. Cette variable contient le nom du fichier à interpréter. Il suffit encore une fois de vérifier si cette variable est définie et de faire appel au convertisseur et au template adéquat.

Si l'utilisateur demande l'interprétation d'un fichier distant à l'aide du formulaire prévu à cet effet, le même mécanisme entre en jeu. On a en effet pris le soin d'envoyer ce formulaire en mode « GET » et avec la même variable « view ». Le présentateur n'a qu'à vérifier si cette variable contient une URL complète ou le nom d'un fichier et l'ouvrir de la façon appropriée avant de faire appel au convertisseur.

Si aucun de ces cas de figure ne se produit, on est dans la situation par défaut. On doit alors simplement afficher la vue par défaut qui est la liste des fichiers présents sur le serveur et le formulaire permettant d'en soumettre d'autres. Le présentateur fait pour cela préalablement appel au Lister afin de générer les données utiles au template.

5 Quelques aspects de la conversion



Un exemple de fenêtre interprétée

5.1 Identifiants

Pour les règles de style CSS ou pour jQuery c'est l'identifiant, attribut « id » HTML, d'un élément qui est utilisé afin de cibler un élément particulier pour lui attribuer des définitions de style ou lui invoquer des méthodes JavaScript. Or il se peut que la description usiXML ne comporte pas d'identifiant pour tous les éléments. Il est donc nécessaire de générer des identifiants pour ces éléments et ces identifiant doivent être uniques ! C'est dans la méthode de conversion de la classe Converter que c'est aspect doit être considéré. La façon le plus simple de faire est d'utiliser un compteur qui est passé comme argument aux constructeurs des objets. On incrémente alors ce compteur à chaque appel de constructeur et

ceux-ci pourront alors l'utiliser afin d'attribuer un identifiant à l'objet au cas où il n'aurait pas été spécifié dans la description. Mais étant donné que la méthode de conversion est appelée récursivement il faut s'assurer qu'au retour de la récursion le compteur poursuive la numérotation et qu'il ne recommence pas là où il en était avant d'effectuer l'appel récursif. C'est pour cette raison que ce compteur doit être une variable de classe.

5.2 Détails de conversion

La conversion est le résultat d'une combinaison subtile et spécifique à chaque élément des composantes HTML, CSS et jQuery. Peut-on détailler cela ?

Les structures HTML à mettre en œuvre sont parfois composées de plusieurs éléments imbriqués. Pour une case à cocher on a par exemple besoin d'un élément pour la case elle-même, d'un autre pour le libellé, et d'un conteneur englobant le tout. On veut en effet toujours matérialiser un élément usiXML par un seul élément HTML afin de pouvoir gérer correctement le positionnement.

Les règles CSS générées seront elles spécifiques à un ou plusieurs de ces éléments HTML. A celles-ci il faut ajouter les règles définies globalement au sein du template et applicables à tous les éléments du même type. Cela se fait grâce à des sélecteurs CSS basés sur le nom des balises en combinaison avec des classes définies sur les éléments HTML. Il est parfois nécessaire aussi d'impliquer des sélecteurs ciblant un enfant direct d'un élément. C'est le cas par exemple pour donner un style spécifique au contenu d'une boîte sans affecter aussi le contenu des boîtes inclues.

Enfin les instructions de conversion jQuery avec leurs options porteront sur un ou plusieurs de ces éléments HTML également. De plus ce code JavaScript induira à l'interprétation par le navigateur une modification de la structure HTML et l'introduction de règles CSS spécifiques.

On voit qu'il serait très hasardeux de se lancer dans une table de conversion attribut par attribut des éléments usiXML avec leur équivalent dans les 3 composantes. On s'en tiendra donc à un exemple intéressant. Pour les autres éléments un tableau de support est disponible au chapitre 7.

5.3 Exemple de conversion de l'élément « slider »

Widget « slider »

Détaillons la conversion de l'élément « slider ». Celle-ci fait intervenir également la conversion des éléments « cursor » qui y sont contenus. Partons pour ce faire du schéma usiXML suivant :

```
<slider id="slider_id" name="slider_name" minValue="10"
maxValue="50" step="2" isVisible="true">
  <cursor defaultPosition="20" />
  <cursor defaultPosition="25" />
</slider>
```

La structure HTML que nous devons générer est ici très simple. Elle se limite à un élément, avec un attribut de classe fixé ainsi que les attributs « id » et « name ». Voici donc la structure HTML de notre exemple :

```
<div class="slider" id="slider_id"
name="slider_name"></div>
```

On fait appel à jQuery pour convertir ceci en un véritable slider avec des curseurs. Le code jQuery pour ce faire est :

```
$('#%id%').slider();
```

« #slider_id » est le sélecteur CSS qui permettra à jQuery de localiser l'élément div dans le document HTML entier afin d'en faire un objet jQuery. On applique alors sur cet objet la méthode « slider() » qui modifie le code HTML selon ce schéma.

```
<div class="ui-slider ui-slider-horizontal ui-widget ui-
widget-content ui-corner-all">
  <a style="left: 0%;" class="ui-slider-handle ui-state-
default ui-corner-all" href="#"></a>
</div>
```

Il s'agit du schéma générique. Les 3 attributs HTML de départ sont en réalité conservés. L'élément « a » inclus correspond au curseur, il y en aura autant qu'il y a de curseurs et la règle de style CSS « left :0% » sera adaptée en fonction des

positions initiales spécifiées pour les curseurs. La classe ui-slider-horizontal donne elle un rendu horizontal au slider, l'équivalent vertical est possible également. Ces paramètres doivent être précisés en argument de la méthode « slider » sous la forme d'une liste entre accolades de paires clé-valeur séparées par des virgules. Les attributs usiXML suivants sont pris en charge de cette façon :

Définition de l'attribut-option	Nom de l'attribut usiXML	Nom de l'option jQuery
Nombre d'unités séparant 2 positions successives de curseur	step (entier)	step (entier)
Valeur minimale du slider	minValue (entier)	min (entier)
Valeur maximale du slider	maxValue (entier)	max (entier)
Orientation du slider	orientation ('horizontal' ou 'vertical')	orientation ('horizontal' ou 'vertical')
Positions initiales des curseurs	defaultPosition des éléments enfants « cursor » (entier)	values (liste des valeurs séparées par des virgules et entre crochets)

Voici l'appel jQuery complet définissant notre slider horizontal avec un step de 2, un minimum et un maximum définis respectivement à 10 et 50 et 2 curseurs l'un initialement à 20 et l'autre à 25 :

```
$('#%slider_id%').slider({step:2,min:10,max:50,orientatio
n:'horizontal',values:[20,25]);
```

Concernant la composante CSS il faut considérer les règles qui seront d'application pour les classes jQuery « ui-slider », « ui-slider-horizontal », « ui-widget ui-widget-content », « ui-corner-all », « ui-slider-handle » et « ui-state-default ». Elles sont définies dans le fichier CSS utilisé par jQuery, nous ne les détaillons pas ici. De plus d'autres classes peuvent être définies de façon dynamique par jQuery lors de l'utilisation du widget, ce qui fait intervenir encore d'autres règles de style.

A ceci il faut ajouter la ou les règles d'application pour la classe « slider » que nous avons ajoutée nous-même. Une règle avec 2 déclarations définies au sein du template et d'application pour tous les widgets intervient donc :

```
.box,.flowBox, .space, .inputText, .outputText, .button,
.radioButton, .radioButton>label, .radioButton>input,
.checkBox, .checkBox>label, .checkBox>input, .slider,
.comboBox, .datePicker, .filePicker, .progressionBar,
.rating {
  display:inline-block;
  vertical-align:middle;
}
```

Enfin il faut ajouter les déclarations que nous définissons spécifiquement pour ce widget. On utilise pour cela un sélecteur CSS basé aussi sur l'identifiant.
L'attribut usiXML de visibilité est géré ici. Voici la conversion :

usiXML	CSS
isVisible="false"	visibility:hidden
isVisibile="true"	visibility:visible (implicitement car valeur par défaut)

6 Conversion des conteneurs

La diversité des conteneurs, leurs nombreuses options, et le rôle de premier ordre qu'ils jouent pour la présentation font qu'ils méritent qu'on s'y attarde.

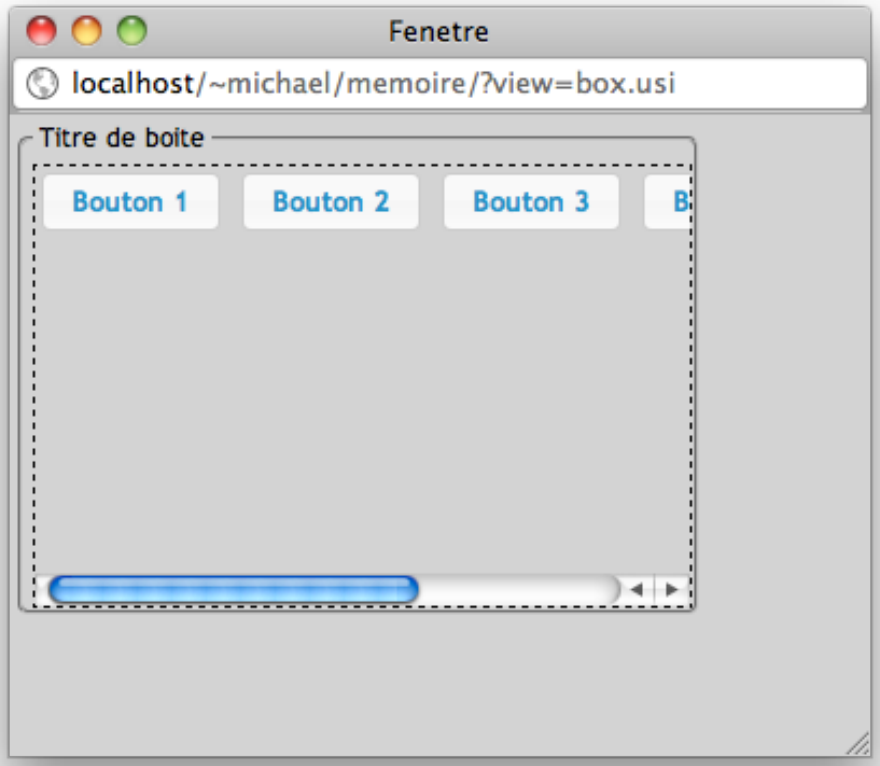
6.1 Conversion des « box »

La conversion des boîtes de base (« box ») nécessite réflexion car plusieurs exigences apparaissent et il faut s'assurer de bien poser le problème afin de ne pas introduire de contradictions. (1) Une boîte peut avoir des dimensions définies ou pas. (2) L'attribut « isScrollable » indique si la boîte doit oui ou non afficher des barres de défilement au cas où le contenu est plus grand que la taille de la boîte. (3) La boîte peut être redimensionnable en fonction de l'attribut « isResizable ». (4) Une bordure peut être définie autour de la boîte par l'attribut « borderWidth ». (5) Enfin un titre devant apparaître dans la bordure peut être prévu par l'attribut « borderTitle ».

L'élément HTML qui correspond le plus à ce type d'utilisation est « fieldset ». II s'agit du seul conteneur adapté aux formulaires permettant d'afficher un titre à l'intérieur de la bordure.

Le premier problème qui se pose concerne le sort qu'il faut réserver aux boîtes dont les dimensions ne sont pas définies. Il semble le plus logique de faire en sorte que leur taille s'adapte à leur contenu. Le point de vue opposé mais moins pertinent est d'adapter leur taille à leur conteneur.

Vient ensuite le problème des barres de défilement. L'élément « fieldset » html n'est pas prévu pour les utiliser. Il est possible de les imposer à l'aide d'une règle CSS mais les barres apparaissent alors au centre de la bordure et le titre de la boîte est impacté par le défilement, ce n'est pas ce qu'on souhaite. Il est donc nécessaire d'ajouter un conteneur « div » à l'intérieur du fieldset. C'est sur celui-ci qu'on pourra appliquer des barres de défilement. Il se repose alors encore une fois le problème des dimensions car si les dimensions sont appliquées uniquement sur le « fieldset » le conteneur « div » inclus aura la possibilité de dépasser (« overflow ») du « fieldset » sans que des barres de défilement apparaissent. Afin de ne pas devoir maintenir les dimensions des 2 conteneurs, qui sont liées par les espacements définis entre-eux et la taille des bordures, on préfère donner les dimensions au conteneur « div » inclus uniquement. Le conteneur englobant « fieldset » s'adapte donc toujours aux dimensions du premier.



Exemple de boîte « box » avec taille définie. Les pointillés représentent le conteneur « div » inclus.

Troisièmement, comment rendre les boîtes redimensionnables ? Ce n'est pas possible en HTML ni en CSS. On devra donc faire appel à jQuery qui prévoit cela. Il suffit comme toujours en jQuery d'appeler une méthode, ici « resizable() », sur l'objet JavaScript représentant l'élément HTML que l'on veut impacter. Cette méthode fera en sorte d'afficher des poignées sur les bords de l'élément redimensionnable et il sera possible à la souris d'en changer la taille. Idéalement c'est ici le « fieldset » que l'on veut redimensionner. Or c'est au conteneur « div » inclus que l'on a attribué des dimensions. On est ici un peu perfectionniste mais c'est possible à l'aide de jQuery ! Il est prévu de rendre une boîte redimensionnable et d'indiquer en option (« alsoResize ») quels autres éléments doivent être redimensionnés dans la foulée à l'aide d'un sélecteur CSS. (alsoResize : #id_fieldset>.boxcontent) usiXML fait une distinction entre le redimensionnement horizontal et vertical qui peuvent être autorisés séparément. En jQuery ce n'est pas prévu. On utilisera dès lors les options de dimensions minimales et maximales et on les fixera à la hauteur ou la largeur de départ de la boîte. Ces dimensions devront être calculées une fois toute l'interface interprétée par le moteur de rendu du navigateur étant donné que

pour les éléments sans dimensions on lui laisse le soin de leur donner la taille adéquate en fonction du contenu.

usiXML permet également de spécifier des tailles minimales. Celles-ci sont spécifiées en pourcentage de la taille originale. On utilise donc ici aussi les options de dimensions minimales de jQuery mais un petit calcul sera nécessaire afin d'obtenir la taille minimale en pixel à fournir. Il suffit de multiplier le pourcentage par la taille de départ qui aura été calculée.

Quatrièmement vient la possibilité de définir une bordure autour de la boîte. C'est ce que prévoit par défaut l'élément HTML « fieldset ». Ici on voudrait plutôt que par défaut il n'y ait pas de bordure, les boîtes étant principalement utilisées afin d'organiser la mise en page des widgets. Une règle CSS mettant la bordure à 0 par défaut fait l'affaire. On pourra indiquer une autre valeur si cela est spécifié autrement dans la description usiXML. Mais attention, au cas où la boîte est redimensionnable, il est bon d'imposer une bordure minimale aux boîtes afin de permettre de visualiser où positionner la souris pour la redimensionner !

Enfin dernièrement, il est prévu de pouvoir indiquer un titre apparaissant au sein de la bordure. Ici aussi l'élément « fieldset » répond à ce besoin, c'est pour cela qu'on l'a choisi. Il faut néanmoins s'assurer que ce titre apparaisse uniquement dans le cas où une taille de bordure supérieure à 0 à été indiquée. Ca n'aura pas de sens sinon.

Pour ce qui concerne les « flowBox » le même schéma HTML est utilisé. Mais la conversion est plus simple étant donné qu'il n'est pas prévu d'y afficher des barres de défilement ni de permettre le redimensionnement. Il n'y a donc pas de difficulté particulière

6.2 Box horizontale vs verticale

Un aspect important de la conversion concerne le type des boîtes de base (« box »). Elles peuvent être de type horizontal ou vertical. Cet attribut n'affecte pas l'aspect de la boîte elle-même mais la façon dont sera effectué le rendu des éléments contenus. Dans une boîte de type horizontal les éléments s'affichent les uns à la suite des autres de façon horizontale. Dans une boîte de type vertical c'est bien sûr la même chose mais verticalement. Cela paraît simpliste énoncé

comme cela mais en réalité cela ne correspond à aucun comportement par défaut en formatage HTML[24]

Pour le rendu horizontal on s'approche de ce que prévoit le mode « inline » à la différence qu'il ne faut jamais effectuer de retour à la ligne, ce qui est pourtant prévu par défaut pour ce mode de rendu. De plus si on peut envisager que les widgets prennent la forme d'éléments inline, ce n'est pas le cas pour les boîtes qui ont généralement des dimensions. En effet, pour rappel on ne peut pas donner de dimensions à des éléments inline. Et de toute façon les boîtes sont en HTML matérialisées par des éléments « div », qui par défaut sont rendus en mode bloc, càd affichés les uns en dessous des autres et sur toute la largeur de leur conteneur si aucune dimension n'est précisée.

On voit qu'un mix des 2 modes de rendu va devoir être utilisé. Ce mode s'appelle « inline-block ». Les éléments sont affichés en ligne tout en permettant de leur donner des dimensions. Il est aussi prévu de pouvoir aligner les éléments verticalement et horizontalement par rapport au conteneur, ce qui sera utile. On verra par la suite comment cela fonctionne. Pour ce qui concerne le rendu du contenu des boîtes de type horizontal on a donc compris qu'il faut modifier le mode d'affichage par défaut de tous les éléments contenus en « inline-block ». Cela se fait grâce à la propriété « display » de CSS. Il reste que des retours à la ligne sont effectués par défaut. Ils sont utiles pour les « flowBox » mais en toute généralité on ne les souhaite pas. C'est ici la propriété « white-space » de CSS qui nous vient en aide. En lui donnant la valeur « nowrap » les retours à la ligne sont désactivés.

Pour les boîtes de type vertical on aurait tendance à vouloir utiliser le mode d'affichage « block » qui correspond de premier abord à ce que l'on souhaite. Le problème est que ce mode, on l'a vu, définit une largeur implicite aux éléments correspondant à largeur du conteneur si aucune autre largeur n'est spécifiée. Or il existe des éléments dont on ne connaîtra pas la taille et on veut pourtant qu'ils ne prennent que la place qui leur est nécessaire. Ces dimensions doivent être calculées automatiquement par le moteur de rendu. C'est le fonctionnement du mode « inline ». La solution est donc ici encore le mode intermédiaire « inline-block ». Afin que les éléments soient affichés les uns en dessous des autres on doit simplement ajouter des retours à la ligne manuels entre ceux-ci au sein du code HTML grâce à la balise «
 ». On ne peut pas le faire uniquement à l'aide de règles CSS.

---

[24] Voir Partie 1 chapitre 4.3, éléments de formatage HTML & CSS

Cela facilite les choses en réalité car il suffit de demander le rendu en « inline-block » de tous les éléments. Le code HTML ou CSS des éléments inclus dans un conteneur ne dépend pas du type du conteneur. Simplement au cas où la boîte est de type vertical il faut au sein de la génération du code html du conteneur ajouter une balise «
 » entre le code html de chaque élément inclus.

Il y a cependant une exception, c'est le cas du « spacer ». Cet élément permet de définir un espace entre 2 autres éléments. Mais cet espace est défini par une seule valeur et cette valeur doit être comprise comme un espacement vertical dans le cas où il est inclus dans une boîte verticale et comme un espacement horizontal dans le cas où il est contenu dans une boîte horizontale. Dans le cas du spacer on doit donc accéder à son élément parent, cela avait été prévu au niveau de la classe de base « cio », et s'il s'agit d'une boîte tenir compte de son type pour générer le code HTML et CSS du spacer. Un espacement horizontal sera un bloc div de la largeur correspondant à l'espacement voulu et d'une hauteur d'un pixel et l'inverse pour un espacement vertical, tous deux étant rendus en « inline-block » comme les autres éléments.

Pour ce qui concerne l'alignement, rien n'est précisé dans la spécification. Les éléments ont donc été alignés horizontalement à gauche et verticalement en haut. Les éléments ont été alignés entre-eux verticalement au milieu, c'est ce qui est le plus esthétique. Mais ceci n'est pas pris en charge au niveau de la boîte mais au niveau des éléments individuellement par la propriété CSS « vertical-align:middle ».

6.3 Box balanced

« isBalanced » est une autre option (attribut) des boîtes de base. Celle-ci indique que les éléments contenus doivent être répartis de façon équilibrée au sein du conteneur afin de le remplir entièrement. Si la boîte est de type horizontal des espaces horizontaux devront être insérés entre les éléments de telle sorte que tout l'espace horizontal soit occupé et que les éléments soient disposés de manière uniforme, et réciproquement pour une boîte de type vertical. La meilleure solution trouvée consiste dans ce cas à créer un tableau HTML de la taille de la boîte sans bordures ni espacement, de créer autant de cellules que d'éléments à insérer et d'y disposer ceux-ci. En effet si on donne une dimension au tableau supérieure à ce qui est nécessaire pour afficher son contenu et qu'on ne donne pas de dimensions aux cellules les moteurs de rendu donneront généralement des tailles aux cellules proportionnelles à leur contenu. En centrant les éléments dans les cellules, cela correspond approximativement à ce

que l'on souhaite. Les espaces entre les éléments ne seront pas parfaitement identiques mais il sera difficile à l'œil de faire la différence.

6.4 flowBox

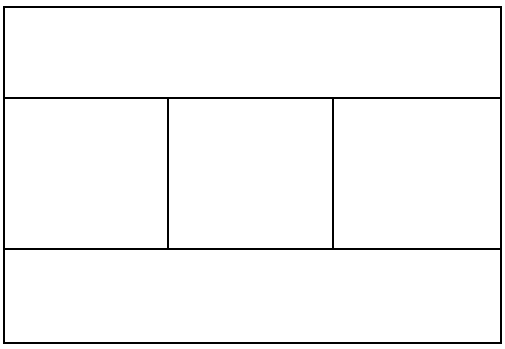
Une boîte de type « flowBox » est une boîte simple où les éléments contenus sont disposés horizontalement et où un passage à la ligne suivante est effectué automatiquement afin de ne pas déborder du conteneur. Les barres de défilement ne sont donc généralement pas nécessaire et ne sont pas prévues dans les spécifications. Le redimensionnement et l'option « isBalanced » ne sont pas d'application ici non plus.

Ces boîtes sont donc implémentées comme des « box » de type horizontal sans barres de défilement, sans redimensionnement possible et en gardant le mode par défaut pour les retours à la ligne : « white-space:normal » contrairement aux « box » normales. Si aucune largeur n'est donnée aux « flowBox » elles éviteront toujours si possible de déborder de leur conteneur parent en faisant passer à la ligne ses éléments inclus.

Les flowBox prévoient un attribut supplémentaire, il s'agit de l'alignement horizontal au sein de la boîte. Il peut être défini à gauche, à droite ou au centre. Cela est simplement pris en charge par l'attribut « text-align » de CSS.

6.5 Mise en page à l'aide de BorderBox

Un « borderBox » est une boîte qui permet d'organiser la mise en page (layout) en 5 régions : haut, bas, gauche, droite et centre qui sont elles-mêmes des boîtes.



Structure d'un BorderBox

La façon la plus simple d'implémenter une telle structure en HTML est d'utiliser un tableau. On voit que la cellule du haut et celle du bas occupent une largeur de 3 colonnes. C'est l'attribut « colspan » de l'élément HTML « TD » représentant ces

cellules qui permet d'obtenir un tel résultat. En lui donnant la valeur « 3 » on indique que la cellule est le résultat de la fusion de 3 cellules.

Mais toutes les zones d'un « borderBox » ne seront pas toujours définies. Il faut donc générer les cellules uniquement si un contenu leur a été attribué et donner la bonne valeur aux attributs « colspan » des cellules du haut et du bas en fonction de la présence de la cellule gauche, centre et droite. Aussi, étant donné que les cellules sont comprises dans des rangées, « TR » en HTML, il faut générer seulement les lignes qui comprennent au moins une cellule.

Enfin chaque cellule-boîte est prévue pour contenir un type de boîte plus spécifique. On fait donc appel en cascade aux méthodes qui fournissent leur représentation afin de les inclure. Selon la spécification usiXML, il ne peut normalement pas se trouver de widget héritant de « TwoDIndividualComponent » directement dans une boîte de « borderBox ».

7 Elements et attributs supportés

Après ces explications techniques concernant l'implémentation, précisons quels sont les éléments usiXML pris en charge. Posons déjà que l'attribut id sera toujours reproduit comme attribut de la balise html principale représentant le widget et que l'attribut isVisible="false" sera traduit par la règle CSS « visibility : none » sur l'élément html principal également. Voici les autres éléments et attributs supportés

7.1 window

- ✓ bgColor
- ✓ fgColor
- ✓ borderTitle
- ✓ width
- ✓ height
- ✓ windowLeftMargin
- ✓ windowTopMargin
- ✓ isResizable (pas supporté par Google Chrome pour raisons d'accessibilité)

7.2 box

- ✓ bgColor
- ✓ fgColor
- ✓ borderTitle
- ✓ borderWidth
- ✓ width
- ✓ height
- ✓ type
- ✓ isScrollable
- ✓ isResizableHorizontal
- ✓ isResizableVertical
- ✓ relativeMinWidth
- ✓ relativeMinHeight
- ✓ isBalanced

7.3 topBox, leftBox, centerBox, rightBox, bottomBox

7.4 flowBox

- ✓ bgColor
- ✓ fgColor
- ✓ borderTitle
- ✓ borderWidth
- ✓ width
- ✓ height
- ✓ align (left, right, center)

7.5 space

- ✓ value : seulement valeur en pixels

7.6 inputText

- ✓ name
- ✓ help
- ✓ defaultContent
- ✓ isMandatory : uniquement du point de vue HTML, pas visible
- ✓ isEnabled
- ✓ bgColor
- ✓ isBold
- ✓ isItalic
- ✓ isUnderlined
- ✓ textColor
- ✓ isEditable
- ✓ isPassword

7.7 outputText

- ✓ defaultContent
- ✓ isEnabled
- ✓ bgColor
- ✓ isBold
- ✓ isItalic
- ✓ textColor
- ✓ isUnderlined

7.8 button

- ✓ name
- ✓ defaultContent
- ✓ isEnabled

7.9 toggleButton

- ✓ name
- ✓ defaultContent
- ✓ isEnabled
- ✓ defaultState

7.10 radioButton

- ✓ name
- ✓ isMandatory
- ✓ defaultContent
- ✓ isEnabled
- ✓ bgColor
- ✓ isBold
- ✓ isItalic
- ✓ isUnderlined
- ✓ textColor
- ✓ groupName
- ✓ defaultState

7.11 checkBox

- ✓ name
- ✓ isMandatory
- ✓ defaultContent
- ✓ isEnabled
- ✓ bgColor
- ✓ isBold
- ✓ isItalic
- ✓ isUnderlined
- ✓ textColor
- ✓ groupName
- ✓ defaultState

7.12 slider

- ✓ defaultContent
- ✓ isEnabled
- ✓ minValue
- ✓ maxValue
- ✓ step
- ✓ orientation

7.13 cursor

- ✓ defaultPosition

7.14 comboBox

- ✓ name
- ✓ defaultContent
- ✓ isEnabled
- ✓ bgColor
- ✓ isUnderlined
- ✓ maxVisible (pas supporté par Google Chrome et Safari, 1 élément ou minimum 4)

7.15 datePicker

- ✓ name
- ✓ defaultContent
- ✓ isEnabled
- ✓ bgColor
- ✓ isBold
- ✓ isItalic
- ✓ isUnderlined
- ✓ textColor

7.16 filePicker

- ✓ name
- ✓ defaultContent
- ✓ isEnabled
- ✓ bgColor
- ✓ isBold
- ✓ isItalic
- ✓ isUnderlined
- ✓ textColor

7.17 progressionBar

- ✓ defaultContent
- ✓ minValue
- ✓ maxValue

8 Nouvel élément : rating

Un nouvel élément usiXML a été imaginé et implémenté. Il s'agit d'un widget permettant de mettre une note à l'aide d'étoiles. On lui a donné le nom « rating ». Il hérite de « 2DgraphicalIndividualComponent » et a 2 attributs supplémentaires. « number » permet de définir le nombre d'étoiles à afficher et « isCancelable » indique si l'annulation est possible à l'aide d'un bouton supplémentaire.

Widget « rating »

Les attributs pris en charge sont les suivants :

- ✓ name
- ✓ defaultContent
- ✓ isEnabled
- ✓ bgColor
- ✓ isCancelable
- ✓ number

9 Evolutions envisageables

Suite à ce travail, quelles sont encore les possibilités d'évolution ?

Même si l'implémentation couvre les éléments d'interface graphique en deux dimensions les plus courants, il reste un certain nombre d'éléments et attributs non supportés. Il y a encore du travail afin d'obtenir un support complet et rigoureux de tous ceux-ci. En effet de par l'héritage de classe, on a par exemple que tout widget héritant de « 2DgraphicalIndivdualComponent » se voit attribuer au minimum 41 attributs. Les widgets ayant tous une structure HTML différente, parfois complexe, et réagissant différemment aux règles de style, cela impose de considérer chaque attribut individuellement. C'est tout à fait normal étant donné qu'on réalise une conversion d'un langage abstrait indépendant de toute implémentation vers des langages concrets qui ont leurs contraintes et leurs limites. Il faut être conscient que certaines choses ne sont pas possibles. En tout cas il faut prévoir un temps conséquent pour les tests.

Le support du positionnement à l'aide d'une grille doit encore être réalisé. Ceci impose un tout autre mode de rendu avec des positionnements absolus non-abordés ici. On n'est plus dans une logique en flux. HTML n'est pas vraiment prévu pour effectuer une mise en page de cette façon mais c'est envisageable, surtout avec l'aide de jQuery. Par contre un algorithme doit être développé afin de gérer la répartition automatique des éléments en fonction des critères d'alignement prévus dans la description usiXML.

Une version « mobile » de l'interpréteur est envisageable. Mais ce serait tout autre chose. Il s'agirait d'effectuer le rendu d'interfaces réalisées pour des smartphones, sur les navigateurs de smartphones. Il ne s'agit pas d'interpréter d'une autre façon des descriptions d'interfaces desktop. On ne parle plus de fenêtres mais plutôt d'écrans ou de pages. Et les interfaces sont bien sûr adaptées aux petits écrans. Des tailles de boîtes plus grandes que l'écran ou la présence d'ascenseurs ne sont pas possibles… De plus aucun code HTML, CSS ou JavaScript ne peut être réutilisé tel quel, toute la conversion est à refaire pour cette version. Ce sont les raisons pour lesquelles ce travail n'a pas été entamé. Cela peut être envisagé mais en étant conscient qu'il s'agit d'un tout autre contexte, aux limites encore plus marquées.

Par contre on est proche du travail effectué ici de par le fait que les mêmes technologies peuvent être utilisées. HTML5 est bien supporté sur les navigateurs de smartphones et les concepteurs de jQuery en ont prévu une version mobile avec les widgets spécifiques aux plateformes mobiles. Tous les widgets sont ainsi

convertis en versions présentant des contrôles plus grand afin d'améliorer l'ergonomie. Une partie de l'interpréteur développé ici peut être réutilisé pour cette implémentation. Les classes génériques représentant les éléments usiXML peuvent être conservées telles quelles. Il faudrait créer des classes « mobile » de la même façon que les classes « web » mais supportant la conversion en mode mobile. Et de légères modifications devraient être apportées au présentateur pour qu'il gère ce 2ème mode de conversion et fasse en sorte que la classe Converter utilise les nouvelles classes « mobile » pour convertir en mode mobile.

Conclusion

A côté des interpréteurs usiXML préexistants en Java et en Flash, il existe donc maintenant un interpréteur web. Celui-ci a l'avantage de ne nécessiter aucune installation côté client. Côté serveur les prérequis ne sont pour autant pas excessifs puisqu'un simple serveur web avec support de PHP suffit, ce qui est très courant.

Plus important, notre interpréteur permet une interprétation à distance ce qui était l'objectif rappelons-le. Le prototype d'interface réalisé en usiXML n'est dorénavant plus lié à au poste de travail du développeur. Celui-ci peut mettre le prototype en ligne sur un serveur où l'interpréteur est installé et le rendre ainsi accessible au format web à toute personne s'y connectant. Mais si le fichier est déjà accessible sur Internet quelque part, l'interpréteur peut même être utilisé pour interpréter ce fichier à partir de son emplacement actuel, à distance via Internet.

Le développement fut un challenge étant donné que le web n'est pas prévu à la base comme support pour des applications graphiques évoluées. Mais on a vu que l'évolution des technologies du web rend cela possible aujourd'hui si on parvient à les combiner judicieusement. La souplesse apportée par cet environnement se paie malheureusement par le manque de repères afin d'élaborer une application robuste.

On a donc dû s'imposer une certaine rigueur, surtout vu la multitude des éléments et attributs à prendre en charge. Ce travail fut l'occasion aussi de découvrir jQuery et d'approfondir certains aspects du langage PHP et des règles de formatage CSS qui permettent de faire énormément de choses mais nécessitent une certaine maîtrise. Enfin la mise en œuvre de l'architecture MVP, bien que ayant demandé un travail supplémentaire, a permis un découpage logique nécessaire à une plus grande clarté. On a ainsi obtenu une meilleure maitrise de l'ensemble et une meilleure maintenabilité.

Bibliographie

R. N. Taylor, Nenad Medvidović, Eric M. Dashofy (2009). *Software architecture : foundations, theory, and practice.* Wiley, 750 pages

D. Greer (2007). *Interactive Application Architecture Patterns.* http://www.aspiringcraftsman.com/2007/08/25/interactive-application-architecture/ (dernière consultation le 16/08/2011)

http://www.w3.org/TR/CSS2/box.html (dernière consultation le 16/08/2011)

http://www.w3.org/TR/CSS2/visuren.html (dernière consultation le 16/08/2011)

http://www.w3.org/TR/CSS2/visudet.html (dernière consultation le 16/08/2011)

http://openweb.eu.org/articles/cascade_css (dernière consultation le 16/08/2011)

http://jqueryui.com (dernière consultation le 16/08/2011)

http://php.net (dernière consultation le 16/08/2011)

http://www.w3schools.com (dernière consultation le 16/08/2011)

http://www.usixml.org (dernière consultation le 16/08/2011)