Outil de publication pour la production d'environnements d'expérimentation Web

Olivier Granges

SIN
Projet de Master

Janvier 2005

Responsable
Dr. Denis Gillet
EPFL / LA

Assistant
Karim Zeramdini
EPFL / LA

Table des matières

Table des figures

Introduction générale

Le Laboratoire d'Automatique (LA) de l'EPFL, en collaboration avec le Laboratoire d'Informatique Théorique (LITH), a mis en œuvre, depuis l'année 2000, un projet visant à offrir une nouvelle approche pédagogique aux futurs ingénieurs formés à la section de génie mécanique de la faculté des Sciences et Techniques de l'Ingénieur (STI), anciennement Département de Génie Mécanique. Ce projet, nommé *eMersion*, a pour finalité d'offrir, aux étudiants, un environnement d'apprentissage leur permettant d'effectuer différents travaux pratiques par l'intermédiaire du Web. Cet environnement est sensé être déployé et géré par des personnes non expertes en systèmes d'information. Afin de satisfaire ces conditions, plusieurs outils sont mis à leur disposition pour leur faciliter la publication de nouveaux environnements d'expérimentation.

Parmi ceux-ci, un outil auteur a été conçu afin de faciliter la conception de nouveaux environnements [Gil03]. Celui-ci offre cependant une approche assez statique. En effet, par l'intermédiaire d'un navigateur Internet, il consiste à remplir trois formulaires distincts permettant de définir un certain nombre de caractéristiques particulières du nouvel environnement qui seront appliquées sur des modèles conçus d'avance. Ces caractéristiques et ces modèles sont cependant limités, et les possibilités de conception de nouveaux environnements s'en trouvent fortement réduites. De plus, cet outil ne propose pas une prévisualisation graphique de l'aspect du nouvel environnement, tel qu'il le serait une fois déployé.

Afin de combler ces lacunes, un nouvel outil auteur a été conçu. Celui-ci a été développé dans le cadre d'un projet de diplôme précédent et vise à offrir une approche purement graphique. Avec ce nouvel outil, il devient possible de créer un nouvel environnement contenant une multitude de composants multimédias différents. Il a également été prévu de gérer l'aspect comportemental entre les différents composants de cet environnement. Toutefois, ce nouvel outil ne prenait pas encore en charge la génération et la publication d'un nouvel environnement, mais se limitait à leur conception.

Ce travail de Master a donc pour objectif de présenter la façon dont nous avons géré la mise en œuvre d'un outil de publication qui vise à générer et publier sur le Web un nouvel environnement en vue de son exploitation. Le premier chapitre, nous permettra de poser le cadre général du projet, son architecture, ainsi que les différents outils mis à disposition. Dans le second chapitre, nous reviendrons sur l'outil auteur qui nous a été fourni, en présentant les adaptations visant à optimiser la compatibilité avec l'outil de publication qui sera implémenté. Le troisième chapitre nous mènera successivement au travers des différentes étapes aboutissant à la conception de l'outil de publication. Finalement, le quatrième et dernier chapitre nous permettra de présenter différents concepts traitant des possibilités de gestion de la communication entre plusieurs composants.

1 Présentation générale

1.1 Définition du sujet

« Le but de ce projet est de développer un outil de publication (Renderer) permettant de générer des environnements d'expérimentation Web. L'outil de publication doit prendre en entrée un document XML [26] dont la grammaire existe déjà. Cette grammaire définit un ensemble de composants multimédias (image, vidéo, audio, texte structuré, etc.), leur emplacement spatial ainsi que la communication (échange et partage de données) et la relation de comportement entre composants. Ainsi, durant la phase de « rendering » l'outil doit analyser le document XML afin de restituer son contenu sous forme d'un environnement Web interactif.

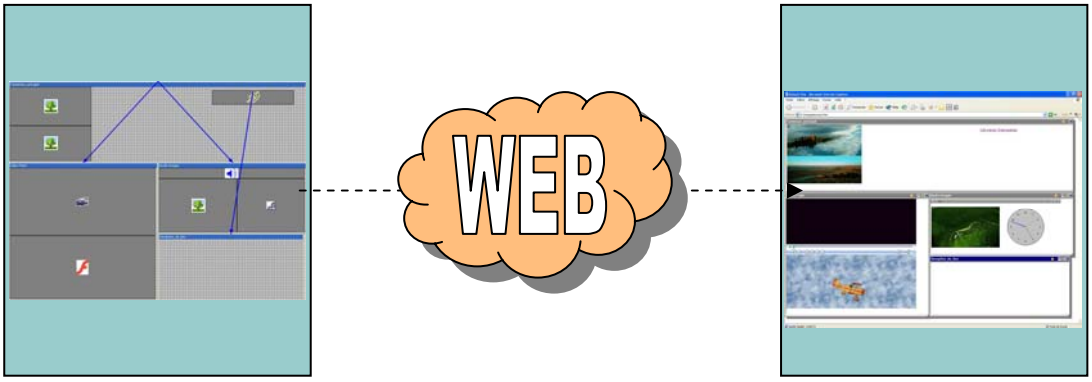
« Cet outil sera intégré à l'environnement eLearning eMersion [23] pour l'apprentissage flexible en automatique. Ce projet est en collaboration entre le laboratoire d'automatique [27] et le laboratoire d'informatique théorique [28].

1.2 Architecture globale

Comme il est décrit dans la définition du sujet ci-dessus, le but final de ce projet revient à publier un environnement Web à partir d'un document XML contenant toutes les informations pertinentes composant cet environnement. Ce fichier XML n'est bien évidemment pas produit à la main par un utilisateur quelconque, mais par un outil spécifique. Cet outil, que nous nommerons dès à présent l'*outil auteur*, consiste en une application permettant à un utilisateur de créer un environnement particulier grâce à un ensemble d'outils graphiques. Cet outil auteur va donc permettre à l'utilisateur de véritablement dessiner l'environnement Web qu'il souhaite créer, tout en spécifiant un certain nombre de contraintes et paramètres particuliers. Toutes ces différentes informations seront contenues dans ce document XML. Ce fichier XML sera donc la description du fonctionnement de l'environnement Web.

La finalité d'une telle opération étant bien entendue la publication de cet environnement, le document XML servira donc de vecteur d'information vers l'outil de publication. Cet outil, que nous nommerons dès à présent le « *renderer* », se trouvera sur un serveur Web distant et devra donc récupérer ce fichier afin de l'analyser et de produire tous les documents nécessaires définis par l'outil auteur dans le fichier XML.

La figure 1-1 représente cette architecture, avec d'un côté la conception d'un nouvel environnement, suivie de sa publication sur le serveur Web.



Côté client:
Production d'un environnement

Côté serveur:
Publication de l'environnement

Figure 1-1 Architecture globale

1.3 Description du fonctionnement

Dans le cadre du renderer, nous utiliserons la dénomination de *cockpit* pour désigner un environnement d'expérimentation Web. Un cockpit pourrait être assimilé à un véritable bureau virtuel. En effet, dans les systèmes d'exploitations actuels (Windows XP, Mac OS X,…) l'utilisateur a à sa disposition un bureau dans lequel il peut agir avec de multiples applications différentes. Ces différentes applications apparaissent dans des fenêtres particulières qui sont indépendantes les unes des autres. L'idée d'un cockpit est assez semblable, à la différence qu'il ne s'agit pas d'un système d'exploitation, mais d'un système de fenêtrage s'exécutant dans un navigateur Internet tel que Internet Explorer, Netscape ou Safari. Par analogie, nous pourrions interpréter un cockpit comme un 'système d'exploitation' s'exécutant sur une plateforme Web virtuelle. Dans le cadre d'un cockpit, nous utilisons le terme de *conteneur* pour définir une fenêtre contenant un composant multimédia. Chaque conteneur pourrait comprendre une application différente, une page Web différente et qui n'auraient pas forcément de rapport direct entre elles. L'utilisateur pourrait influer directement sur chaque conteneur en le déplaçant, en le redimensionnant, en le minimisant ou encore en le fermant. D'autres types d'événements seraient encore possibles, tels que l'ouverture dans un nouveau conteneur du cockpit lors d'un clic sur un lien dynamique situé dans un conteneur différent, ou encore l'échange d'informations entres différents conteneurs par l'établissement d'une communication entre différents composants.

1.3.1 Architecture côté client

L'utilisateur qui souhaite créer un nouvel environnement d'expérimentation Web aura à sa disposition un outil auteur lui permettant de spécifier toutes les caractéristiques de son environnement. Ainsi, lorsque l'utilisateur va créer un nouveau cockpit avec son outil auteur, il aura la possibilité d'insérer des images, des sons, des vidéos, des applets ou encore d'autres pages Web, des liens dynamiques, des services Web, etc. Lorsque tous ces composants seront insérés dans l'outil auteur, ils seront également copiés dans un répertoire spécifique, lié à cet environnement. L'utilisateur pourra également insérer des contraintes comportementales dans son cockpit. Nous pourrions imaginer que l'utilisateur aimerait lier deux conteneurs lorsque ceux-ci seraient déplacés ou encore de fermer plusieurs conteneurs simultanément lorsqu'un conteneur particulier est fermé.

Lorsque toutes les caractéristiques d'un cockpit auront été définies par l'auteur, celui-ci souhaitera le publier sur le serveur. Pour cela, il faudra transmettre les informations et les documents nécessaires à l'outil de publication chargé d'effectuer les opérations de publications sur le serveur. L'idée sera donc de n'envoyer qu'un seul document à l'outil de publication. Ce document sera une archive compressée contenant tous les autres documents du cockpit, dont notamment le fameux fichier XML définissant les caractéristiques du nouveau cockpit.

La figure 1-2 représente l'architecture côté client permettant d'une part la conception de l'environnement, suivie, d'autre part, de la création d'une archive ZIP contenant tous les éléments de cet environnement en vue de son déploiement.

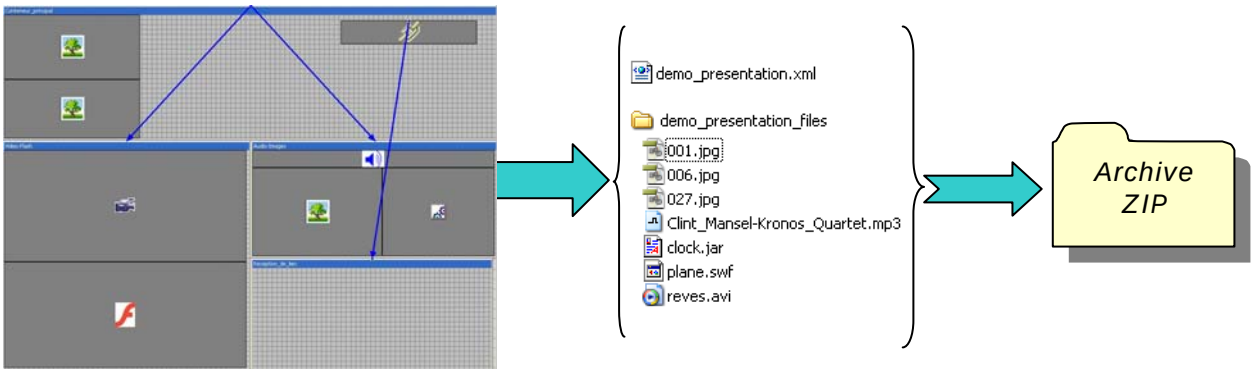


Figure 1-2 Architecture côté client

1.3.2 Architecture côté serveur

La partie s'exécutant sur le serveur à travers l'outil de publication aura pour objectif de générer un espace de travail correspondant le plus fidèlement possible aux caractéristiques définies lors de la création du cockpit avec l'outil auteur. La toute première étape sera donc de charger sur le serveur le document d'archive généré par l'outil auteur. Une fois que ce document se trouvera sur le serveur, il sera possible de le manipuler efficacement. Tout d'abord, il faudra le décompresser et récupérer l'organisation hiérarchique telle qu'elle l'était dans l'outil auteur. Comme expliqué plus haut, le fichier XML décrivant le nouveau cockpit sera véritablement la charnière centrale de tout le processus de publication. Il faudra donc analyser scrupuleusement les données de ce fichier afin d'appliquer toutes les contraintes lors de la génération des documents. Cette génération s'effectuera par une transformation XSLT [32].

Finalement, une fois que tous les documents auront été générés correctement, il faudra publier l'ensemble afin que le cockpit soit accessible depuis n'importe quel navigateur.

La figure 1-3 représente l'architecture côté serveur permettant la décompression de l'archive ZIP, suivie des opérations de transformations pour la publication sur le serveur.

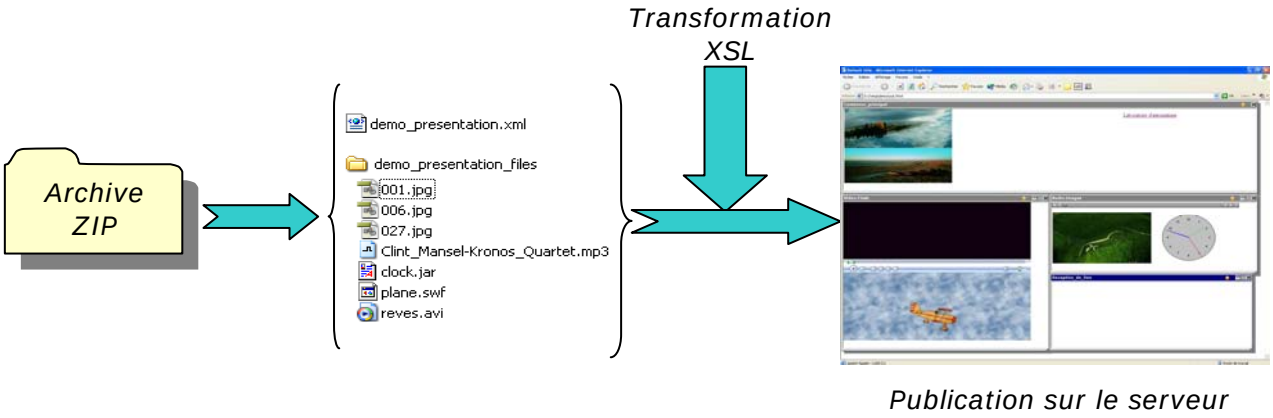


Figure 1-3 Architecture côté serveur

1.4 Outils mis à disposition

Ce projet de Master se déroulant dans le cadre de la mise en œuvre de l'environnement *eMersion*, plusieurs éléments de base ont été fournis afin de mener à bien le développement de ce projet. Nous décrivons ci-dessous les deux projets qui ont été mis à disposition en présentant le cadre de ces projets, ce qu'ils fournissaient et ce qui y manquait.

1.4.1 Outil auteur

Dans le cadre d'un projet de diplôme effectué par S. Pasche en février 2004 et ayant pour titre « Outil auteur pour la production de documents actifs composites structurés » [Pas04], un outil auteur permettant la conception de cockpits a été développé. Cet outil graphique aboutissait à la création d'un document XML correspondant à un environnement d'expérimentation Web dessiné par l'utilisateur. Cette application fournissait différents outils permettant d'insérer plusieurs types de composants multimédias dans différents conteneurs, et d'établir des règles comportementales spécifiques. Toutefois, cette application s'arrêtait à la phase de production d'un document XML et ne traitait pas du tout de la publication côté serveur.

Cela a impliqué que, lors de la conception de l'outil de publication, il a fallu ajuster certaines parties de cette application afin de la rendre conforme à certaines contraintes imposées par l'outil de publication. De ce fait, le document XML produit a également subit quelques modifications. Il a donc fallu ajuster et corriger le Schéma XML [29] décrivant le document XML généré lors de la création d'un cockpit.

Comme présenté dans l'architecture ci-dessus, une archive devra être envoyée sur le serveur dans la partie de publication. Il a donc fallu songer à insérer la gestion des différents fichiers liés au cockpit et de les compresser dans un fichier d'archive.

Finalement, afin de faciliter la tâche du concepteur, il a également fallu songer à offrir un accès à la page de publication du cockpit directement depuis l'outil auteur.

1.4.2 Ebauche de génération et de gestion de conteneurs à partir d'un document XML

Dans le cadre d'un projet de semestre effectué par D. Berkovits, en juin 2004, et ayant pour titre « Outil de publication pour la gestion de l'emplacement spatial au sein d'environnements d'expérimentation Web » [Ber04], une première tentative de gestion

du document XML décrivant un cockpit généré par l'outil auteur a été tentée. Ce projet avait pour objectif de tester la faisabilité d'une génération d'un cockpit à partir de ce document XML. Pour cela, il se limitait à la gestion de conteneurs ayant des images, des hyperliens ou des documents structurés de type HTML.

Par ailleurs, la gestion des différents conteneurs s'était heurtée à un problème très conséquent. En effet, pour que le comportement des conteneurs se déroule de façon plus ou moins cohérente, il fallait user d'une astuce très particulière qui consistait à faire apparaître un message d'alerte dans le navigateur, pour que les actions sur les conteneurs soient possibles. De plus, cette astuce ne fonctionnait que dans le navigateur Internet Explorer sous Windows XP. Il était impossible d'utiliser tout autre navigateur autant sous Windows XP que sous Mac OS X.

2 Adaptation de l'outil auteur

2.1 Introduction

L'outil auteur fournit comme base de ce projet a été effectué lors d'un travail de diplôme il y a environs une année. Ce travail avait pour objectif de générer un document XML à l'issu de la création d'un nouveau cockpit par son concepteur. Cet outil a donc été développé de manière totalement indépendante de tout outil de déploiement du cockpit sur un serveur. Or, lorsque le moment est venu d'effectuer la phase de production du cockpit en reprenant le document XML, il s'est avéré nécessaire d'effectuer quelques ajustements afin de rendre l'ensemble de deux projets totalement compatible. D'autre part, il a fallu compléter cet outil afin qu'il puisse prendre en compte l'intégration de services Web. Nous allons donc présenter ci-dessous les modifications effectuées à l'outil auteur.

Nous pouvons également rappeler que l'outil auteur a été développé en utilisant la plateforme Eclipse [10]. Il a été conçu comme étant un nouveau plug-in qui vient se greffer dans celle-ci.

2.2 Ajustement du document XML généré

2.2.1 Traitement de caractères spéciaux

Lorsque nous avons commencé à générer des environnements à partir des fichiers XML produits par l'outil auteur, nous avons immédiatement constaté que le format de certaines valeurs n'était pas compatible avec les normes utilisées par les navigateurs Internet. En effet, lorsque des fichiers sont reliés à un cockpit particulier, ils sont copiés dans un sous répertoire particulier et rattachés à ce cockpit. Les informations relatives à ces fichiers sont insérées dans le fichier XML, ceci afin de retrouver le fichier en question lors de la phase de déploiement. Toutefois, le chemin du répertoire contenant le fichier étant enregistré de manière relative au répertoire contenant le fichier XML, le caractère '\' était utilisé pour séparer les différents répertoires lorsque l'outil auteur était exécuté sur une plateforme Windows. Or, dans la sémantique utilisée par les navigateurs, le caractère employé pour séparer des répertoires est le caractère '/'. Dès lors, les navigateurs généraient des erreurs car il devenait impossible de retrouver les fichiers référencés par le cockpit. Il a donc fallu revoir le code source d'origine pour que les chemins relatifs des fichiers insérés dans le document XML respectent le format géré par les navigateurs.

Un autre problème survenait également avec les navigateurs lorsqu'un nom de fichier contenait un caractère d'espacement. Pour éviter toute complication, il s'est donc avéré nécessaire de vérifier les noms attribués aux différents composants par le concepteur d'un cockpit. Lorsque un caractère d'espacement est détecté, il est remplacé par un caractère de soulignement.

2.2.2 Adaptation des paramètres des composants

Il a fallu également ajuster dans certains cas les paramètres à insérer pour les composants multimédias. Par exemple, pour les applets, il n'avait pas été prévu de pouvoir insérer une applet sous la forme d'une archive JAR. Or, dans la plupart des cas, une applet se présente sous la forme d'une archive qui contient plusieurs classes nécessaires à l'exécution de l'applet, ainsi que les différents documents utilisés par l'applet, telles que les images, les icônes, etc. La manière d'insérer une applet auparavant imposait qu'elle tienne dans une classe unique et sans lien vers d'autres fichiers, limitant ainsi fortement son potentiel d'utilisation. Pour les formats audio ou vidéo supportés, il s'est avéré primordial d'ajuster les formats disponibles en fonction des formats réellement gérés par la transformation XSLT que nous avons due implémenter.

2.2.3 Mise à jour du Schéma-XML définissant un cockpit

Lorsqu'un nouveau cockpit est créé, toutes les caractéristiques de celui-ci sont enregistrées dans un document XML. Or, ce document doit respecter un certain nombre de règles correspondant à un formalisme lié au développement possible des cockpits. Ce formalisme particulier est défini dans un Schéma-XML.

Or, comme la structure du document XML généré a été partiellement retouchée, il a fallu reprendre le Schéma-XML utilisé pour décrire le standard XML de nos cockpits. Lors de cette opération, nous avons aussi corrigé d'autres incohérences retrouvées dans le schéma qui ne correspondait pas toujours aux documents XML produits par l'outil auteur d'origine.

2.3 Nouvelles fonctionnalités

2.3.1 Décompression des fichiers d'archive

Lors de la conception d'un cockpit, il est possible d'insérer un document structuré de type HTML à l'intérieur d'un conteneur. Cependant, un document HTML est rarement constitué de texte brut, mais par contre, souvent de documents tels que des images, des vidéos, des feuilles de style ou autres scripts. Il a donc été prévu de permettre à l'utilisateur de fournir tous les fichiers liés à un document HTML sous la forme d'une archive ZIP ou JAR, et ceci afin qu'il ne charge, vers l'outil auteur, qu'un seul fichier et non pas chaque fichier un à un ! Cependant, lors de la publication de cet environnement, lorsque le déploiement de ce cockpit sur le serveur sera achevé, si une connexion est établie avec la page HTML précitée, aucun document référencé ne sera détecté par cette page. Cette situation serait tout à fait logique étant donné que tous les fichiers référencés se trouveraient dans une archive. Il a donc fallu imaginer une solution à ce problème. Pour cela, nous avons décidé que, lorsque le concepteur du cockpit effectue une référence à une archive, celle-ci sera automatiquement décompressée dans le sous répertoire rattaché au cockpit. Dès lors, le problème présenté ci-dessus ne sera plus possible étant donné que tous les fichiers référencés seront directement accessibles par le document HTML se trouvant dans un conteneur du cockpit.

Le processus de décompression des archives est pris en charge par la classe `UnZip.java` du paquetage `org.cockpitcomposer.rendering`.

2.3.2 Génération d'une archive ZIP

Le but ultime de la création d'un nouvel environnement Web sous la forme d'un cockpit étant de le déployer sur un serveur, il a fallu imaginer la manière dont tous les fichiers liés à un cockpit particulier allaient être envoyés au serveur chargé de le déployer. Etant donné qu'il faudra émettre un fichier à travers le réseau Internet, il est primordial de minimiser la quantité d'information à transmettre. Pour cela, nous souhaitons transmettre un fichier unique de taille minimale. Afin de satisfaire à ces exigences que nous nous sommes imposées, il a fallu mettre au point une API qui va rassembler tous les documents d'un cockpit dans un fichier d'archive. Cette API va par ailleurs utiliser une méthode de compression ZIP dont les librairies sont fournies par défaut dans les versions actuelles de Java. Comme nous souhaitons une archive de taille minimale, nous allons utiliser systématiquement le facteur de compression maximal. Le fait d'utiliser un facteur de compression maximal implique un temps de création plus lent de l'archive compressée, mais ce temps supplémentaire peut être négligé avec la puissance des machines actuelles d'une part, et d'autre part, ce temps supplémentaire sera largement rattrapé lors de la transmission de l'archive sur le réseau, vers le serveur.

Lors de la génération de l'archive, le document XML décrivant le cockpit est pris en compte, ainsi que le sous répertoire lié à ce cockpit et contenant l'ensemble des documents référencés par le document XML. Ce processus est pris en charge par la classe `Zip.java` du paquetage `org.cockpitcomposer.rendering`.

2.3.3 Sélection d'un cockpit à déployer

Pour permettre au concepteur du cockpit d'accéder au serveur afin d'effectuer la phase de déploiement directement depuis l'outil auteur, nous avons rajouté un module de sélection du cockpit à déployer [Figure 2-1].

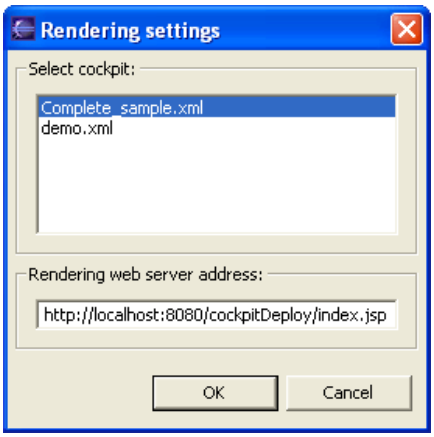


Figure 2-1 Fenêtre de sélection d'un cockpit à déployer

Ce module est accessible par un bouton directement depuis l'outil auteur, et se présente sous la forme d'une fenêtre dans laquelle apparaissent tous les cockpits disponibles, ainsi que l'adresse du serveur utilisé pour le déploiement. Pour déterminer les cockpits à afficher dans la fenêtre, nous parcourons le répertoire de travail nommé 'cockpits' dédié à la gestion des cockpits. Nous récupérons tous les cockpits contenus dans ce répertoire et en affichons les noms dans une liste. Dès lors, l'utilisateur n'a plus qu'à sélectionner le cockpit qu'il souhaite générer parmi ceux affichés.

Quant à l'adresse du serveur affichée dans la fenêtre, il s'agit de l'adresse du serveur définie par défaut dans le fichier de paramétrage du projet `ConfigConstants.java` du paquetage `org.cockpitcomposer`. Ainsi, lorsque le concepteur a sélectionné le cockpit à générer parmi ceux disponibles dans la liste et cliqué sur le bouton 'OK', le navigateur par défaut est démarré vers l'adresse du serveur.

Le processus présenté ci-dessus est géré par les classes `StartRenderingAction.java` et `StartRenderingDialog.java` du paquetage `org.cockpitcomposer.rendering`.

2.3.4 Démarrage du navigateur par défaut

Une fois le cockpit à déployer sélectionné, il a fallu gérer le démarrage du navigateur par défaut du système. Ceci comporte des caractéristiques particulières puisque l'utilisateur peut avoir plusieurs navigateurs différents installés sur son ordinateur, et ces navigateurs varient également en fonction de l'environnement sur lequel s'exécute l'outil auteur. De plus, si plusieurs navigateurs entrent en concurrence, un seul navigateur est définit par défaut. Il a donc fallu gérer ces différentes particularités afin de garder un code compatible sur les plateformes Mac, Windows et Unix. Pour cela, nous faisons appel à la classe `BrowserControl.java` [12] qui gère le démarrage du navigateur par défaut en fonction de la plateforme. Par ailleurs, si l'outil auteur est exécuté sur une plateforme Mac, le démarrage du navigateur par défaut est géré par la classe `BrowserLauncher.java` [11].

Finalement, afin de prendre en charge le bon cockpit par l'outil de déploiement sur le serveur, nous complétons le lien vers l'outil de déploiement passé au navigateur par défaut. Pour cela, nous rajoutons comme paramètre le nom complet avec sa position absolue de l'archive ZIP contenant les fichiers du cockpit. Ceci est effectué en vue de la phase de déploiement côté serveur où l'utilisateur sera convié à sélectionner cette archive avant de générer le nouvel environnement.

2.3.5 Prise en charge des services Web

Finalement, il est important de préciser que nous avons rajouté la possibilité pour le concepteur d'intégrer des services Web [30] au sein de son cockpit. Pour cela, un service Web est inséré de la même façon qu'un autre composant. Le concepteur se doit simplement d'insérer le fichier WSDL [31] de description d'un service Web de la même manière qu'il le ferait pour une page HTML dans son cockpit. Concernant la définition des concepts se rattachant aux services Web, ils seront présentés au chapitre 4 dans la section traitant de l'intégration des services Web.

2.4 Conclusion

Nous avons présenté dans ce chapitre les retouches effectuées à l'outil auteur original. Celles-ci se sont avérées minimes mais essentielles en vue d'une compatibilité nécessaire avec l'outil de déploiement du cockpit sur le serveur.

Dans un deuxième temps, nous avons présenté les fonctionnalités supplémentaires qui ont été rajoutées à l'outil auteur. Nous avons présenté les outils de gestion des archives pour leur décompression et leur compression. Après cela, nous avons montré la manière de sélectionner un cockpit à générer, suivie du démarrage du navigateur par

défaut vers l’adresse du serveur. Finalement, nous avons indiqué que nous avons ajouté la prise en charge des services Web par l’outil auteur.

En conclusion, nous avons tenté de minimiser le plus possible les modifications à effectuer sur l’outil auteur, étant donné que ceci ne rentrait pas dans le cadre de ce projet et que nous étions en possession d’un outil fonctionnant déjà de manière performante.

3 Mise en œuvre de l'outil de publication

3.1 Génération et gestion des conteneurs

3.1.1 Introduction

Le but du projet de semestre qui a été repris [Ber04], était de récupérer les résultats du fichier XML produit par l'outil auteur et d'évaluer la possibilité de générer dynamiquement le contenu de ce fichier par l'interaction de conteneurs. A travers une transformation XSL, le fichier XML devait être parcouru pour générer automatiquement des fichiers HTML. La gestion des différents conteneurs et de leur interaction est faite en JavaScript. Les conteneurs sont des fenêtres comparables à des popup apparaissant dans une page Web et interagissant entre-eux en fonctions des paramètres enregistrés dans le fichier XML.

Malheureusement, le programme ne fonctionnait pas correctement à cause de certains bogues complexes qui n'avaient pas pu être résolus. Or, ce travail étant nécessaire pour la suite du projet, il a donc été demandé, dans un premier temps, de tout reprendre afin de l'analyser et de corriger toutes les erreurs, pour pouvoir travailler sur une base solide. Nous allons donc commenter dans les lignes qui suivent les solutions apportées pour résoudre les différents bogues, les modifications effectuées, ainsi que les améliorations apportées à cet outil.

3.1.2 Remarque préliminaire : JavaScript

Avant de poursuivre, il s'avère essentiel d'effectuer une remarque très importante. En effet, à sa base, toute page Web est décrite par du code HTML. Le code HTML consiste en un langage de balisage qui décrit la structure typographique et graphique d'un document Web. Par défaut, le code HTML est un code purement statique. Toutefois, l'évolution des technologies Web a permis le développement de solutions pour rendre ce code HTML dynamique à travers un langage de script, le JavaScript. Ce langage apporte des solutions pour modifier de manière dynamique les différentes propriétés constituant une page HTML.

Le JavaScript, comme son nom l'indique, est un langage de script et non pas un langage de programmation compilé ; le JavaScript offre donc des possibilités de débogage très réduites et difficiles à mettre en place. De plus, le code est exécuté du côté du client et non pas du côté serveur, cela implique que l'exécution d'un programme JavaScript va dépendre du navigateur utilisé par l'utilisateur, de sa version, de son système d'exploitation, ainsi que de la puissance de sa machine. Toutes ces particularités ajoutées au fait que le JavaScript ne suit pas une norme commune définie par les concepteurs de navigateurs Web (fonctions existantes pour un navigateur, mais pas pour l'autre ; comportements différents de certaines fonctions selon les navigateurs) font que l'écriture de code en JavaScript peut rapidement tourner au cauchemar si l'on souhaite une compatibilité maximale.

3.1.3 Situation et fonctionnement du programme lors de la reprise

Lors de la reprise du projet, le programme fonctionnait partiellement sur Internet Explorer grâce à une astuce faisant apparaître un message d'alerte à l'écran [Figure 3-1]. Sous Netscape, le message d'alerte n'apparaissait même pas, toutes les fonctionnalités étaient bloquées, aucune interaction n'était possible. Sous Internet Explorer, le message d'alerte avait été inséré, sinon rien ne fonctionnait et un message d'erreur apparaissait à l'écran. Dans le rapport du projet, aucune indication ne permettait de véritablement comprendre pourquoi le programme fonctionnait partiellement sous Internet Explorer en utilisant cette astuce particulière. Il a donc été nécessaire de parcourir l'ensemble du code, ligne par ligne, afin de s'imprégner de la méthode de programmation pour rechercher une erreur quelconque. Malheureusement, cela n'a pas permis de trouver ce qui ne fonctionnait pas dans ce qui avait été écrit.

Figure 3-1 Message d'alerte affiché dans la version d'origine

3.1.4 Origine des bogues

Heureusement, le navigateur Mozilla possède un outil de débogage JavaScript qui permet d'exécuter le code ligne par ligne en temps réel dans le navigateur, tout en analysant l'évolution des variables contenues dans la mémoire. Cette analyse approfondie a permis de déceler que le programme faisait appel systématiquement à des objets de valeur nulle qui n'avaient pas été initialisés dans la mémoire du navigateur. Dès lors, on pouvait supposer que Internet Explorer, en faisant afficher le message d'alerte, poursuivait le chargement des valeurs en arrière plan, permettant ainsi le fonctionnement partiel du programme après que l'utilisateur ait appuyé sur le bouton 'OK' du message d'alerte. Toutefois, un tel comportement n'était pas très logique car après le message d'alerte, le programme aurait dû poursuivre dans le même état qu'il était juste avant l'affichage du message. Par déduction, nous pouvons affirmer que le fonctionnement de Netscape était plus logique, car l'affichage d'un tel message ne doit rien modifier au fonctionnement du programme en soi.

Après avoir compris que le programme utilisait systématiquement des objets qui n'étaient pas initialisés, il a fallu évaluer la manière qui avait été choisie pour programmer les différentes fonctions souhaitées. Nous pouvons décrire le fonctionnement d'un environnement géré par un cockpit de la façon suivante :

> *Une page HTML principale, contenant le code JavaScript, permet de définir la structure de la page avec ses différents conteneurs. Or, chaque conteneur représente également une page HTML extérieure qui va être chargée dans l'espace qui lui est mis à sa disposition par la page principale. Cette page agit de manière autonome, comme si elle était elle-même entièrement dans le navigateur, et elle ne "voit" pas qu'elle est en réalité dans un conteneur. Ainsi, si dans la fenêtre du navigateur il y a*

trois conteneurs différents qui s'affichent, ces trois conteneurs représentent trois pages HTML différentes. Pour afficher ces trois conteneurs, nous aurons donc en tout quatre pages HTML: une page HTML pour gérer les conteneurs et leurs interactions, plus une page HTML par conteneur.

Le bogue le plus important survenait du fait que la page HTML principale, contenant toute la structure des conteneurs en JavaScript, utilisait des fonctions qui faisaient appel aux pages HTML des conteneurs. Ces pages HTML n'étaient donc pas encore chargées dans la mémoire du navigateur, alors que celui-ci devait déjà y faire référence à travers les fonctions JavaScript écrites. Il s'est donc avéré que l'ensemble du code JavaScript écrit dans le cadre du projet de semestre ne pouvait pas fonctionner, car toutes les fonctionnalités développées utilisaient cette manière de faire. Il a donc fallu recommencer depuis le début, et réécrire tout le code JavaScript, en partant du code mis à disposition par B. Gosselin (Popup Windows) [8]. Ce code avait été le point de départ du projet de semestre qu'il a fallu reprendre.

Le code de B. Gosselin gère la création d'une fenêtre popup en JavaScript dans un navigateur. Un popup correspond, dans notre cas, à la définition d'un conteneur. Par contre, ce code ne gère pas la génération de plusieurs popups dans une même page Web, ni les différentes interactions entre plusieurs fenêtres, telles que la fermeture, le déplacement ou encore le redimensionnement coordonné entre plusieurs fenêtres popups que nous développerons plus bas.

3.1.5 Descriptif des bogues et solutions apportées

I) Remarque

Dans les lignes ci-dessous, nous parlerons de différents types d'événements possibles sur les conteneurs. Pour chaque événement, une action particulière intervient. Les événements sont les suivants :

ONCLICK Il est possible de définir un hyperlien dans un conteneur, et de le lier à un événement ONCLICK permettant d'ouvrir la cible de l'hyperlien dans un conteneur différent de celui dans lequel il se trouve.

ONCLOSE Evénement correspondant à la fermeture d'un conteneur. Lors d'un tel événement, il est possible de lier plusieurs conteneurs entre-eux. Ainsi, lors de la fermeture d'un conteneur source, tous les autres conteneurs liés par cet événement se fermeront également.

ONMOVE Evénement correspondant au déplacement d'un conteneur. Lors d'un tel événement, il est possible de lier plusieurs conteneurs entre-eux. Ainsi, lors du déplacement d'un conteneur source, tous les autres conteneurs liés par cet événement se déplaceront également.

ONSIZE Evénement correspondant au redimensionnement d'un

conteneur. Lors d'un tel événement, il est possible de lier plusieurs conteneurs entre-eux. Ainsi, lors du redimensionnement d'un conteneur source, tous les autres conteneurs liés par cet événement se redimensionneront également.

II) Bogue 1: Affichage des liens dynamiques (hyperliens) dans un autre conteneur (événement ONCLICK)

a) CAUSE

Une des possibilités importantes d'une gestion de l'affichage dans plusieurs conteneurs est le fait de pouvoir cliquer sur un hyperlien situé dans un conteneur et de faire afficher la page HTML correspondante à cet hyperlien dans un autre conteneur. La méthode utilisée précédemment faisait référence à un objet nul pour les raisons expliquées au paragraphe précédent. La solution adoptée était compliquée et tentait de modifier dynamiquement l'attribut "`target`" de la page source lorsqu'un clic survenait sur le lien. Par ailleurs, la fonction JavaScript utilisée était contenue dans le fichier HTML principal de gestion des conteneurs et non pas dans la page où se trouvait l'hyperlien. Ce code tentait donc d'accéder à des valeurs contenues dans un autre fichier HTML.

b) SOLUTION

La solution adoptée pour corriger ce problème est en réalité toute simple. En effet, lorsqu'on définit un lien dynamique dans une page HTML, la norme HTML permet de définir un attribut "`target`" auquel correspond le nom d'une fenêtre HTML de destination, dans laquelle nous souhaitons que la page apparaisse lorsque nous cliquons sur le lien. Or, chaque conteneur possède un nom unique, défini en fonction de son identifiant. Il suffit donc, dans le target du lien, de préciser le nom du conteneur dans lequel la page doit apparaître.

Cette solution ne nécessite aucune gestion de cet événement par du code JavaScript, il ne s'agit que de pur code HTML. Or, ces pages HTML sont générées automatiquement à partir d'un fichier XML, en suivant une transformation XSLT. Il ne se pose donc aucun problème pour gérer cet événement lors de la génération des pages HTML.

Cette solution fonctionne correctement sur tous les navigateurs et sur tous les systèmes d'exploitation courants.

III) Bogue 2: Redimensionnement automatique des conteneurs lors du redimensionnement de la fenêtre principale du navigateur

a) CAUSE

Auparavant, lorsque l'utilisateur redimensionnait la fenêtre principale du navigateur, rien ne se passait, la taille des conteneurs restait identique, ce qui ne favorisait pas la lisibilité du contenu.

b) SOLUTION

Pour y remédier, une fonction "`resizeAll()`" a été implémentée. Cette fonction est appelée chaque fois que le navigateur intercepte un événement de type "`window.onresize`". Cet événement se produit lorsque l'utilisateur redimensionne la fenêtre du navigateur. La fonction "`resizeAll()`" calcule les nouvelles dimensions disponibles pour l'affichage dans le navigateur. Ces valeurs, combinées aux anciennes dimensions, nous donnent le ratio du redimensionnement. Dès lors en multipliant les dimensions de chaque conteneur avec ce ratio, il devient possible de modifier leur taille en fonction du redimensionnement global de la fenêtre du navigateur effectué par l'utilisateur. L'aspect initial est ainsi conservé lors d'un redimensionnement.

IV) Bogue 3: Gestion de la fermeture simultanée des conteneurs liés par l'opération de fermeture (événement ONCLOSE)

a) CAUSE

Grâce à l'outil auteur, l'utilisateur peut définir certains comportements spécifiques pour déterminer une certaine interaction entre les conteneurs. Un de ces comportements est la fermeture liée de plusieurs conteneurs. En effet, il est possible de définir que, lorsqu'un utilisateur ferme un conteneur particulier, d'autres conteneurs se ferment également avec lui.

La solution proposée auparavant se trouvait confrontée aux mêmes problèmes de valeurs nulles pour les objets lors de l'exécution des fonctions. Par ailleurs, la solution consistait à générer une fonction particulière pour chaque liaison entre deux conteneurs. Ainsi, si un conteneur était lié à deux autres conteneurs par une clause de fermeture commune, il fallait générer deux fonctions différentes. En plus du fait que la solution proposée ne fonctionnait pas, il est évident que cette solution alourdissait considérablement le code, impliquant donc des risques de ralentissements considérables lors de l'exécution. Par ailleurs cette solution était relativement redondante, car les fonctions générées étaient toutes identiques à quelques noms de variables près!

b) SOLUTION

Pour remédier à ce problème, il a fallu réfléchir à une solution évitant la génération de fonctions de manière dynamique, lors de la génération du code par la transformation XSLT, pour ne pas se retrouver avec une multitude de fonctions redondantes.

Pour cela, une solution simple a été trouvée. En effet, la solution de base, implémentée par B. Gosselin pour fermer un conteneur simple, fait appel à la fonction "`hidebox(id)`". L'idée était donc d'utiliser cette même fonction par toutes les fenêtres liées, lors d'une fermeture liée de plusieurs conteneurs. La principale difficulté était d'implémenter une méthode pour retrouver les conteneurs qui étaient liés pour la fermeture. Si cette information est disponible, alors il devient trivial de faire appel successivement à la fonction de base "`hidebox(id)`", en lui passant à chaque fois l'identifiant `(id)` de tous les conteneurs liés par la fermeture. L'idée a donc été de faire appel à une liste particulière lors de fermeture d'un conteneur. Cette liste doit contenir l'ensemble des identifiants des conteneurs à fermer lors de la fermeture du conteneur source. Dès lors, chaque conteneur doit posséder sa propre liste. Le principal avantage de cette solution se trouve dans le fait que, lors de la transformation XSLT pour générer

le code JavaScript dans la page HTML, il suffit simplement d'insérer, dans des listes, les identifiants des conteneurs liés par la fermeture pour chaque conteneur. Dès lors, il n'est plus nécessaire de générer dynamiquement de multiples fonctions.

Ainsi, dans le code original, nous remplaçons l'appel de la fonction "`hidebox(id)`" par la fonction "`hideboxes()`". La fonction "`hideboxes()`" est une nouvelle fonction que nous avons écrites, et qui va récupérer la liste des identifiants des conteneurs liés au conteneur source pour l'événement ONCLOSE, grâce à une fonction "`getCloseIdList(winId)`" que nous avons écrite. Cette nouvelle fonction renvoie la liste des identifiants des conteneurs liés pour l'événement ONCLOSE, en prenant en paramètre l'identifiant du conteneur source. Une fois que cette liste est récupérée, la fonction "`hideboxes()`" boucle sur le contenu de la liste, et applique la fonction de base "`hidebox(id)`" sur chaque identifiant contenu dans la liste.

Etant donné que le contenu des listes est généré dynamiquement lors de la transformation XSLT, et que nous ne sommes pas sensés connaître d'avance le nombre de conteneurs qui vont se trouver dans la fenêtre du navigateur, il a fallu trouver un moyen, afin de récupérer chaque liste par la fonction "`getCloseIdList(winId)`", sachant que cette fonction se trouvant dans le fichier "`popupManager.js`", et que ce fichier n'est pas généré dynamiquement. Par ailleurs, nous supposons que nous ne connaissons pas la structure des noms des identifiants des conteneurs, ni leur nombre total. Pour cela, nous devons utiliser le prédicat qui stipule que lorsque l'utilisateur clique sur le bouton de fermeture d'un conteneur, ce conteneur va se fermer. Partant de ce fait trivial, nous imposons alors que, l'identifiant de chaque conteneur source se trouve toujours à la première place de chaque liste qui lui est propre. Dès lors, il devient trivial de retrouver la liste correspondant à un conteneur particulier, sachant qu'il est possible d'identifier cette liste par son premier élément!

Pour résumer, nous utilisons une liste par conteneur. Cette liste contient l'ensemble des identifiants des conteneurs à fermer. Finalement, le premier élément de chaque liste correspond à l'identifiant du conteneur auquel elle appartient.

Cette solution est simple, rapide et efficace. Elle fonction correctement avec tous les navigateurs.

La figure 3-2 représente la structure des listes contenant les identifiants des conteneurs pour les trois types d'événements. Les conteneurs C2 et C3 sont liés au conteneur C1 pour la fermeture. C1 est lié à C2 pour la fermeture et le redimensionnement, C3 est lié à C2 pour le déplacement. Finalement, C2 est lié à C3 pour le déplacement.

Conteneur C1		
Close	*Move*	*Size*
C1	*C1*	*C1*
C2		
C3		

Conteneur C2		
Close	*Move*	*Size*
C2	*C2*	*C2*
C1	*C3*	*C1*

Conteneur C3		
Close	*Move*	*Size*
C3	*C3*	*C3*
	C2	

Figure 3-2 Structure des listes pour la gestion des événements sur les conteneurs

c) Remarque : non transitivité des événements

La méthode de fermeture implémentée et expliquée ci-dessus n'est pas transitive. A savoir, dans la figure 3-2, en fermant le conteneur 2 (liste *Close* du conteneur C2), le conteneur 1 sera aussi fermé mais pas le 3, car C1 est contenu dans la liste, mais pas C3. Or, si la méthode était transitive, lorsque le conteneur 1 se ferme à la suite du 2, le conteneur 3 devrait aussi se fermer par transition de la fermeture du 1, car C3 est contenu dans la liste *Close* du conteneur C1. Le fait de ne pas gérer la transitivité de l'événement permet une plus grande souplesse dans la gestion des conteneurs et offre plus de libertés à l'utilisateur.

Cette remarque se généralise à tous les événements faisant appel à la même méthode, à savoir les événements ONCLOSE, ONMOVE et ONSIZE.

V) Bogue 4: Gestion du déplacement simultané des conteneurs liés par l'opération de déplacement (événement ONMOVE)

a) CAUSE

Le problème rencontré était très semblable à celui du bogue 3 décrit dans la section IV ci-dessus. Les causes étaient identiques, mais cette fois, ce n'étaient pas une, mais deux fonctions particulières qui étaient créées dynamiquement pour chaque liaison entre deux conteneurs. Ainsi, si un conteneur était lié à deux autres conteneurs par une clause de déplacement commun, il fallait générer quatre fonctions différentes.

b) SOLUTION

La solution appliquée est identique à celle présentée pour le bogue 3. Pour résumer, nous utilisons une liste par conteneur. Cette liste contient l'ensemble des identifiants des conteneurs à déplacer. Finalement, le premier élément de chaque liste est l'identifiant du conteneur auquel il appartient.

La remarque sur la transitivité, expliquée dans la section IV ci-dessus, s'applique également dans ce cas.

VI) Bogue 5: Gestion du redimensionnement simultané des conteneurs liés par l'opération de redimensionnement (événement ONSIZE)

a) CAUSE

Le problème rencontré était très semblable à celui du bogue 3 décrit dans la section IV ci-dessus. Les causes étaient identiques, mais cette fois, comme pour le bogue 3, ce n'étaient pas une, mais deux fonctions particulières qui étaient créées dynamiquement pour chaque liaison entre deux conteneurs. Ainsi, si un conteneur était lié à deux autres conteneurs par une clause de redimensionnement commun, il fallait générer quatre fonctions différentes.

b) SOLUTION

La solution appliquée est identique à celle présentée pour le bogue 3. Pour résumer, nous utilisons une liste par conteneur. Cette liste contient l'ensemble des identifiants des conteneurs à redimensionner. Finalement, le premier élément de chaque liste est l'identifiant du conteneur auquel il appartient.

La remarque sur la transitivité, expliquée dans la section IV ci-dessus, s'applique également dans ce cas.

Par contre, en implémentant cette solution, un autre problème s'est posé. En effet, il s'est avéré impossible de déplacer les conteneurs, tout en les redimensionnant. Pour cet événement, le déplacement des conteneurs, liés au conteneur source qui est redimensionné, s'avère nécessaire. De plus, les conteneurs liés sont également redimensionnés de manière inversement proportionnelle à la variation de taille du conteneur source. Lors de l'implémentation, il s'est avéré possible d'effectuer soit un déplacement, soit un redimensionnement, mais pas les deux simultanément. La solution choisie a été d'effectuer un déplacement de tous les conteneurs liés au conteneur source, et le redimensionnement est effectué lorsque l'utilisateur lâche l'icône de redimensionnement du conteneur avec sa souris.

Cependant, nous avons dû distinguer quatre cas particuliers de gestions différentes des déplacements liés au redimensionnement d'un conteneur source :

1) Le premier cas a lieu lorsque le conteneur lié est à droite du conteneur source. Dans ce cas, nous effectuons un déplacement horizontal équivalent au redimensionnement horizontal du conteneur source, ainsi qu'un redimensionnement horizontal inversement équivalent.

2) Le deuxième cas a lieu lorsque le conteneur lié est au-dessous du conteneur source. Dans ce cas, nous effectuons un déplacement vertical équivalent au redimensionnement vertical du conteneur source, ainsi qu'un redimensionnement vertical inversement équivalent.

3) Le troisième cas a lieu lorsque le conteneur lié est à droite et au-dessous du conteneur source. Dans ce cas, nous effectuons un déplacement vertical et horizontal, ainsi qu'un redimensionnement horizontal et vertical inversement équivalent.

4) Le quatrième se produit lorsque le conteneur lié est à gauche et au-dessus du conteneur source. Lorsque le conteneur lié se situe à cette position, le redimensionnement des deux conteneurs est possible simultanément car aucun déplacement n'est effectué. C'est la seule situation où un redimensionnement dynamique des conteneurs est effectué.

La distinction de ces quatre cas est due au fait que le redimensionnement d'un conteneur s'effectue par l'icône de redimensionnement située en bas à gauche du conteneur. Ces quatre cas nous garantissent un comportement cohérent du redimensionnement.

La limite entre ces quatre cas est calculée en fonction des coordonnées du point supérieur gauche du conteneur lié, par rapport à un point situé au 3/4 du point supérieur gauche du conteneur source.

La figure 3-3 représente graphiquement la distinction entre ces quatre cas. Dans cette figure, le plan dans lequel se trouve le conteneur source est divisé en quatre parties différentes, représentant les quatre cas qui vienne d'être décrits.

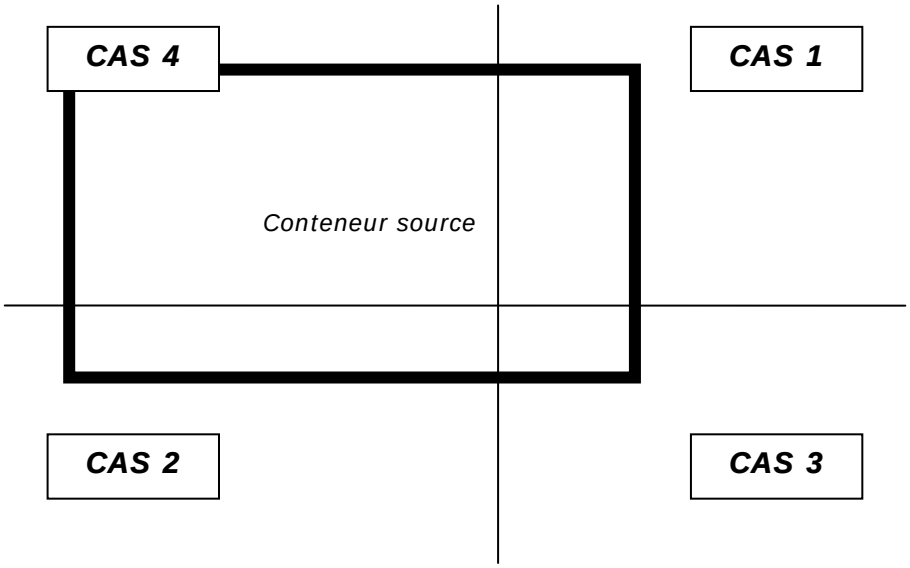


Figure 3-3 Distinction des quatre cas possibles pour la gestion du redimensionnement d'un conteneur source lié à un autre conteneur pour cette opération

3.1.6 Améliorations du comportement du gestionnaire de conteneurs

I) Création de nouvelles icônes d'interaction

Une fois que tous les bogues décrits ci-dessus ont pu être corrigés, il a fallu se concentrer sur le rendu graphique des conteneurs afin que ceux-ci ressemblent à des fenêtres standard d'une application Windows ou Mac OS X. Pour cela, un ensemble de boutons a été conçu afin de rendre plus vivante l'interaction avec les conteneurs. Par ailleurs, l'aspect graphique variant sous Mac OS X, des boutons de type "Aqua", spécifiques à Mac OS X, ont été dessinés. Dès lors, lorsque le curseur de la souris passe sur un bouton, celui-ci s'éclaircit alors qu'un des signes '+', '-' ou 'x' apparaît sur Mac OS X en relatif à la fonctionnalité du bouton en question. Lorsqu'une fenêtre est maximisée, le bouton de maximisation est grisé pour indiquer que cette action n'est plus possible, alors que le bouton de minimisation est activé. De même lors d'une minimisation, le bouton de minimisation est grisé pour indiquer que cette action n'est plus possible, alors que le bouton de maximisation est activé. Sur Mac OS X, lorsqu'un conteneur est activé, sa barre de titre et son bord apparaissent en gris foncé, mais lorsqu'il est inactif sa barre de titre et son bord apparaissent en gris clair.

Finalement, si un conteneur est définit comme n'étant pas redimensionnable, les icônes de maximisation, de minimisation et de redimensionnement n'apparaissent tout simplement pas.

II) Amélioration de la fluidité

Lorsque l'utilisateur effectue un redimensionnement ou un déplacement d'un conteneur, le comportement général de l'application s'est avéré très désagréable, car l'ensemble du contenu était constamment régénéré. Ce comportement utilisait une grande puissance du processeur et donnait une impression de saccades très désagréable. Pour remédier à ce problème gênant, nous avons opté pour masquer l'ensemble du

contenu de chaque conteneur. Le navigateur peut ainsi mieux gérer les opérations de déplacement et de redimensionnement. Ainsi, lorsque l'utilisateur clique sur la barre de titre (déplacement) ou sur l'icône de redimensionnement en bas à droite de la fenêtre du conteneur, le contenu de chaque conteneur disparaît. Lorsque l'utilisateur relâche le bouton de la souris, l'ensemble de tous les contenus réapparaissent aussitôt.

Cette solution favorise une utilisation plus aisée de ces deux opérations, mais entraîne également un gros désavantage. En effet, lorsque l'utilisateur clique simplement sur la barre de titre d'un conteneur pour en prendre le focus, mais sans effectuer de déplacement, l'ensemble des contenus de tous les conteneurs disparaissent et réapparaissent aussitôt. Ce comportement donne l'impression d'un scintillement très désagréable et risque de distraire l'attention de l'utilisateur. Pour éviter ce genre de désagrément, il a donc été décidé de mesurer un certain déplacement relatif du conteneur, avant de masquer les contenus. Le seuil a été fixé à une variation de déplacement minimal de 3 pixels, car il est possible que lors du clic bref sur la barre de titre, le conteneur se déplace légèrement de 1 ou 2 pixels. Ainsi en mesurant ce déplacement relatif minimal, nous pouvons considérer qu'il s'agit bien d'une situation où l'utilisateur souhaite réellement déplacer un conteneur.

III) Gestion dynamique des événements par l'utilisateur

Après avoir effectué les débogages et les modifications présentées ci-dessus, il a été demandé de vérifier la possibilité d'une interaction dynamique de l'utilisateur sur la gestion des événements de fermeture, de déplacement ou de redimensionnement des conteneurs. En effet, lors de la génération des différents conteneurs avec l'outil auteur, un comportement par défaut peut être déterminé pour les actions entre les différents conteneurs. Cependant ce comportement par défaut pourrait gêner l'utilisateur final lors de certaines manipulations; le fait de pouvoir modifier dynamiquement les liens entre ces conteneurs, directement par l'utilisateur, ajouterait donc un apport non négligeable à la gestion du comportement des conteneurs.

Maintenant, nous pouvons réellement prendre conscience de l'utilité d'avoir choisi de gérer les différents conteneurs par un système de listes. En effet, en plus d'être extrêmement simple, cette solution offre également la possibilité de pouvoir retravailler dynamiquement le contenu de ces listes. En effet, puisque lors de l'exécution d'une fermeture, d'un déplacement ou d'un redimensionnement, la première opération effectuée est la lecture du contenu de la liste liée au conteneur sur lequel se produit l'événement ; il suffit donc de modifier le contenu de cette liste pour modifier le comportement lors de cet événement. Dès lors, le problème principal revient à trouver un moyen simple, rapide et efficace, pour permettre à l'utilisateur de consulter l'état de ces listes et de les modifier à son gré.

La solution adoptée a été d'implémenter un système de menus et de sous-menus, représentant chaque événement et l'ensemble des conteneurs liés ou non à ces événements. Pour cela, nous avons créé une nouvelle icône figurant dans la barre de titre de chaque conteneur, qui permet d'accéder aux menus représentant les conteneurs liés à celui-ci pour chaque événement. Lorsque l'utilisateur clique sur cette icône, un menu apparaît avec la liste des événements possibles sur ce conteneur. En sélectionnant un événement, la liste complète des conteneurs est affichée avec leur propre titre respectif. A gauche de chaque titre se trouve une icône de type '`checkbox`' permettant de savoir si ces conteneurs sont liés ou non. Un simple clic sur la case permet de lier ou de délier chaque conteneur pour l'événement en question. Cette solution fonctionne

parfaitement et les listes contenant les identifiants des conteneurs sont correctement gérées.

Pour implémenter cette solution plusieurs, fonctions ont été développées. Tout d'abord, '`setParamMenu()`' est la fonction qui crée les balises dynamiques contenant le code HTML des menus. Le contenu des menus est créé lorsque l'utilisateur clique sur l'icône d'affichage du menu. Le menu s'affiche sous la position courante du pointeur de la souris, au-dessus de l'icône d'affichage du menu. Avant d'afficher le menu, un test est effectué afin de vérifier que le menu s'affiche de manière entièrement visible à l'écran. En effet, si l'icône pour afficher le menu se trouve à une extrémité de l'écran, le menu risque de se trouver en dehors de la zone visible par l'utilisateur. Pour éviter cela, un petit test contrôle et ajuste, si besoin, la position du menu afin qu'il soit entièrement visible par l'utilisateur. Pour masquer le menu, une icône de fermeture a été ajoutée en haut à gauche du titre du menu.

La gestion des sous-menus affichant la liste des conteneurs liés à l'événement sélectionné dans le menu est effectuée par la fonction '`setSubMenu()`'. Cette fonction parcours la liste du conteneur source pour l'événement sélectionné. Chaque élément contenu dans cette liste correspond à l'identifiant d'un autre conteneur, permettant d'afficher dans ce sous-menu les conteneurs qui sont liés. De même que pour l'affichage des menus, un test est aussi effectué pour vérifier un positionnement cohérent des sous-menus, afin de les rendre visibles par l'utilisateur. Par ailleurs, lorsqu'un sous-menu s'affiche, celui-ci est décalé par rapport au menu pour se retrouver en face de la case sélectionnée dans le menu. Finalement, la case d'un événement du menu reste en évidence, tant que son sous-menus correspondant est affiché.

Les fonctions '`displaySM()`' et '`hideSM()`' sont utilisées respectivement pour afficher et masquer les sous-menus. La fonction '`showLinkedEvent()`' sert à parcourir la liste des conteneurs liés pour ajuster la bonne icône dans le sous-menu dans le cas où un conteneur est lié. Les fonctions '`addLinkedEvent()`' et '`delLinkedEvent()`' sont utilisées respectivement pour ajouter et pour supprimer une liaison entre deux conteneurs, agissant directement sur le contenu des listes liées aux conteneurs.

Ces différentes fonctions fournissent à l'utilisateur final une très grande maniabilité de l'environnement, et une manière toute simple d'influer sur les comportements de chaque conteneur.

La figure 3-4 donne un aperçu visuel des menus de gestion dynamique du comportement conteneur par l'utilisateur final.

Figure 3-4 Menu et sous-menu pour la gestion dynamique des événements

3.1.7 Adaptation du fichier de transformation XSLT

Etant donné que le code qui doit être généré dynamiquement n'est plus le même que précédemment, la plupart du code du fichier de transformation XSLT d'origine a dû être modifié. Seuls les parties générant les entêtes et la fonction '`multiPopup()`' utilisée pour l'initialisation des conteneurs ont été conservées.

Concernant le code que l'on souhaite générer, nous avons séparés le code généré dynamiquement, du code statique ; les parties statiques du code ne sont pas générées par la transformation XSLT. Le reste des fonctions JavaScript se trouvent dans trois fichiers différents, à savoir `popupWinMac.js`, `DHTMLAPI.js` et `popupManager.js`. Les deux premiers fichiers sont appelés au début du code. Le dernier fichier doit être inséré à la suite du code généré dynamiquement, car il contient des fonctions qui exploitent les valeurs générées dynamiquement, telles que les listes liant les différents conteneurs pour certains événements.

Après avoir déclaré la fonction '`multiPopup()`' servant à générer les différents conteneurs, trois tableaux sont déclarés afin d'y insérer les listes des identifiants des conteneurs liés par la fermeture, le déplacement ou le redimensionnement.

La gestion des événements ONCLOSE, ONMOVE et ONSIZE est faite de la même façon dans les trois cas. Nous utilisons une liste par événement et par conteneur. Chaque liste contient les identifiants des conteneurs liés au conteneur source. Ainsi la liste du conteneur 1, lié à un événement de fermeture, contient l'ensemble des identifiants des conteneurs qui seront fermés lorsque le conteneur 1 sera fermé. Il ressort de cette implémentation que chaque liste est liée à un conteneur, et l'identifiant du conteneur sert d'identifiant pour la liste en question. Cet identifiant correspond au premier élément de chaque liste puisque tout événement agit également sur le conteneur qui l'invoque.

La gestion de ces trois événements est réalisée respectivement par trois templates dans chaque cas. Un template pourrait être définit comme une sous-routine dans la syntaxe XSLT. Cette sous-routine peut être invoquée pour un nœud particulier d'un document XML. Nous présentons ci-dessous les templates pour la gestion de l'événement de fermeture. Concernant la gestion des deux autres événements, elle est effectuée exactement de la même façon, seules les noms des variables changent.

1) Le premier template est utilisé pour la déclaration des listes de fermeture pour chaque conteneur.

2) Le deuxième template va insérer dans ces listes les identifiants des conteneurs liés par la fermeture, en insérant toujours comme premier élément, l'identifiant du conteneur auquel appartient la liste.

3) Finalement, le dernier template est en fait utilisé par le deuxième template décrit ci-dessus. En effet, il va rechercher dans le document XML les identifiants des conteneurs liés par une clause de fermeture au conteneur source. Il va donc rechercher les éléments de type `<Connection>` du document XML ayant un attribut '`sourceEventType`' avec la valeur '`ContainerOnClose`'.

Finalement, il peut être utile de préciser que le code HTML généré automatiquement contient également des commentaires générés dynamiquement, afin

de permettre à un utilisateur quelconque de comprendre le fonctionnement du code HTML.

3.1.8 Tests de fonctionnement en fonction du système d'exploitation et du navigateur

L'objectif d'un tel outil déployé sur le Web est bien entendu qu'il puisse fonctionner sur le maximum de plateformes différentes et avec le maximum de navigateurs différents. C'est pourquoi nous présentons ci-dessous les navigateurs sous lesquels a été testé cet outil.

I) Windows

a) Internet Explorer

Fonctionnement idéal, toutes les fonctionnalités sont parfaitement opérationnelles. La fluidité lors des différentes interactions est idéale et très agréable pour les manipulations, aucune saccade n'est à remarquer.

Le bouton 'back' du navigateur gère de manière cohérente les actions de l'utilisateur, lorsqu'il a chargé de nouvelles pages HTML dans les conteneurs.

Le bouton 'reload' conserve le contenu des différents conteneurs, mais réajuste leur taille à la taille initiale.

b) Mozilla et Netscape

Toutes les fonctionnalités sont opérationnelles. Par contre la fluidité est nettement moins bonne que sous Internet Explorer, provoquant quelquefois des comportements légèrement désagréables.

Le bouton 'back' n'a pas un comportement régulier. Tant que l'utilisateur n'a pas fait de reload, il ne peut pas utiliser le bouton 'back'. Par contre, après un 'reload', le bouton 'back' est actif, mais son comportement est identique à l'utilisation du bouton 'reload'; tous les conteneurs sont réinitialisés et leur taille reprend la taille initiale.

Le bouton 'reload' a pour effet de réinitialiser le contenu de tous les conteneurs et de les remettre à leur position et à leur taille initiale.

c) Navigateur conseillé

Internet Explorer, pour sa fluidité et la cohérence du comportement des boutons de navigation.

II) Mac OS X

a) Safari

Fonctionnement idéal, toutes les fonctionnalités sont parfaitement opérationnelles. La fluidité lors des différentes interactions est idéale et très agréable pour les manipulations, aucune saccade n'est à remarquer. La seule chose à spécifier vient du fait que les navigateurs Mac ne supportent pas les effets de transparence. Ainsi, l'ombre portée des conteneurs ne s'affiche pas de manière aussi agréable que sous Windows. Sous Mac, une bande foncée non transparente est dessinée.

b) Netscape

Toutes les fonctionnalités sont opérationnelles. Par contre la fluidité est nettement moins bonne que sous Safari, provoquant quelquefois des comportements légèrement désagréables.

c) Internet Explorer

Ne fonctionne pas du tout, Internet Explorer ne parvient même pas à démarrer, provoquant une erreur générale du programme conduisant à une fermeture instantanée.

d) Navigateur conseillé

Safari pour sa fluidité.

III) UNIX

a) Internet Explorer

Fonctionnement idéal, toutes les fonctionnalités sont parfaitement opérationnelles. La fluidité lors des différentes interactions est idéale et très agréable pour les manipulations, aucune saccade n'est à remarquer.

b) Netscape

Toutes les fonctionnalités sont opérationnelles. Par contre la fluidité est nettement moins bonne que sous Internet Explorer, provoquant quelquefois des comportements légèrement désagréables.

c) Navigateur conseillé

Internet Explorer pour sa fluidité.

3.1.9 Conclusion

Nous pouvons dire finalement que l'objectif fixé a été atteint avec succès. A savoir, au départ, nous étions confronté à un programme qui ne fonctionnait pas correctement et dont il a fallu débusquer toutes les erreurs pour le rendre pleinement fonctionnel. A noter que la presque totalité du code a dû être reprise suite à la découverte de l'origine du problème principal. La principale cause du mauvais fonctionnement de l'application venait du fait que l'ensemble des problèmes rencontrés survenaient lors de l'accès à des objets de valeurs nulles. Ces objets ne pouvaient pas être instenciés au moment de l'exécution des fonctions, car celles-ci faisaient références à des fichiers HTML extérieurs qui n'étaient pas chargés dans la mémoire du navigateur au moment de l'exécution.

Afin de remédier à ces problèmes, il a fallu effectuer une nouvelle approche. Celle-ci évite la génération dynamique de nouvelles fonctions redondantes. Un système de listes liées à chaque conteneur pour chaque événement a été développé. Ces listes contiennent les identifiants des conteneurs sur lesquels vont devoir s'appliquer les différents types d'événement.

Finalement, cette nouvelle approche utilisée s'est avérée très pratique, puisqu'elle a pu être généralisée aux trois événements qu'il fallait gérer pour chaque conteneur, à savoir la fermeture (ONCLOSE), le déplacement (ONMOVE) et le redimensionnement (ONSIZE). De plus, elle a permis de mettre en œuvre une solution permettant à

l’utilisateur final de reconfigurer à sa guise les différents événements sur chaque conteneur, et cela d’une manière simple et intuitive.

Dès lors, la conception du fichier XSL s'est également avérée grandement facilitée, car elle se limitait à la génération automatique des listes pour chaque événement et chaque conteneur.

L'outil, ainsi obtenu, se trouve donc maintenant prêt et fonctionnel pour être utilisé dans la suite de ce projet.

3.2 Gestion des composants multimédias et développement d'outils complémentaires

3.2.1 Introduction

Un environnement Web ne se limite pas à de simples lignes de textes affichées sur une page HTML. Depuis la naissance du réseau Internet, les technologies Web ont évolué dans le but de rendre le contenu des documents Web le plus interactif et le plus dynamique possible. De nos jours, un véritable document multimédia comprend en plus du texte et des images, des vidéos, du son, des animations, des applications intégrées, d'autres documents dynamiques, etc.

Etant donné que ce projet rentre dans le cadre du développement d'une plateforme d'enseignement et d'apprentissage à distance, il s'avère donc primordial de gérer le plus vaste choix de composants multimédias possibles. Dans cette optique, l'intégration de composants multimédias spécifiques dans un document HTML nécessite l'utilisation d'applications extérieures qui ne sont pas intégrées par défaut dans les différents navigateurs Internet. Afin de pouvoir faire fonctionner ces composants externes dans une page Web, il faut utiliser un plug-in spécifique pour chaque type de composant multimédia.

Par ailleurs, il a été demandé de développer différentes API avec des spécificités particulières, dans le but d'une intégration future dans l'environnement *eMersion*.

3.2.2 Gestion des plug-in's multimédias

I) Pourquoi des plug-in's ?

Comme nous l'avons présenté ci-dessus, un plug-in est une application externe qui vient s'intégrer au navigateur Internet afin d'exécuter un type de document multimédia qui lui est propre. Par exemple, pour visionner des vidéos intégrées à une page Web, il faut utiliser un plug-in de visualisation. En poursuivant l'exemple des vidéos, nous pouvons préciser qu'il existe plusieurs types de vidéos différentes et plusieurs plug-in's différents pour chaque type de vidéo ! La plupart des plug-in's prennent en charge la plus grande majorité des formats de vidéo, mais pas tous. En effet, les plug-in's sont fournis par les compagnies qui ont développées certains formats qui leurs sont propres. Du coup, les plug-in's qu'ils fournissent gèrent, en plus de leurs propres formats, la plus grande majorité des formats ouverts, mais pas ceux de leurs concurrents. Ceci a pour conséquence qu'il faut gérer la détection du format de vidéo, et de lancer le plug-in spécifique à chaque type de vidéo !

En plus des vidéos, l'outil auteur prévoit également de gérer des sons, des applets, des documents SMIL, SVG et Flash. Tous ces différents types de documents multimédias nécessitent également leur propre plug-in.

II) Implémentation de la détection des plug-in's

Nous avons donc implémenté une méthode particulière pour chaque type de document. Dans le code HTML généré, nous avons inclus des structures de contrôle qui ont pour objectifs de vérifier si un plug-in spécifique est installé ou non. Lorsque l'utilisateur affiche une page du cockpit, si celle-ci contient un document multimédia, le code JavaScript contenu dans la page va effectuer plusieurs tests pour vérifier si le plug-

in nécessaire fonctionne sur la machine. Dans le cas où le plug-in n'est pas installé, à la place du document multimédia apparaît une page HTML avec un lien vers le site du fournisseur du plug-in afin qu'il puisse être installé. Dans le cas où le plug-in est détecté, le document multimédia est exécuté normalement.

La gestion des plug-in's varie également selon le navigateur utilisé. Nous pouvons distinguer deux groupes différents. Le premier est celui constitué d'Internet Explorer, quelle que soit la plateforme. Le deuxième regroupe tous les autres navigateurs. En effet, avec Internet Explorer il faut utiliser une solution particulière et nettement plus compliquée.

Par contre, pour les autres navigateurs, il est possible d'utiliser le typage MIME. Le typage MIME correspond à une signature particulière d'un plug-in. Lorsqu'un plug-in est installé, il est possible de le détecter avec ces navigateurs, en vérifiant si le typage MIME correspondant au plug-in est défini dans les spécifications du navigateur.

Malheureusement, pour Internet Explorer, il ne reconnaît pas le typage MIME. Pour détecter l'installation des plug-in's, il faut donc faire appel à une approche totalement différente, en combinant les deux langages de script que sont VBScript (Visual Basic Scripting) et JavaScript. En utilisant ces scripts, nous testons s'il est possible d'instancier un objet particulier correspondant au plug-in qui est testé. S'il est possible de créer un tel objet, nous pouvons considérer que le plug-in est installé ; dans le cas contraire, une exception est levée. Si tel est le cas, il faut alors l'intercepter et en déduire que le plug-in n'est pas installé.

III) Utilisation des plug-in's dans une page HTML

L'intégration d'un objet multimédia dans une page HTML s'effectue par l'utilisation d'un plug-in spécifique. La détection a été précisée ci-dessus, maintenant nous allons décrire comment exécuter ces plug-in's dans une page HTML.

En effet, comme pour la détection, nous devons à nouveau distinguer les deux mêmes groupes, à savoir Internet Explorer et les autres navigateurs. Microsoft, le développeur de Internet Explorer, a développé une gestion différente des plug-in's par rapport aux autres navigateurs. La norme du code HTML prévoit une balise `<EMBED>` spécifique pour l'intégration de composants multimédias, d'une façon toute simple à travers leurs plug-in's. Or depuis la version 6 d'Internet Explorer, Microsoft a supprimé le support de cette balise pour la remplacer par un autre concept, portant la dénomination d'*objets ActiveX*. Le concept d'objet ActiveX a été mis en œuvre par Microsoft pour concurrencer les applets Java. Dans une page HTML exécutée sous Internet Explorer, ils permettent d'exécuter les composants multimédias de la même manière que la balise `<EMBED>`. Par contre, les objets ActiveX ne sont pas supportés par les autres navigateurs, ce qui nécessite de combiner ces deux concepts pour permettre d'exécuter les composants multimédias sur tous les navigateurs.

IV) Remarques

a) Composants SVG

SVG [33], pour Scalable Vector Graphics, est un langage permettant la description des graphiques bidimensionnels en XML. SVG se place comme un concurrent à la technologie Flash développée par Macromedia [34], il est recommandé par le W3C [25] pour le traitement graphique sur le Web. Pour fonctionner, il nécessite un plug-in fourni par l'entreprise Adobe [35]. Malheureusement, ce plug-in ne fonctionne que sur Internet

Explorer et non pas sur les navigateurs de la famille Netscape/Mozilla/Firefox. En effet, le projet Mozilla prévoit un support natif de la norme SVG, dans le code même du navigateur. Cependant, ce support natif n'est pas encore au point à l'heure actuelle, et les composants de types SVG sont donc encore mal gérés par ces navigateurs.

b) Composants SMIL

SMIL [36], pour Synchronized Multimedia Integration Language, est un langage de création de présentation multimédia basé sur XML et normalisé par le W3C. Son objectif est d'associer audio, vidéos, images et textes en un seul document. Un composant SMIL peut être exécuté avec le plug-in QuickTime Player [37]. Malheureusement, son comportement ne correspond pas toujours au contenu formel de ce qui a été défini dans le code SMIL. Ceci est dû au fait que les normes gravitant autour du standard XML évoluent tellement rapidement, que les outils permettant de les exploiter ont constamment un temps de retard sur la norme en vigueur ! Une solution pour remédier à ce problème consiste à convertir un document SMIL en une vidéo. Ainsi, par transposition, l'exécution d'un composant SMIL se réduirait à la simple exécution d'une vidéo.

3.2.3 Outils complémentaires

L'environnement *eMersion* étant en constante évolution, il a été demandé d'effectuer quelques recherches supplémentaires, en développant des outils qui pourraient servir dans le cadre général de l'environnement *eMersion*. Ces différents outils ne seront pas forcément intégrés dans le cadre strict du projet présenté ici, mais pourraient l'être dans l'idée d'une évolution future d'*eMersion*. Nous allons donc décrire trois outils ayant un objectif bien particulier. Le premier permet de calculer la vitesse de connexion de l'utilisateur vers un serveur particulier. Le deuxième outil permet de créer des images à partir d'un texte. Le troisième redimensionne la taille d'images.

I) Calcul de la vitesse de connexion au serveur

Nous pouvons nous demander quelle serait la finalité d'un tel outil. L'idée finale de cet outil serait de fournir un ordre de grandeur du type de connexion depuis laquelle l'utilisateur se connecte au serveur. Ceci serait fort utile dans le cas où le serveur fournirait plusieurs types de qualité de vidéo par exemple. Nous pouvons très bien imaginer, comme cela se fait régulièrement aujourd'hui sur divers site, que l'environnement *eMersion* puisse fournir des accès à des vidéos à ses utilisateurs. Dans ce cas, il serait possible de fournir des vidéos de qualités différentes, en fonction de la vitesse de connexion entre le serveur et l'utilisateur. Nous sommes partis dans l'idée de mettre en œuvre un outil simple, demandant peu de ressources et fournissant une réponse dans un temps court.

Afin de réaliser cet objectif, nous nous sommes basés sur un exemple de script fournit par www.toutjavascript.com [3], et nous avons réalisé un simple programme en JavaScript qui s'exécute directement depuis une page HTML. Il va charger une image depuis le serveur sur l'ordinateur. Connaissant la taille exacte de l'image, il est alors possible de mesurer la vitesse de connexion en mesurant le temps mis pour charger cette image. Afin de rendre le résultat plus précis, nous pouvons envisager de répéter cette

opération plusieurs fois, ceci dans le butde mesurer une moyenne plutôt qu'une valeur unique.

Bien entendu, il ne s'agit que d'une photographie instantanée de l'état du réseau. Il faut donc rester prudent avec les valeurs obtenues. Toutefois, cela peut s'avérer utile dans le cas d'une détection approximative pour évaluer s'il s'agit d'un réseau à haut débit ou non. Dans ce cas, les valeurs obtenues sont largement suffisantes.

II) Génération d'image à partir de texte

L'objectif du développement d'une telle application serait de générer dynamiquement des boutons particuliers, afin de traiter des liens hypertextes comme des images. Pour cela, nous allons générer des images au format PNG. En effet, le format PNG, au contraire du format GIF, est un format ouvert pour lequel il existe de nombreux outils qui permettent de gérer facilement des images de ce format. Par ailleurs ce format est bien toléré par les différents navigateurs, à la nuance près qu'Internet Explorer ne gère pas de manière optimale les effets de transparence. Cependant, la qualité des images rendues avec le format PNG est de qualité nettement supérieure pour une taille nettement inférieure au format GIF.

La classe `Txt2PngFactory.java` [4] effectue les opérations de générations d'image. Pour créer des images avec cette classe, nous utilisons la classe exécutable `CreateImage.java` qui prend en paramètres le texte, le nom de la police et les couleurs de l'image à générer.

III) Redimensionnement des images

Le redimensionnement des images a pour objectif d'ajuster la taille des images insérées dans le cockpit en fonction de leur taille réellement occupée dans le navigateur. Cet outil prend en paramètre une image de format GIF ou JPG en entrée et la transforme en une image du même type, mais dans les nouvelles dimensions. Cependant, cet outil ne pourra pas être intégré dans l'outil auteur. En effet, dans l'outil auteur, la taille des images n'est pas traitée en pixels, mais en pourcentage de la surface occupée par rapport à la taille totale disponible. Cette approche de développement des cockpits apporte une maniabilité bien plus grande de la manipulation du cockpit, étant donné que la taille des images s'adaptera toujours par rapport à la taille totale de l'environnement, quelque soit la résolution de l'écran de l'utilisateur ou la taille du redimensionnement du navigateur.

Il est important de préciser que les fichiers enregistrés en format GIF utilisent une librairie non standard, car le format GIF est un format propriétaire protégé pour lequel il n'existe pas de librairie officielle permettant l'écriture de tels fichiers. Avant d'effectuer toute transformation sur une image, le format de celle-ci est testé pour savoir s'il respecte la structure des formats JPG ou GIF. Ceci permet un traitement correct des fichiers selon leur format.

Le programme `Resize.java` créé par R. Belknap [7] a été adapté pour gérer les images de type GIF. Cette classe utilise la librairie `ImageEncoder.jar` qui a été générée à partir de la librairie permettant de gérer des images, et fournie par Acme [5]. Nous y avons également rajouté `ImageInfo.java` [6], l'outil créé par M. Schmidt. Cette libraire permet de vérifier que l'image passée en paramètre correspond bien au bon format d'image. Ce n'est pas l'extension (*.gif, *.jpg) qui est testée, mais bien la structure interne du fichier image. La librairie récupérée d'Acme contient également une API pour écrire une image au format *.gif. Ce format est un format commercial pour lequel il n'existe aucune API '*open source*' officielle.

Le programme `Resize.java` prend pour paramètres le nom de l'image source, le nom de l'image que l'on veut créer avec son extension. Ensuite, il est possible d'insérer une seule ou deux valeurs. Si une seule valeur est insérée, il s'agira du facteur de redimensionnement de l'image, et ses proportions seront conservées. Par contre, si deux valeurs sont insérées, ce seront les nouvelles dimensions de l'image, et ses proportions ne seront pas conservées.

3.2.4 Conclusion

Nous avons décrit, dans les lignes ci-dessus, les raisons pour lesquelles nous avons dû prendre en considération des plug-in ainsi que leur nécessité pour prendre en charge des environnements Web multimédias. Nous avons également vu comment détecter si un plug-in était installé ou non sur une machine, et le comportement dans les deux cas. Nous avons également remarqué les subtilités introduites dans Internet Explorer, par Microsoft, qui pousse à imposer ses propres normes au détriment de l'uniformisation des normes proposées par le W3C.

Dans un deuxième temps, nous avons présenté quelques outils qui ont été développés durant ce projet, mais qui ne lui serviront pas directement. Ils pourraient cependant s'avérer utiles dans le développement futur de l'environnement *eMersion*.

3.3 Transformation XSLT

3.3.1 Introduction

Une fois que la gestion des différentes caractéristiques multimédias a été achevée, il a fallu adapter la feuille de style utilisée pour effectuer les transformations XSLT. Comme nous l'avons précisé plus haut, la feuille de style générant le cockpit supportait jusqu'à présent des conteneurs ayant des images, des hyperliens ou des documents structurés de type HTML. Or, le but final de ce projet étant bien de générer les cockpits créés par l'outil auteur, il est donc primordial de généraliser la feuille de style à toutes les caractéristiques gérées par l'outil auteur. Pour cela, nous allons décrire ci-dessous comment nous avons intégré les différents composants dans la feuille de style. Nous y intégrerons la gestion des plug-in's présentée au chapitre précédent.

3.3.2 Remarques

I) Gestion de l'espace occupé par les objets multimédias

Les différents types d'objets multimédias, décrits dans ce chapitre, occupent un espace qui leur est attribué lors de la conception avec l'outil auteur. Afin de rendre une maniabilité optimale lors de l'utilisation, les dimensions de ces objets ne sont pas précisées en pixels, mais en pourcentage de la taille occupée dans le conteneur. Lors des différentes phases de conception de la feuille de style, nous avons rencontrés des problèmes pour reproduire les tailles de ces objets dans une page HTML. En effet, les applets, les éléments en Flash ou les vidéos ne supportent pas les attributs de largeur et de hauteur exprimés en pourcentage. Il a donc fallu trouver une autre approche pour générer un rendu fidèle au cockpit conçu avec l'outil auteur. Ainsi, nous avons opté pour reproduire les dimensions en pourcentage de ces éléments dans des balises HTML de type `<DIV>`. Ces balises ont l'avantage d'offrir une flexibilité très vaste. Nous pouvons ainsi les positionner au pixel près, et leur indiquer une taille spécifique en pourcentage par rapport à leur élément père. Dès lors, nous avons insérés des balises de type `<DIV>` dans le conteneur à la place des objets multimédias. Ensuite, nous avons inséré les objets multimédias dans les balises `<DIV>` en leur indiquant d'occuper toute la place à disposition. Cette manière d'agir nous garanti un rendu parfaitement fidèle au cockpit construit avec l'outil auteur.

II) Superposition de composants multimédias

Lors du développement de la feuille de style XSLT, nous avons été confronté à un comportement surprenant de certains composants multimédias. En effet, la façon dont a été conçue la gestion des conteneurs permet de les déplacer à volonté. Cela permet également de superposer différents conteneurs. Lorsqu'une telle opération est effectuée avec des conteneurs contenant des applets, des vidéos ou des documents Flash, la superposition des différentes couches n'est pas gérée de manière optimale par les navigateurs. En effet, les composants multimédias restent systématiquement visibles ! Ainsi, si un conteneur vient se positionner sur une vidéo, le comportement logique voudrait que la vidéo soit masquée par le conteneur ! Malheureusement, ce n'est pas le cas, la vidéo restera visible au détriment du conteneur sensé la recouvrir ! Cet inconvénient a pu être corrigé en ce qui concerne la gestion des éléments en Flash car la

norme Flash permet d'introduire un paramètre de transparence offrant la gestion d'une telle situation. Malheureusement, concernant les vidéos et les applets, aucun réglage de ce genre n'est fournit par les plug-in's qui les exécutent !

3.3.3 Gestion des applets

Une applet est une application écrite en Java qui peut être intégrée directement dans un document HTML. Une applet nécessite que le plug-in Java soit installé sur la machine de l'utilisateur. Si le test est concluant, l'applet est exécutée. Pour traiter une applet, il est nécessaire de préciser le nom de la classe principale d'exécution de l'applet, ainsi que le nom du ficher d'archive JAR contenant l'ensemble des classes compilées de l'applet, ainsi que tous les différents fichiers référencés par l'applet tels que les images utilisées comme icônes.

Bien que le standard HTML contienne une balise `<APPLET>` pour intégrer une applet dans du code HTML, nous avons décidé de gérer les applets comme les autres objets multimédias, en utilisant la balise `<OBJECT>` dans le cas d'Internet Explorer et la balise `<EMBED>` pour les autres navigateurs.

3.3.4 Gestion des vidéos

Concernant la gestion des vidéos, nous utilisons deux plug-in's différents que sont QuickTime Player et Windows Media Player. Nous avons volontairement écarté la prise en charge des vidéos par le plug-in Real Media Player. En effet, ce lecteur de vidéos nécessite une configuration particulière dans les paramétrages contenus dans le code HTML. Avec ce plug-in, il n'est pas possible d'effectuer une référence directe à une vidéo présente sur le disque. Pour lire une vidéo quelconque, il est nécessaire de référencer un fichier externe qui contient le lien vers la vidéo à visionner. Ce procédé s'avère très fastidieux dans notre démarche de génération dynamique de l'environnement par les feuilles de style XSLT. Par contre, les deux autres plug-in's utilisés (QuickTime Player et Windows Media Player) permettent un référencement direct aux vidéos à visionner, sans passer par ce fichier intermédiaire.

Nous avons donc partagé la prise en charge des différents formats par ces deux plug-in's. Les formats de vidéos AVI et WMV sont pris automatiquement en charge par Windows Media Player, alors que les formats MOV et MPEG sont pris en charges par QuickTime Player.

3.3.5 Gestion des composants audio

La gestion des composants audio est très semblable à la gestion des composants vidéo. Les formats pris en charges par le cockpit sont les formats MP3, WAV et AIFF. Etant donné que le plug-in QuickTime Player permet de gérer tous ces formats, nous gérons les composants audio uniquement avec ce plug-in.

Une différence notable des composants audio par rapport aux composants vidéo se situe dans l'absence d'image ! Or, lors de la conception du cockpit, l'insertion d'un composant audio revient à lui attribuer une surface dans un conteneur. La question s'est donc posée de savoir comment représenter un tel composant qui n'a pas de caractéristique graphique ! La solution est venue du fait qu'il fallait insérer une composante graphique de gestion du son, à travers une barre de défilement. Le plug-in QuickTime Player met à disposition une barre de défilement avec l'ensemble des boutons permettant de démarrer le son, de l'arrêter ou de le mettre sur pose [Figure 3-5].

Nous avons donc exploité cette composante graphique pour représenter un composant audio dans l'espace qui lui est attribué dans un conteneur.

Cependant, la représentation d'un tel objet graphique pose certaines limites techniques et esthétiques. A savoir, il ne serait pas très agréable de voir à l'écran une grosse barre épaisse qui occuperait toute la hauteur disponible. Pour cela, nous avons décidé d'étendre la largeur de la barre de défilement sur toute la largeur de l'espace à disposition, mais de garder une hauteur fixe, quel que soit la hauteur de l'espace à disposition !

Figure 3-5 Représentation graphique d'un composant de type audio

3.3.6 Gestion des composants Flash

Un composant Flash est un composant contenant des animations vectorielles. Ces composants nécessitent l'utilisation du plug-in fournit par Macromedia. Comme indiqué dans la remarque ci-dessus, il a été possible de définir un paramètre de transparence dans le code HTML d'invocation de ce type de composant. Ce paramètre permet de supporter la superposition de tels composants par d'autres conteneurs.

Lors de phases de tests, il a été constaté que l'exécution de ces composants peut provoquer quelques conflits avec le navigateur Firefox si le plug-in QuickTime est installé en parallèle du plug-in de Macromedia. En effet, le plug-in QuickTime permet également d'exécuter les composants Flash. Or s'il est installé sur la machine de l'utilisateur, Firefox va favoriser ce plug-in au détriment du plug-in officiel de Macromedia. Malheureusement, QuickTime ne supporte pas aussi bien les composants Flash que le plug-in officiel, ce qui peut provoquer dans certains cas des comportements aléatoires de l'animation Flash. Ce problème n'a pas été constaté avec les autres navigateurs.

3.3.7 Gestion des documents structurés

La structure qui a été déterminée pour générer les cockpits offre également la possibilité s'insérer d'autres documents à l'intérieur du cockpit. Ces éléments sont des documents HTML, SMIL, SVG ou XML.

Concernant l'insertion de document HTML, la solution adoptée est toute simple, puisqu'il s'agit simplement d'insérer une référence à ce document HTML externe qui doit être contenu à l'intérieur d'un conteneur. Pour les documents de type SVG ou SMIL, ils sont traités par des plug-in's. Toutefois, comme nous l'avons déjà précisé plus haut, la gestion des documents de type SVG par Firefox n'est pas encore au point. En effet, Firefox ne supporte pas le plug-in fournit par Adobe, car le projet Mozilla qui développe ce navigateur souhaite gérer ce format de manière native directement à l'intérieur de son propre code. Malheureusement, cette prise en charge native n'est pas encore au point à l'heure actuelle.

I) API de gestion des documents structurés de type XML

Comme indiqué, il est également possible d'insérer des documents XML dans un conteneur. En effet, en liant un document XML à une feuille de style XSLT, il est possible de générer un document HTML. Dès lors, si un document XML est inséré, nous allons le traiter comme s'il s'agissait d'une page HTML. Nous référençons la page HTML qui sera générée au moment de la publication de l'environnement.

Lorsque le concepteur du cockpit insère un document XML, il doit également fournir une feuille de style pour la transformation XSLT. Il a aussi la possibilité de fournir un Schéma-XML ou une grammaire DTD [38] du fichier XML pour effectuer une validation du fichier XML. Ainsi, lorsque un document structuré de type XML est inséré, nous nous trouvons avec deux ou trois fichiers, si l'utilisateur souhaite valider son document XML. Etant donné que dans le conteneur nous ne référençons pas directement le fichier XML, mais un fichier HTML issu d'une transformation XSL, il nous faut gérer la génération de ce document HTML.

Cette partie de génération du document HTML est gérée directement sur le serveur lors du déploiement du cockpit. En effet, nous avons créé une API dédiée au serveur qui s'exécute avant la génération globale du fichier XML du cockpit. Cette API va parcourir le document XML du cockpit à la recherche de documents structurés de type XML. Si de tels documents sont détectés, l'API va exécuter une validation du document XML si un Schéma-XML ou une DTD y est rattachée. Si tel n'est pas le cas, ou si la validation s'est déroulée avec succès, la phase de génération de ce document structuré est lancée pour produire le document HTML référencé par le cockpit. Si la phase de validation échoue, une erreur est générée. Finalement, lorsque la génération du document HTML est achevée, l'API cède la place au serveur pour qu'il puisse poursuivre la génération du cockpit dans son ensemble.

La classe `StructDocManager.java` est la classe principale de l'API qui gère toutes les opérations que sont la recherche de documents structurés de type XML, la validation du document XML ou encore la transformation XSLT. La classe `StructDocFinder.java` est chargée de rechercher dans le document XML de description du cockpit si des documents structurés de type XML s'y trouvent. La classe `XMLValidator.java` se charge de la validation du document XML avec le Schéma-XML ou la DTD correspondante. Finalement, la classe `XSLTransform.java` effectue la génération du document HTML cible.

La figure 3-6 représente le fonctionnement de l'API de gestion des documents structurés de type XML sous la forme d'une machine d'états.

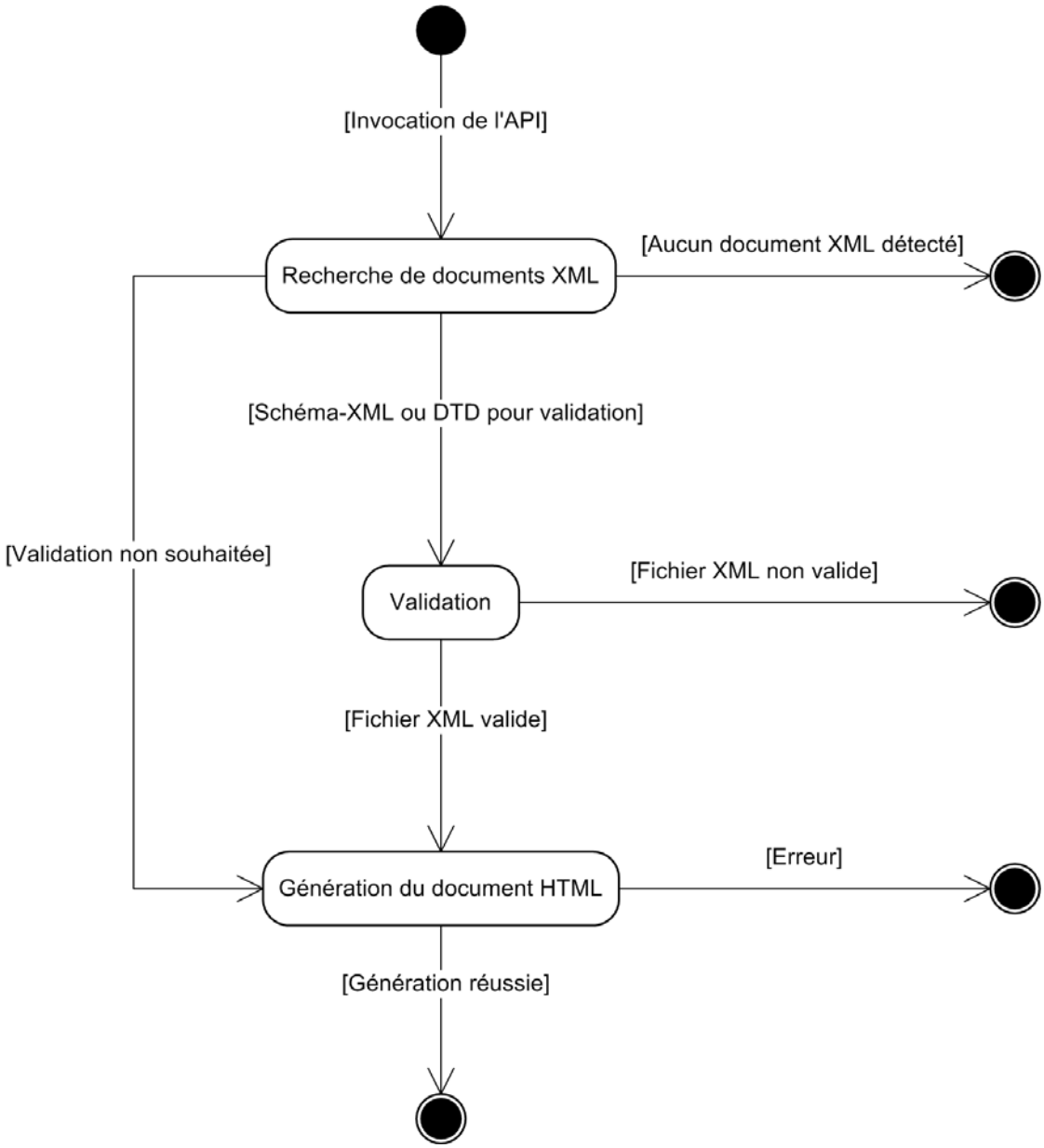


Figure 3-6 Structure de l'API de gestion des documents structurés de type XML

3.3.8 Conclusion

Nous avons décrit dans cette section la manière dont nous avons géré la transformation XSLT qui permet de générer le cockpit. Pour cela, nous avons traité de la manière dont nous devons reproduire les différents composants multimédias pouvant être insérés dans un cockpit. Nous avons pour cela présenté la notion de plug-in nécessaire à la gestion de ces éléments multimédias. Nous avons également présenté la façon dont nous avons traité l'occupation de l'espace à disposition de ces éléments dans les différents conteneurs, notamment l'approche utilisée pour représenter des composants de type audio. Finalement, nous avons présenté le traitement de composants structurés, parmi lesquels nous nous sommes attardés sur la conception d'une API Java qui permet de détecter et de générer les documents structurés de type XML.

3.4 Elaboration de l'outil de publication

3.4.1 Introduction

Nous allons décrire ci-dessous la manière dont nous avons géré le déploiement du cockpit sur le serveur, du chargement du fichier d'archive à l'affichage du nouveau cockpit dans le navigateur. Cette partie correspond véritablement à l'aboutissement des différents concepts présentés dans les chapitres précédents. En effet, sans une base fonctionnelle et efficace, il aurait été utopique de se lancer dans la partie de conception côté serveur. De ce fait, en pouvant compter sur une base solide que nous avons décrite jusqu'à présent, cette partie peut être développée de façon assez brève puisqu'il s'agit de la suite logique de tout ce qui précède.

Après une brève présentation de l'architecture globale du serveur, nous expliquerons comment le chargement du cockpit s'effectue sur le serveur, et comment il est décompressé. Nous expliquerons ensuite les différentes phases de contrôle et d'analyse du fichier XML de description du cockpit, suivie de la transformation XSLT. Finalement nous préciserons comment est gérée la structure hiérarchique des répertoires, aboutissant à l'affichage du nouveau cockpit généré.

3.4.2 Architecture globale du serveur

L'application de déploiement a été développée en vue d'être déployée sur un serveur de type Tomcat. « *Tomcat est l'un des composants du projet Jakarta, dont le but est de fournir des solutions serveur, basées sur la plateforme Java, de qualité identique aux applications commerciales mais développées avec l'esprit d'ouverture du logiciel libre.*

Tomcat est donc un moteur de servlets qui a été conçu en suivant le guide de référence officiel de l'implémentation des technologies Java Servlet et Java Server Pages (JSP). Les spécifications de ces dernières sont élaborées par Sun, selon le processus établi par la Java Community.

Les JSP sont des pages contenant du code Java imbriqué dans du HTML. Cette approche est similaire à celle de l'intégration PHP/HTML. » [9]

Selon la définition qui précède, Tomcat correspond à un serveur Web permettant de traiter des servlets et des pages JSP. Une servlet est une application Java s'exécutant sur un tel serveur et permettant de communiquer avec un navigateur Web en suivant le protocole HTTP. Une page JSP combine la force d'une servlet en offrant l'avantage d'une gestion de la mise en page suivant la norme HTML. Il est ainsi possible de créer des documents JSP fournissant un support Web pour les navigateurs, et très facile à mettre en place. D'ailleurs, lors de la première invocation d'une page JSP, le serveur Tomcat va compiler la page, et générer les servlets correspondantes. Nous pouvons donc remarquer la similitude entre servlets et JSP, les servlets étant favorisées si une forte partie de programmation est nécessaire, alors que les JSP sont plutôt utilisées si le rendu HTML doit être favorisé. Notre application serveur va donc combiner ces deux différents concepts, en fonction de l'objectif à atteindre par un module particulier.

Nous présentons ci-dessous point par point les différentes étapes de déploiement d'un nouveau cockpit, en suivant l'architecture présentée dans la figure 3-7.

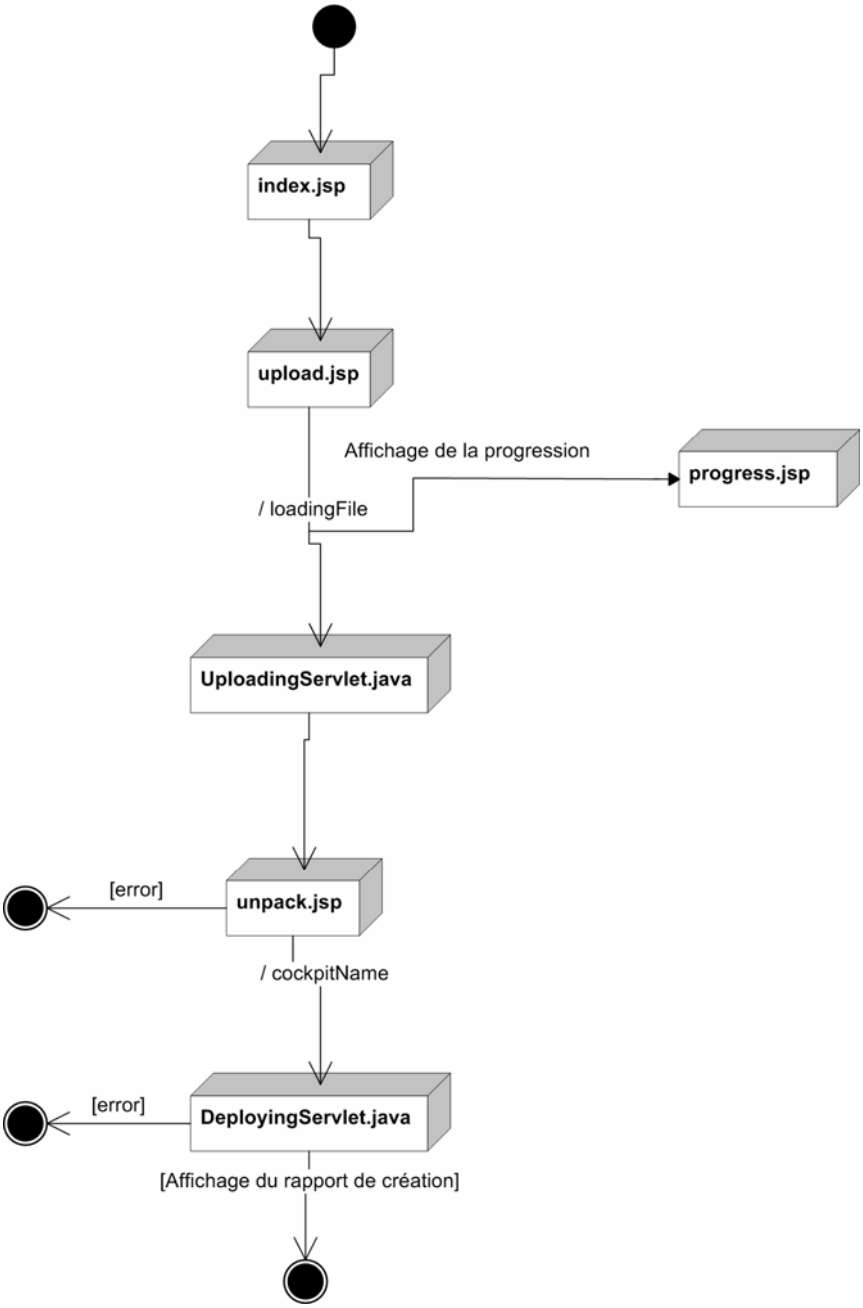


Figure 3-7 Architecture du serveur

I) Chargement de l'archive du cockpit

La conception de l'application serveur avait pour objectif d'être totalement indépendante de l'outil utilisé pour créer le cockpit. Certes, dans notre cas, nous avons pu exploiter le fait de posséder un outil auteur qui crée le fichier XML de description du cockpit, en nous fournissant aussi tous les fichiers liés à ce cockpit directement dans une archive. Cependant, nous aurions très bien pu imaginer créer le fichier XML à la main, de lui attacher tous les fichiers liés, et d'insérer le tout dans une archive ; ou alors simplement d'utiliser un autre outil de conception du cockpit, pour finalement transmettre cette archive au serveur afin de la déployer. Nous nous sommes donc fondés

sur ce principe fondamental de distinction entre les deux outils. Cette approche conduit donc l'utilisateur à sélectionner manuellement l'archive du cockpit à déployer depuis sa propre machine.

Cependant, dans une approche plus pratique, nous avons toutefois tenté de gérer le chargement directement depuis l'outil auteur. Nous nous sommes alors heurté à des problèmes de sécurité. En effet, Tomcat étant un serveur suivant le protocole HTTP, il n'est pas possible de charger des éléments de manière indépendante sans passer par des requêtes HTTP. En effet, pour des raisons de sécurité, il est nécessaire de gérer le chargement depuis une page JSP par l'utilisation d'un formulaire HTML. La page de démarrage que nous avons créée pour le chargement du cockpit accepte, comme paramètre facultatif, un nom de cockpit fournit avec son chemin absolu. En exploitant ce paramètre, nous avons tenté de l'insérer directement dans le champ de sélection du cockpit. Nous nous sommes heurtés aux mêmes problèmes de sécurité. En effet, afin d'éviter le chargement abusif de fichiers depuis un client vers un serveur, il n'est pas possible de définir une valeur par défaut à ce champ. Seul l'utilisateur a le droit de sélectionner un fichier se trouvant sur son disque pour le transférer vers le serveur.

Toutefois pour faciliter la tâche du concepteur du cockpit lorsqu'il le déploie directement depuis l'outil auteur, nous transmettons le nom du cockpit à l'application de déploiement. Comme le cockpit à déployer ne peut pas être inséré directement dans le champ de sélection du formulaire HTML, nous affichons textuellement le chemin du cockpit à générer dans le navigateur. Ainsi l'utilisateur n'a plus qu'à sélectionner ce cockpit pour le générer. Il est important de préciser que le paramètre du nom du cockpit pris en charge par la page de chargement est facultatif. De cette manière, si l'utilisateur se connecte manuellement au serveur, sans passer par l'outil auteur, la même page est affichée, à la différence qu'aucun nom de cockpit à charger n'est affiché à l'écran.

Au démarrage, en se connectant sur la page `index.jsp`, l'utilisateur est directement redirigé sur la page `upload.jsp`. Cette page offre la possibilité à l'utilisateur de sélectionner un cockpit à charger depuis son ordinateur à travers un formulaire HTML. Par ailleurs, si le démarrage du navigateur s'est fait depuis l'outil auteur, le nom du cockpit et son chemin sont affichés. Lorsqu'une archive ZIP est sélectionnée, le chargement peut démarrer. Celui-ci est géré par la servlet `UploadginServlet.java`. Durant le chargement, une fenêtre informant la progression du chargement s'affiche, gérée par la page `progress.jsp`.

II) Décompression de l'archive

Une fois le chargement de l'archive sur le serveur achevée, la décompression de ce document est gérée par la page `unpack.jsp`. Par ailleurs, si des erreurs se sont produites lors du chargement de l'archive, elles sont affichées par cette page et le processus de déploiement est interrompu. De plus, cette page affiche le nom qui sera attribué au cockpit en fonction du nom de l'archive. Toutefois, nous avons offert la possibilité à l'utilisateur de modifier ce nom s'il le désire. Lorsque l'utilisateur accepte cette page, la phase de décompression et de déploiement est totalement prise en charge par la servlet `DeployingServlet.java`. Dès cet instant, plus aucune intervention de l'utilisateur n'est nécessaire jusqu'à la réussite ou l'échec du déploiement. Tout au long des opérations gérées par cette servlet, des informations sont recueillies, concernant les différentes opérations en cours ou les erreurs qui se produiraient. Ceci permet d'afficher un rapport d'exécution lorsque les opérations se sont achevées.

La première opération gérée par cette servlet concerne la décompression de l'archive dans le répertoire courant de l'application. Ceci est géré par la classe `Unzip.java`.

III) Génération des documents structurés internes

Lorsque tous les fichiers nécessaires au déploiement ont été décompressés avec succès, l'étape suivante consiste à parcourir le document XML de description du cockpit, à la recherche de documents structurés de type XML. Cette opération est toujours prise en charge par la même servlet `DeployingServlet.java`. Nous ne nous attarderons pas plus longtemps sur cette opération car elle a déjà été amplement décrite dans ce chapitre, à la section traitant de l'*API de gestion des documents structurés de type XML [3.3.7.I]*. Comme nous l'avons décrit dans ce paragraphe, le but de cette opération est de rechercher des documents structurés de type XML, dans la description XML du cockpit. En effet, ces documents sont gérés comme des documents de type HTML situés dans un conteneur. Cependant, ces fichiers HTML n'existant pas encore, ils sont générés à ce moment précis du déploiement.

IV) Exécution de la transformation XSLT

Lorsque la recherche de documents structurés internes de type XML s'est achevée avec succès, le processus de déploiement se poursuit par la génération finale des pages HTML composant le nouveau cockpit et ses conteneurs. De même que ci-dessus, cette phase a déjà largement été commentée dans la section 3.3 intitulé *Transformation XSLT*, et nous ne nous attarderons pas plus longtemps sur ce sujet.

3.4.3 Affichage du cockpit et structure hiérarchique des répertoires

Une fois que toutes les transformations se sont accomplies avec succès, la hiérarchie du cockpit est copiée dans le répertoire source du serveur Tomcat. En effet, la structure des répertoires d'un serveur Web fonctionne sur deux niveaux. Le premier niveau correspond à la structure locale des répertoires telle qu'elle se présente dans l'organisation des fichiers du disque. Le deuxième niveau correspond à la structure des répertoires gérée par le serveur qui doit représenter une structure correspondant à celle interprétée par les navigateurs Web. A cet effet, un répertoire nommé '/webapps', dans la hiérarchie locale des répertoires du serveur Tomcat, correspond au répertoire racine accessible par un navigateur. Tous les sous-répertoires et autres fichiers contenus dans ce répertoire sont accessibles par un navigateur Web. Ainsi, ce répertoire '/webapps' correspond à la source du domaine accessible par le serveur ; c'est dans ce répertoire qu'est copié notre nouveau cockpit. Ainsi par exemple, si nous avons nommé notre cockpit 'mon_cockpit', le répertoire source '/webapps' contiendra un sous-répertoire sous la forme '/webapps/mon_cockpit'. Ce sous-répertoire contiendra toute la hiérarchie générée pour notre nouvel environnement.

Le fait d'organiser ainsi la structure des répertoires permet de référencer automatiquement le nouveau cockpit par le serveur Tomcat. En effet, dans le cas où des servlets sont générées comme c'est le cas si nous gérons des services Web, concept qui sera traité dans le dernier chapitre de ce rapport *[4.2]*, ces servlets doivent se trouver dans un sous-répertoire spécifique, afin que le référencement automatique géré par le serveur Tomcat puisse les retrouver. Cette manière d'agir a aussi pour avantage que notre nouveau cockpit pourra être référencé directement par son nom dans le domaine

géré par le serveur. Ainsi, en poursuivant l'exemple présenté ci-dessus, si notre serveur Tomcat correspond au domaine www.mon_domaine.com, le répertoire '/webapps' va correspondre à l'adresse 'http://www.mon_domaine.com', et le répertoire '/webapps/mon_cockpit' correspondra à l'adresse 'http://www.mon_domaine.com /_monCockpit'.

Finalement, lorsque la génération sera achevée, le rapport de génération s'affichera sous format html et si des erreurs se sont produites, elles seront également affichées à l'écran. Par contre, si aucune erreur ne s'est produite, un hyperlien vers le nouveau cockpit sera disponible pour permettre à l'utilisateur d'accéder directement au nouveau cockpit.

3.4.4 Conclusion

Nous avons présenté, dans cette section, la manière dont nous avons traité la génération d'un nouveau cockpit sur le serveur. Dans un premier temps, nous avons introduit le lecteur avec les concepts de serveur Tomcat, de servlets et de JSP, en distinguant l'utilité de ces deux concepts en fonction du but de notre application. Dans un deuxième temps, nous avons présenté l'architecture du processus de déploiement d'un nouveau cockpit. Pour cela, nous avons présenté pour quelles raisons l'utilisateur doit sélectionner le cockpit sur son disque, comment l'archive est chargée, puis décompressée, puis dans la dernière étape, comment l'environnement est généré. Finalement, nous avons terminé avec la présentation de la hiérarchie des répertoires gérée par le serveur Tomcat sur le disque dur local du serveur, par rapport à la hiérarchie accessible depuis un navigateur Web.

Nous pouvons donc constater que plusieurs concepts, présentés plus haut dans ce rapport, aboutissent logiquement dans ce chapitre-ci lors de la génération finale de l'environnement. Le fait de se lancer dans la conception de la gestion côté serveur en sachant que les différents modules de génération XSLT étaient au point, et que le comportement des conteneurs du cockpit fonctionnait correctement, nous a permis de nous concentrer sur la conception d'une application serveur qui puisse produire des cockpits directement opérationnels.

4 Communication entre composants actifs

4.1 Communication entre applets

4.1.1 Introduction

Après avoir fini le développement de l'outil de déploiement du cockpit sur le serveur, il a été demandé d'analyser la faisabilité de communication entre plusieurs composants. En effet, lors de la conception du cockpit, il est possible d'y insérer des applets qui correspondent à des applications autonomes, s'exécutant sur une page Web. Etant donné que nous sommes en présences d'applications, celles-ci ne se limitent pas à des aspects purement graphiques ou comportementaux comme les autres types de composants, mais elles gèrent également des variables, et offrent la possibilité d'effectuer des traitements sur celles-ci. L'idée est donc de briser la vision d'une applet comme un composant unique et isolé, mais au contraire, il faudrait la voir comme une partie communiquant avec une autre applet qui, elle-même, pourrait renvoyer des données à la première applet.

Nous allons donc commencer par analyser la problématique telle qu'elle nous a été posée. Ensuite nous présenterons une solution qui permet d'établir une communication entre deux applets par l'intermédiaire d'une servlet. Finalement, nous analyserons une solution permettant à deux applets de communiquer directement et sans intermédiaire.

4.1.2 Problématique

Nous devons d'abord indiquer que parmi tous les types de composants que nous avons la possibilité d'insérer dans un conteneur du cockpit, seule une applet permet véritablement d'effectuer des traitements sur des valeurs passées en entrée. Les autres types de composants tels que les documents HTML ou les animations Flash, ne permettent pas de gérer des données à traiter ou à transformer, et sont donc relativement statiques du point de vu comportemental. En opposition, un applet permet d'exploiter toute la force du langage de programmation Java, pour mettre sur pied une véritable application performante.

Pour cela, il s'avère important de présenter comment fonctionne une applet. Lorsqu'un utilisateur charge un document HTML contenant une applet, cette applet sera chargé sur le disque dur local de l'ordinateur de l'utilisateur. Cette applet va donc s'exécuter en local sur la machine de l'utilisateur, et non pas sur le serveur distant, depuis lequel s'est chargée la page HTML. Ceci a pour résultante, qu'une applet s'exécute réellement comme une application local avec un interfaçage sur la page Web. Afin de garantir une certaine norme de sécurité pour prévenir des abus possibles, une applet ne pourra pas communiquer avec tout le réseau Internet à sa guise. En effet, il a été imposé, qu'une applet ne peut communiquer qu'avec le serveur depuis lequel elle est issue. De plus, une applet utilise la machine virtuelle Java pour s'exécuter. Dans celle-

ci, elle utilise une ressource mémoire dépendante du serveur d'où provient cette applet, ainsi que du répertoire dans lequel se trouvait l'applet sur le serveur. Aussi, pour éviter tout risque de fraude, l'applet ne peut accéder qu'aux variables se trouvant dans cette ressource mémoire de la machine virtuelle. Nous constatons donc que ces contraintes de sécurité limitent fortement la marge de manœuvre possible pour établir une communication entre deux applets. Cependant, il demeure possible d'exploiter cette manière de communiquer avec le serveur et d'accéder aux ressources mémoires.

4.1.3 Une servlet comme intermédiaire

Nous présentons ici une approche particulière que nous avons tentée pour faire transférer des données d'une applet à une autre. Pour cela, nous avons exploité le fait qu'un applet puisse communiquer avec le serveur dont elle est issue. L'idée est donc de mettre en œuvre une servlet particulière, sur le serveur d'origine des applets, qui puisse jouer le rôle d'intermédiaire entre les deux applets. Afin de montrer la faisabilité de cette approche, nous avons implémenté deux applets différentes. La première permet d'écrire un message qui sera transmis à la servlet. La deuxième applet va se connecter à la servlet pour vérifier si un message y est enregistré, et l'affiche si elle en trouve un. Cependant, le fait de devoir utiliser une servlet comme intermédiaire, impose une certaine limitation à cette manière d'agir. D'abord, étant donné qu'une servlet fonctionne selon le protocole HTTP, elle se conforme au schéma requête/réponse pour établir une communication avec le client. Cela signifie qu'une servlet ne peut pas établir spontanément une connexion avec un client pour lui transmettre une donnée, mais elle agit toujours en réponse à une requête du client. Ainsi, dans notre exemple présenté ci-dessus, la servlet ne peut pas transmettre spontanément le message reçu par l'applet émettrice à l'applet réceptrice. Pour que le message puisse être affiché, il est nécessaire que l'applet réceptrice effectue une requête envers la servlet pour vérifier si un nouveau message y a été transmis.

Malheureusement, l'applet réceptrice n'a pas de moyen précis pour savoir si un message se trouve ou non sur la servlet. Pour remédier à se problème, il s'avère donc nécessaire à cette applet d'effectuer une requête à intervalles réguliers vers la servlet. Ce comportement n'est de loin pas idéal, car la plus grande majorité des connexions effectuées seront infructueuses et utiliseront inutilement des ressources réseaux. Par ailleurs, cette approche amène d'autres limitations car, tout d'abord, les deux applets doivent obligatoirement être issues du même serveur, étant donné qu'une applet ne peut communiquer qu'avec son serveur d'origine. D'autre part, cette approche nécessite une intervention direct dans le code même des applets, puisqu'il est nécessaire d'insérer le nom du serveur et celui de l'applet dans les paramètres de connexion des applets à la servlet.

La figure 4-1 représente l'architecture de ce qui vient d'être décrit, où deux applets communiquent par l'intermédiaire d'une seule servlet. Cette figure indique également l'ordre d'exécution des différentes actions.

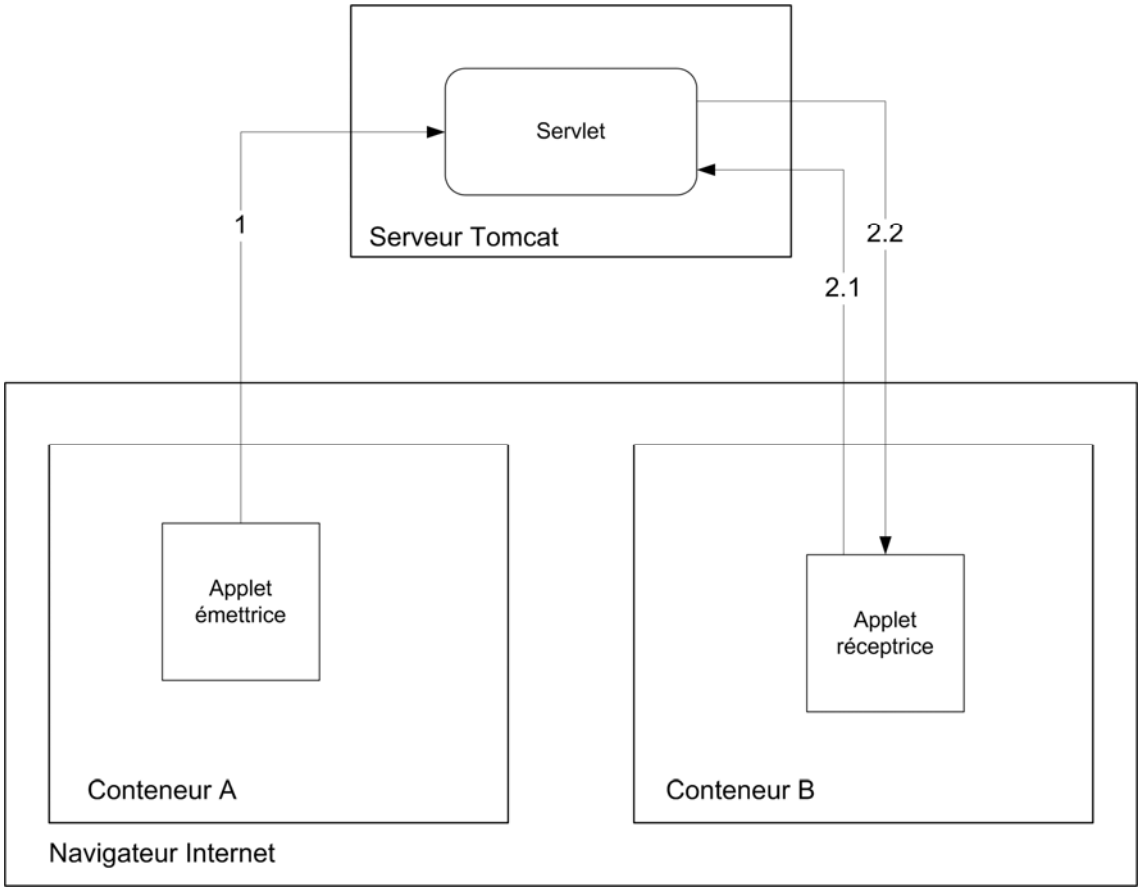


Figure 4-1 Deux applets communiquant par l'intermédiaire d'une seule servlet

4.1.4 Une servlet par applet

Dans une deuxième étape, afin d'éliminer une contrainte présente dans l'approche consistant à transférer des informations entre deux applets par l'intermédiaire d'une servlet, nous avons lié chaque applet à sa propre servlet. Cette approche permet de relier deux applets chargées depuis deux serveurs différents, via deux servlets différentes. En effet, étant donné qu'une applet ne peut communiquer qu'avec son serveur d'origine, cette approche permet de relier deux applets se trouvant sur des serveurs différents, où les servlets feront transférer les informations entre les deux serveurs. L'architecture de cette nouvelle approche reste sensiblement identique à l'architecture précédente, et les différences ne sont pas visibles lors de l'exécution, puisque l'interfaçage graphique des applets reste le même. En effet, l'unique différence revient en la possibilité de faire communiquer deux applets issues de serveurs différents.

Cependant, dans cette nouvelle approche, l'applet qui reçoit le message n'a toujours pas de moyen de savoir si une nouvelle information a été transmise à sa servlet. Ceci implique, à nouveau, que cette applet doive effectuer des requêtes à intervalles réguliers vers sa servlet pour vérifier l'arrivée d'un nouveau message. Par ailleurs, les paramètres de connexion entre les deux servlets doivent toujours être codés dans le code de la première servlet, afin qu'elle puisse émettre une requête et transmettre le paramètre à la deuxième servlet. Finalement, les applets doivent toujours référencer leur propre servlet dans leur code.

La figure 4-2 représente l'architecture de ce qui vient d'être décrit, où deux applets situées sur deux serveurs différents communiquent par l'intermédiaire de deux

servlets situées respectivement sur chacun des deux serveurs. Cette figure indique également l'ordre d'exécution des différentes actions.

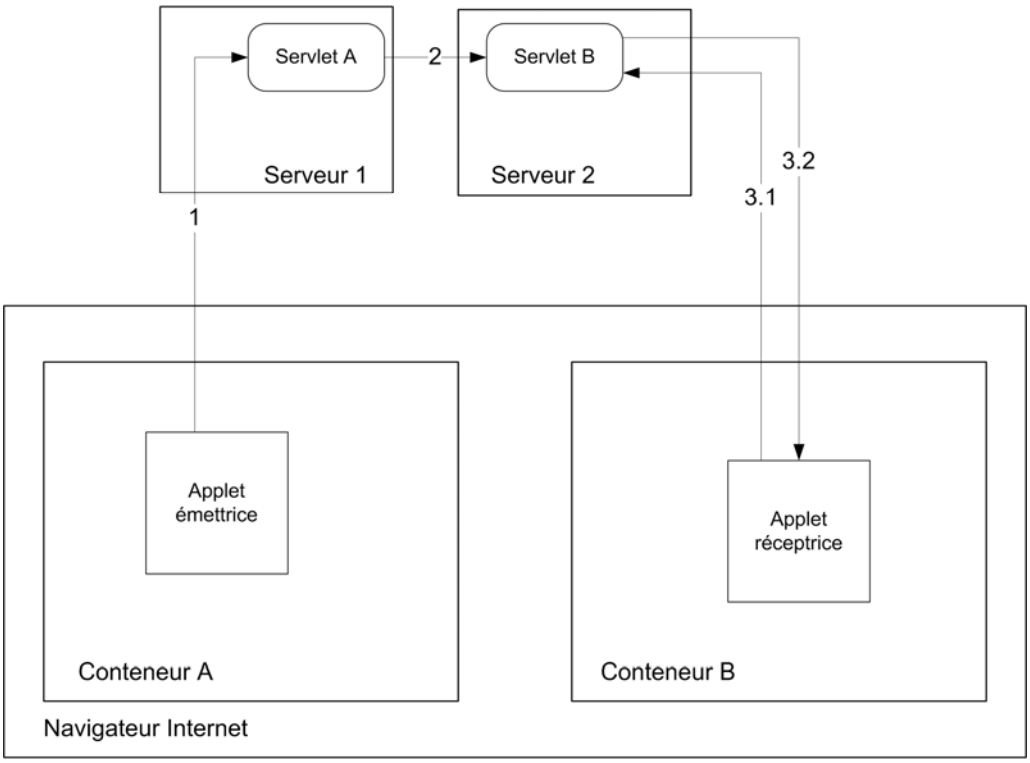


Figure 4-2 Deux applets communiquant par l'intermédiaire de deux servlets

4.1.5 Communication directe entre deux applets

Dans cette approche, nous avons tenté d'éliminer l'utilisation de servlets comme intermédiaires entre deux applets. Pour cela, nous avons exploité le fait que lors de son exécution, une applet place ses ressources mémoires dans la machine virtuelle Java en fonction du serveur dont elle est issue et du répertoire dans lequel elle se trouvait sur ce même serveur d'origine. En exploitant cette caractéristique, nous pouvons imaginer deux applets issues du même serveur et du même répertoire sur ce serveur. Dès lors, lorsque ces deux applets s'exécuteront sur la machine de l'utilisateur, elles utiliseront le même espace mémoire. Cette astuce nous permet alors de retrouver et de manipuler des variables d'une applet par l'autre applet ! En effet, si une applet fournit des méthodes de type public pour accéder à certaines fonctionnalités, ces méthodes seront alors accessibles par l'autre applet moyennant la possibilité de référencer la deuxième applet depuis la première.

Le fait de devoir référencer mutuellement chaque applet nécessite donc que ces références soient insérées directement dans le code des applets. Par exemple, si nous définissons deux classes étendant la classe '`Applet`' et nommée '`applet_A`' et '`applet_B`', chaque applet devra faire référence à elle-même dans la méthode d'initialisation '`init()`'. Ce qui donnerait :

```
public class applet_A extends Applet{
      public static applet_A selfRef=null;
      private applet_B linkedApplet=null;
      (…)

      //méthode d'initialisation de l'applet
      public void init() {
            selfRef=this;     //autoréférencement de l'applet
            (…)
      }

      //méthode de connexion à l'autre applet
      private void sendToLinkedApplet(String messageToSend){
            linkedApplet = applet_B.selfRef;
            linkedApplet.displayNewMessage(messageToSend);
            (…)
      }

      //méthode de gestion d'un nouveau message par une autre applet
      public void displayNewMessage(String newMessage){
            //gestion du paramètre newMessage sur cette applet
            (…)
      }
}
```

Figure 4-3 Strcuture du code des applets pour une communication directe

Dans l'exemple de code de la figure 4-3, nous constatons qu'une applet peut faire référence à une autre applet puisque chaque applet se référence elle-même en tant qu'objet public par la variable '`public static applets_A selfRef`'. Dès lors, nous pouvons faire appel aux méthodes de type `public` des autres applets, car nous récupérons la référence à l'objet `Applet` de l'autre applet par l'instruction '`linkedApplet = applets_B.selfRef`'. De cette manière, nous pouvons manipuler l'applet `applet_B` depuis l'`applet_A` en invoquant la méthode `linkedApplet.displayNewMessage(messageToSend)` qui a été implémentée dans l'`applet_B`.

Afin de démontrer la faisabilité de cette approche, nous avons conçu deux applets selon le modèle que nous venons de décrire. Ces applets permettent d'écrire un message particulier qui va s'afficher dans l'espace prévu à cet effet dans l'autre applet. Les deux applets sont conçues de manière identique, les référencements sont symétriques. Cet exemple nous montre que cette approche fonctionne parfaitement et qu'il n'est pas nécessaire d'utiliser une servlet comme intermédiaire. En faisant l'essai dans le cadre d'un cockpit, nous avons créé deux conteneurs contenant chacun une applet. Ainsi, lorsque nous envoyons des messages d'une applet à l'autre, le fonctionnement est conforme à nos attentes.

Par contre, concernant le codage des applets, nous sommes toujours confrontés à la nécessité d'insérer un référencement dans le code même des applets. Il n'est pas possible de prendre une applet quelconque et de s'imaginer pouvoir agir directement sur son fonctionnement interne par l'intermédiaire d'une autre applet, sans intervention au cœur même du codage de l'applet ! Cependant, nous pouvons partir du principe que si nous souhaitons faire communiquer deux applets différentes, nous pouvons imposer les

principes décrits ci-dessus, afin que ce concept soit pris en compte au moment de l'implémentation des applets.

La figure 4-4 représente l'architecture permettant à deux applets issues du même serveur de communiquer directement, sans intermédiaire.

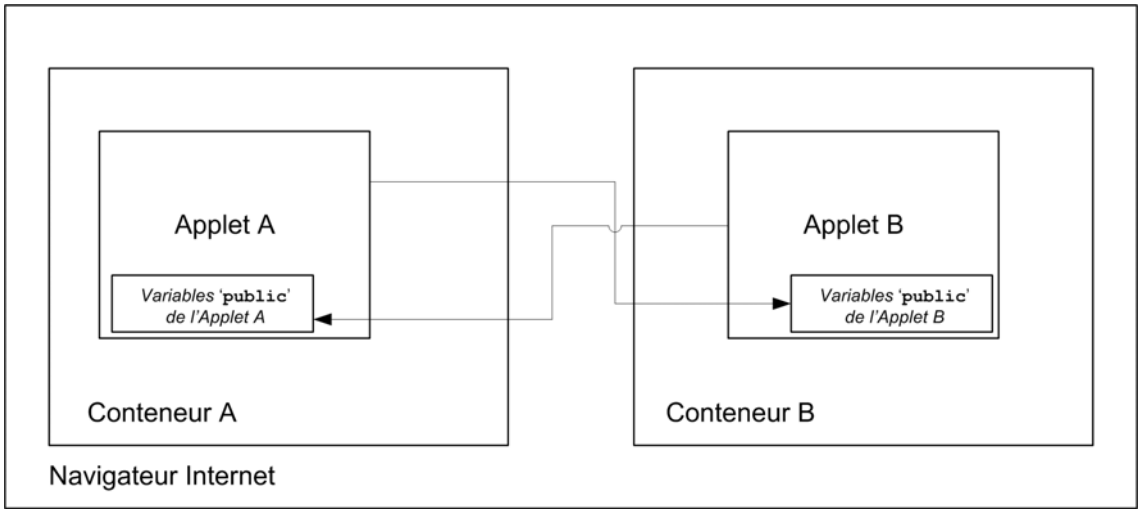


Figure 4-4 Communication directe entre deux applets

4.1.6 Conclusion

Nous avons tenté de présenter, dans cette section, différentes approches pour établir des échanges de données entres plusieurs composants. Pour cela, nous avons traité une telle situation au travers d'applets. Nous avons commencé par présenter la communication de deux applets qui utilisaient une servlet comme vecteur intermédiaire de l'information. Cette approche impose plusieurs limitations, notamment, l'applet qui reçoit l'information ne peut pas être informée directement lorsqu'une nouvelle information lui est destinée. Elle doit au contraire aller chercher cette éventuelle information en invoquant une requête vers la servlet agissant comme intermédiaire. De plus, chaque applet doit nécessairement avoir pour origine le même serveur, puisqu'une applet ne peut communiquer qu'avec son serveur d'origine. Afin de supprimer cette dernière contrainte, nous avons présenté une approche mettant en action une servlet par applet. Cette architecture offre l'avantage de connecter deux applets n'ayant pas forcément le même serveur pour origine.

Finalement, nous avons tenté une approche pouvant se passer d'exploiter des servlets comme intermédiaires. Pour cela, nous avons exploité le partage des ressources mémoires dans la machine virtuelle Java lorsque deux applets sont issues du même répertoire dans le même serveur. Ceci permet d'effectuer un référencement mutuel de chaque applet, et d'intervenir directement sur l'autre applet. De plus, cette approche se différencie des deux approches précédentes, par le fait qu'il n'est plus nécessaire de réveiller à intervalles réguliers une applet pour vérifier si une nouvelle information lui est destinée, car cette nouvelle information peut lui être transmise directement par l'applet émettrice.

Nous avons donc démontré qu'il était possible d'établir une communication entre plusieurs composants, en respectant toutefois un cadre strict, au dehors duquel toute communication ne serait plus possible. Notamment, dans toutes les approches présentées, il est nécessaire d'effectuer un référencement direct des composant à

l'intérieur même du code, que ce soit dans les servlets ou au sein même des applets, l'idéal étant bien entendu de pouvoir faire communiquer deux composants sans devoir faire aucune intervention dans le code.

Ces différentes approches sont donc très limitées dans l'optique d'une automatisation de la gestion de la communication entre plusieurs composants, puisque il est nécessaire d'adapter le code même des applets et des servlets afin de les faire communiquer entre-elles. Or, il ne faut pas oublier que les outils qui sont utilisés, dans le cadre de l'environnement *eMersion*, sont destinés essentiellement à des utilisateurs possédant des connaissances restreintes en programmation. Ces constatations nous obligent donc à songer à une approche radicalement différente qui ne nécessiterait pas d'intervention au coeur même du code des composants. Ceci nous a mené à étudier la prise en charge du concept des services Web qui sera présentée dans la section suivante.

4.2 Intégration de services Web

4.2.1 Introduction

Dans la suite de ce qui a été présenté à la section précédente, nous avons été tout logiquement porté à chercher une solution plus générale, permettant d'établir des liens entre composants. L'idéal étant de pouvoir créer des liens de communication entre composants de manière totalement automatisée, et sans intervention au sein du code des composants tel que nous avons dû le faire avec les applets. Pour cela, nous avons été invités à explorer les concepts établis par les services Web [30]. Nous allons donc présenter, dans cette section, ce que signifie la notion de services Web et comment ils sont gérés.

Dans un deuxième temps, nous allons présenter comment nous avons créé un outil permettant de générer automatiquement un interfaçage de communication avec un service Web.

Finalement nous préciserons comment nous avons pu intégrer ces concepts dans le cadre de l'outil auteur et de l'outil de publication.

4.2.2 Définition des services Web

> Voici une définition de la conception des services Web [Cha02, p.17-18] :
> *« L'architecture des services Web repose sur un mécanisme de transport d'une demande de service entre un client et un serveur, tous deux connectés au réseau.*
> *Dans cette architecture, le client peut être soit un navigateur Web, soit une application. Le rôle du serveur, quant à lui, est joué par une application qui s'exécute sur un moteur de scripts interfacé à un serveur HTTP ou sur un serveur J2EE ou .NET. Le point crucial est ici l'étanchéité entre les clients et les serveurs qui ignorent tout de l'implémentation, dite privée, des opérations respectives de leurs correspondants et ne se connaissent et se reconnaissent qu'au travers de descriptions publiques, appelées services Web. »*

De manière générale, nous pouvons définir un service Web (WS, Web Services) comme étant un moyen d'échanger des informations entres deux applications par l'intermédiaire du Web. De manière plus précise, il existe de nombreuses applications qui sont mises à dispositions sur le Web et qui fournissent un service particulier. Pour pouvoir profiter de ce service, il faut créer une application qui va envoyer une information à un service Web particulier. Celui-ci va effectuer un traitement particulier à cette information et renvoyer le résultat du traitement à l'application qui en a fait appel.

Par exemple, il existe des services Web auxquels il suffit de renvoyer le numéro postal d'une ville pour obtenir la météo courante du lieu correspondant. Dans un autre registre, des services Web renvoient la cotation courante d'une valeur boursière en fonction de l'indice de cotation reçu. A partir de ces exemples, il est possible de s'imaginer de nombreux autres services Web disponibles, aussi variés les uns que les autres.

Le fonctionnement d'un service Web peut être variable, et il n'est pas réellement utile de savoir comment l'information envoyée est traitée avant de recevoir une réponse. En fait, un service Web peut être conçu comme une boîte noire qui reçoit des valeurs en entrée et renvoie les solutions après traitement. Le contenu de la boîte noire peut se programmer dans n'importe quel langage. Afin de pouvoir concevoir une application cliente qui communique avec un service Web, il est nécessaire de savoir comment sont gérés les paramètres d'entrée du service Web ainsi que le type de ces valeurs.

Par ailleurs, la communication avec un service Web s'effectue au travers d'un protocole de communication particulier nommé SOAP [39]. Ce protocole est en réalité une couche supplémentaire au protocole HTTP utilisé sur le Web. Le fait d'exploiter comme base le protocole HTTP offre de multiples avantages, car ce protocole est le protocole standard de communication sur le Web. De ce fait, l'utilisation du protocole SOAP n'est autre qu'une particularisation du protocole HTTP, offrant ainsi l'avantage de naviguer facilement au travers des différents pare-feux chargés de bloquer toute communication non désirée.

La description d'un service Web particulier s'effectue au travers d'un fichier WSDL (Web Service Definition Language) [31] qui décrit toutes les caractéristiques pour établir une communication avec un service Web, le format des valeurs en entrée, en sortie, le nom des fonctions, le nom du service Web, l'adresse Web du service, les commentaires sur chaque fonction fournie, etc. Un document WSDL correspond en fait à un fichier XML auquel correspond un Schéma-XML définissant la norme WSDL. Ce fichier WSDL est donc véritablement le mode d'emploi d'un service Web.

La notion de service Web est un concept récent qui commence à se populariser assez rapidement sur le réseau Internet. Les concepts qu'il propose sont très prometteurs et correspondent d'une certaine manière à la vision du réseau Internet comme un énorme système réparti, mettant à disposition de tous, sa capacité de calcul infinie. A ce moment, il deviendrait possible d'exécuter des applications distantes depuis n'importe quelle machine et de manière simple, sans se soucier de la manière dont est programmée l'application.

4.2.3 Utilité des services Web dans le cadre des cockpits

Nous pourrions nous poser la question de l'utilité de traiter du cas des services Web dans le cadre du développement des cockpits destinés à un environnement d'expérimentation et d'apprentissage, tel que voulu par l'environnement *eMersion*. Bien entendu, offrir la prise en charge des services Web étend les possibilités d'insérer des composants multimédias et interactifs dans le cadre d'un cockpit. Mais bien plus que cela, la prise en charge des services Web permettrait d'inclure différentes applications développées de manière autonome par différents laboratoires souhaitant participer à la plate-forme d'expérimentation *eMersion*. Dès lors, il suffirait pour ces laboratoires d'effectuer de minimes adaptations à leurs applications, afin de les rendre accessibles selon la conception voulue par les services Web. Cela permettrait véritablement d'exploiter des outils déjà opérationnels, sans devoir recommencer à zéro une nouvelle implémentation destinée à être portée dans un environnement Web. Par ailleurs, en poursuivant le raisonnement, nous pourrions même imaginer faire communiquer différentes applications fournies par différents laboratoires, par l'intermédiaire de services Web.

Cependant, cette approche dépasse le cadre de ce projet et ne sera pas traitée plus largement. Nous nous contenterons d'expliquer comment nous avons traité la génération

d'interfaces de communication avec des services Web, par la génération automatisée de formulaire HTML.

4.2.4 Conception d'un interfaçage automatique d'un service Web

Afin de pouvoir exploiter la notion de service Web dans notre projet, nous avons mis en place une approche visant à générer une interface de communication automatique à partir du fichier de description d'un service Web (WSDL). Le but est donc d'analyser ce fichier, et d'en produire un page Web qui permettrait d'insérer les informations pour les renvoyer au service Web. Cependant, la page HTML n'est pas en mesure de rentrer directement en communication avec un service Web. En effet, pour qu'une communication puisse se réaliser, il est nécessaire de passer, par exemple, par l'intermédiaire d'une servlet. Celle-ci analyserait les informations récupérées du formulaire HTML, effectuerait les opérations de transtypage pour faire correspondre les types des valeurs reçues du formulaire HTML aux valeurs acceptées par le service Web. Toutes ces valeurs seraient ensuite transmises au service Web en attendant qu'il renvoie les résultats à la servlet. Lorsque la servlet recevrait ces résultats, elle les reformaterait pour les insérer dans une page HTML qu'elle renverrait au navigateur qui lui avait envoyé la requête initiale.

Concernant l'établissement de la communication entre la servlet et le service Web, il s'avère nécessaire de passer par une interface de communication. Cette interface de communication se nomme un 'Stub'. Ce 'Stub' peut être vu comme un composant spécialisé de la servlet qui a pour unique objectif d'établir les connexions avec le service Web par l'utilisation du protocole SOAP. Ce protocole correspond au langage de communication avec les services Web. Le 'Stub' va donc transformer les informations de la servlet pour les adapter aux normes du protocole SOAP, et les transmettre au service Web. De même, la réponse du service Web, toujours selon les normes du protocole SOAP, va être récupérée par ce 'Stub' qui va à nouveau adapter ces informations pour que la servlet puisse les renvoyer au client sous la forme d'une page HTML.

Nous avons opté pour la prise en charge de la communication avec les services Web par un 'Stub', pour la simple raison qu'il existe des outils très performants qui permettent de générer automatiquement les différents fichiers du 'Stub' client, à partir d'un fichier WSDL décrivant un service Web particulier. Pour cela, nous utilisons une API nommée `wsdl2Java.java`, fournie par Axis [20] et contenue dans la librairie `axis.jar`. De cette manière, nous avons pu nous concentrer sur la conception de la servlet.

La figure 4-5 représente l'architecture de communication avec un service Web, telle que nous l'avons implémentée dans le cadre de son intégration dans un nouveau cockpit.

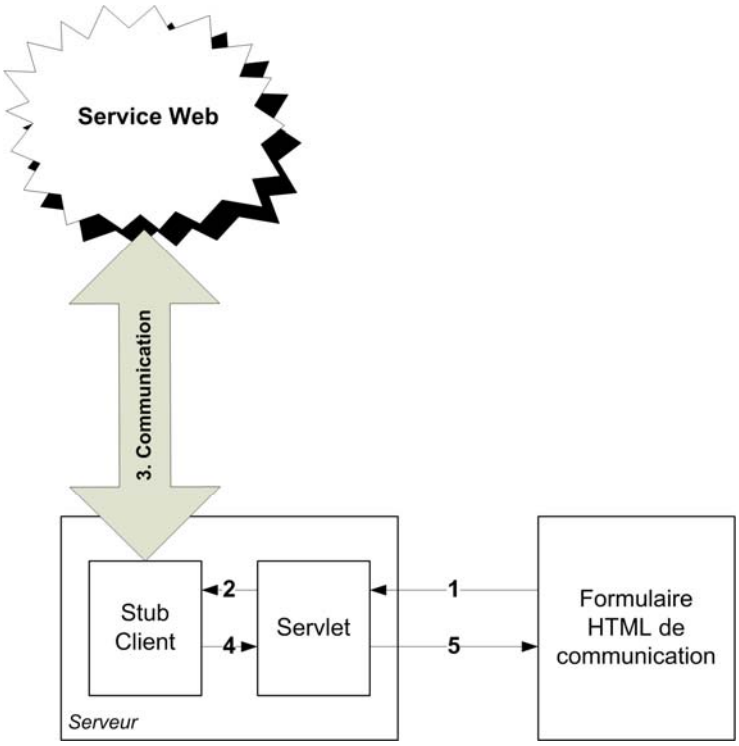


Figure 4-5 Architecture de communication avec un service Web

Afin de satisfaire cette nouvelle approche, il s'est avéré nécessaire de concevoir de nouvelles transformations XSLT. Par contre, cette fois, trois transformations différentes sont nécessaires. La première consiste à produire la page HTML avec le formulaire d'interrogation du service Web. Ensuite, il est nécessaire de produire le code Java de la servlet servant de lien direct avec le service Web. Par ailleurs, lorsqu'on parle de servlet, il est nécessaire également de produire un fichier nommé 'Web.xml', utilisé pour référencer la servlet dans le serveur Tomcat.

La page HTML générée va produire un formulaire particulier pour chaque fonction fournie par le service Web. Cependant, certains services Web fournissent une multitude de fonctions différentes, risquant ainsi de rendre le fichier HTML difficile à comprendre. Afin d'éviter ce problème, si un service Web fournit plus d'une fonction, la page HTML affichera la liste des fonctions disponibles en permettant à l'utilisateur de sélectionner la fonction souhaitée. Selon la fonction choisie, le formulaire d'accès au service Web va changer pour correspondre à la fonction sélectionnée. Le code Java généré produira également une méthode d'accès au service Web pour chaque fonction du service Web. Ainsi, la méthode invoquée dépendra du formulaire utilisé par la page HTML.

Il est utile de préciser que la prise en charge du fichier de description WSDL par cet outil, prend en compte les types `string`, `integer`, `short`, `float`, `double`, `long`, `byte` et `boolean`. Concernant les types complexes, la norme WSDL permet de définir d'autres types, en suivant la structure de type définie par la norme XML-Schema. Ceci offre une très large possibilité de types complexes qu'il est alors possible de définir. Pour cela, notre outil prend en charge les services Web ayant comme types complexes des structures contenant les types simples, pris en charge et énoncés ci-dessus. De plus les tableaux sont également pris en compte pour les réponses contenant des tableaux avec des types simples. Ceci limite donc notre outil aux services Web utilisant les types

supportés par notre outil. Toutefois, cela satisfait déjà à générer des interfaces de communication pour une multitude de services Web.

Finalement, la servlet et le 'Stub' que nous avons générés ne sont pas encore compilés, puisque nous avons généré du code Java pour la servlet et pour le 'Stub'. Il est donc nécessaire de penser à une solution pour compiler le code Java de manière dynamique, sans intervention de l'utilisateur. Cette solution sera présentée ci-dessous, lorsque nous traiterons de l'intégration de l'interfaçage avec un service Web dans l'outil de publication.

La figure 4-6 nous permet de visualiser la manière dont sont produits les documents nécessaires à l'établissement d'une communication avec un service Web. Tous ces nouveaux fichiers sont issus du fichier WSDL de description du service Web. La page HTML est gérée par le fichier de transformation XSLT nommé '`WS-Html-Generator.xsl`'. La génération de la servlet est gérée par le fichier '`WS-Servlet-Generator.xsl`', alors que le fichier `Web.xml` est généré par le fichier '`WS-Web_Xml-Generator.xsl`'.

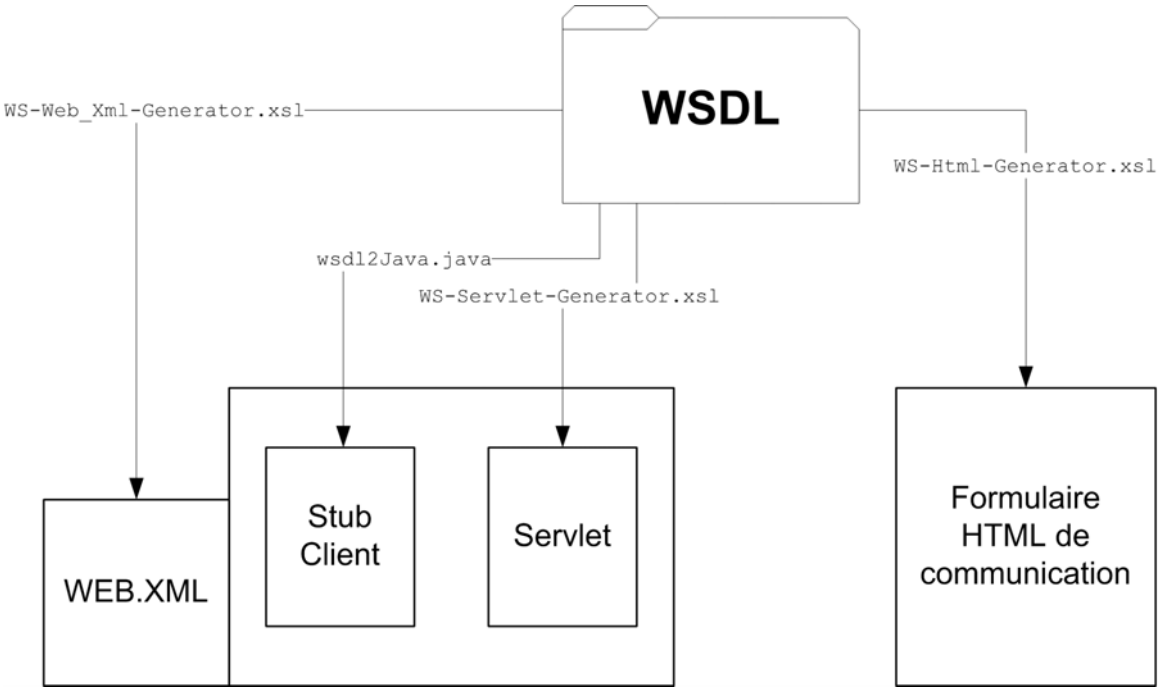


Figure 4-6 Documents générés à partir du fichier WSDL

4.2.5 Intégration des services Web dans les outils auteur et de déploiement

Le concept de services Web étant nouveau dans le cadre de l'outil auteur, et de l'outil de déploiement des cockpits sur le serveur, il s'est avéré nécessaire de mettre à jour ces deux outils, afin que les services Web soient totalement intégrés et pris en charge. Pour cela, nous avons dû modifier légèrement l'outil auteur, afin que le concepteur d'un cockpit puisse insérer un service Web dans un conteneur. De la même façon qu'un utilisateur insère un document structuré, une vidéo ou encore une applet, nous avons rajouté la possibilité d'intégrer un service Web à l'intérieur d'un cockpit. Comme pour l'insertion d'un document XML à l'intérieur d'un conteneur, un service

Web sera inséré par le biais de son fichier de description WSDL. Ainsi, lorsque l'utilisateur voudra insérer un service Web dans son cockpit, il devra définir le fichier WSDL correspondant. De cette façon, lors de la génération du cockpit, nous allons considérer que nous traitons une simple page HTML, cette page étant générée automatiquement à partir du fichier WSDL lors du déploiement. En offrant la possibilité d'insérer un nouveau type de document, il a également fallu adapter le Schéma-XML décrivant la structure autorisée pour les cockpits. Pour cela, nous avons rajouté un élément supplémentaire, ayant pour nom 'WebService', dans les types pouvant être insérés dans un conteneur du cockpit.

Concernant l'outil de publication du cockpit, il a fallu prendre en compte ce nouvel élément définit dans le fichier XML de description du cockpit. De la même façon que nous avons procédé lors de la génération d'une page HTML à partir d'un fichier XML contenu dans un document structuré du cockpit, il a fallu parcourir le fichier XML de description du cockpit à la recherche d'élément nommé 'WebService'. Si de tels éléments sont trouvés, il faut alors effectuer les trois transformations XSLT présentées ci-dessus, ainsi que la génération du 'Stub'. Les services Web sont donc gérés, de prime abord, de la même façon que les documents structurés de type XML. Nous avons d'ailleurs conservés les mêmes classes utilisées pour la génération de documents structurés de type XML, en les adaptant pour gérer les services Web. Par contre, cette fois nous effectuons trois transformations XSLT avec les feuilles de styles présentées au paragraphe précédent. Nous générons donc un document HTML contenant les formulaires d'accès au service Web, le code Java de la servlet de connexion, les différents fichiers formant le 'Stub', ainsi qu'un fichier '`Web.xml`' pour toutes les servlets générées. De cette manière, si nous générons un cockpit contenant plusieurs services Web différents, un seul et unique fichier '`Web.xml`' sera généré pour permettre au serveur Tomcat de référencer toutes les servlets. Finalement, il faut effectuer la compilation dynamique des fichiers Java qui viennent d'être générés pour la servlet et le 'Stub'. Afin que la compilation puisse s'exécuter correctement, il est nécessaire d'ajuster dynamiquement le 'classpath' utilisé par le compilateur Java, pour que celui-ci puisse retrouver les librairies référencées par les fichiers de la servlet et du 'Stub'.

Pour terminer ce point, précisons encore que les fichiers générés pour le cockpit doivent respecter une structure de répertoire particulière, afin de permettre au serveur Tomcat de référencer dynamiquement le cockpit. Cette structure a déjà été présentée lorsque nous avons présenté l'outil de déploiement des cockpits. Comme nous l'avons déjà précisé dans ce chapitre, le répertoire de base du cockpit se trouvera dans le répertoire '/webapps' du répertoire de base de l'application Tomcat. Dans ce répertoire, nous retrouverons tous les fichiers de bases du cockpit, mais également un répertoire nommé '/WEB-INF', contenant le fichier '`Web.xml`' que nous venons de générer. Dans ce même répertoire, se trouvera un répertoire nommé '/classes', avec tous les fichiers générés lors de la compilation des fichiers sources de la servlet.

Dès à présent, lorsque nous exécutons un cockpit contenant un service Web, nous pouvons constater que le comportement correspond à notre attente, que le serveur Tomcat prend en charge correctement les servlets générées dynamiquement dans le cadre des services Web. Le comportement des services Web contenus dans des conteneurs est parfaitement cohérent, et la communication par l'intermédiaire de la servlet nous renvoie les résultats conformes à la demande effectuée au service Web.

4.2.6 Conclusion

Nous avons présenté, dans cette section, la manière dont nous avons géré les services Web dans ce projet. Nous avons commencé par présenter une définition, le fonctionnement général ainsi que l'utilité d'un service Web.

Dans un deuxième temps, nous avons décrit la manière dont nous avons pris en charge la génération automatique d'un interfaçage de communication avec un service Web, à partir de sa description dans un ficher WSDL. Pour cela, nous avons créé trois nouvelles feuilles de style destinées à générer un document HTML, le code de la servlet de communication ainsi qu'un fichier '`Web.xml`' nécessaire au serveur Tomcat pour référencer les servlets.

Le document HTML, produit automatiquement, contient la liste des fonctions fournies par le service Web. L'utilisateur peut alors choisir la fonction qu'il souhaite appliquer au service Web. La servlet agit comme intermédiaire et renvoie la réponse du service Web.

Finalement, nous avons présenté la manière dont nous avons pu intégrer ce concept de services Web dans l'outil auteur et adapter l'outil de publication pour qu'il prenne correctement en charge les éléments de type 'WebService', contenus dans le fichier XML de description du cockpit. Nous avons enfin rappelé la structure des répertoires nécessaires à la bonne prise en charge par le serveur Tomcat du nouveau cockpit contenant des services Web.

Conclusion générale

La première étape de ce projet a consisté à reprendre l'outil auteur implémenté lors d'un projet précédant, afin de l'ajuster aux contraintes de l'outil de publication. En effet, puisque l'outil auteur a été développé de manière indépendante, il s'est avéré nécessaire d'y apporter des modification, notamment en y ajoutant une fonctionnalité supplémentaire permettant d'invoquer l'outil de publication directement depuis cet outil auteur, tout en préparant les fichiers destinés à être envoyé sur le serveur.

Dans un deuxième temps, nous avons présenté les étapes successives qui ont abouties à la conception de l'outil de publication. Pour cela, nous avons présenté en détail la façon dont nous avons dû déboguer une ébauche de génération d'environnement Web, et surtout les modifications et les compléments que nous y avons effectués. Cela nous a mené à gérer différents composants multimédias qui pouvaient être introduits lors de la conception d'un nouvel environnement. Suite à cela, nous avons traité de la façon dont était généré un environnement correspondant à celui créé par le biais de l'outil auteur. Lorsque tous les points précédents ont été achevés, il a fallu rassembler l'ensemble de ces opérations, pour qu'elles soient effectuées de manière automatisée lors de la phase de publication sur le serveur.

Une fois que nous nous sommes retrouvés avec un outil de publication opérationnel, nous nous sommes intéressés sur la façon dont nous pouvions faire communiquer différents composants entre eux. Pour cela, nous avons traité différents cas d'applet Java s'échangeant des informations.

Finalement, nous avons introduit la notion de services Web et la façon dont nous les avons gérés dans le cadre de l'outil auteur, et leur prise en charge par l'outil de publication. Nous avons conçu un outil intégré à l'outil de publication qui permet de générer automatiquement une interface de communication avec divers services Web. Cette approche prometteuse nous a permis de suggérer une nouvelle vision concernant la communication avec des applications distantes et qui mériterait d'être approfondie dans le cadre de projets futurs.

EPFL, le 7 janvier 2005
Olivier Granges

Glossaire

Définitions de termes techniques employés dans le cadre de ce rapport : [16]

ActiveX	Technologies due à Microsoft permettant à différents logiciels de communiquer entre eux quelque soit la plate-forme sur lesquels ils tournent.
API	*Application and Programming Interface*. Ensemble de bibliothèques permettant une programmation plus aisée car les fonctions deviennent indépendantes du matériel.
applet	Application Java destinée à être exécutée dans un navigateur Internet.
Classpath	Variable d'environnement définissant les chemins d'accès aux classes Java.
Cockpit	Environnement d'expérimentation Web créé par un outil auteur.
Conteneur	Fenêtre d'un cockpit contenant un ou plusieurs composants multimédias.
CSS	*Cascading Style Sheets*. Norme du W3C définissant des feuilles de style utilisées dans un document HTML. Elles permettent de gérer des règles comportementales, typographiques, graphiques d'une page HTML dans un navigateur Internet.
DTD	*Document Type Definition*. Structures types de documents XML qui décrivent les éléments composant les textes et les règles qui les régissent.
Flash	Logiciel conçu par Macromedia permettant de construire des animations vectorielles complexes.
GIF	*Graphics Interchange Format*. Format de fichiers d'image le plus répandu sur le Web et inventé par Compuserve. Les fichiers au format GIF sont fortement compressés (méthode LWZ) tout en gardant une qualité très correcte. Le nombre maximal de couleurs contenu dans une image au format GIF est de 256.
HTTP	*HyperText Transfer Protocol*. Protocole de communication utilisant le principe de requête/réponse, utilisé sur internet pour établire les communications entre les navigateurs Internet et les serveurs Web.

JAR	*Java ARchive*. Fichier d'archive Java.
Java	Langage de programmation produit par la société Sun. Il s'agit d'un langage multi plateformes permettant de créer des applications autonomes. Il peut également être utilisé pour gérer des environnements Web.
JavaScript	Langage de script utilisé dans des pages HTML afin de les rendre dynamiques.
JPEG	*Join Photographic Experts Group*. Standard pour la compression des images qui contiennent plusieurs millions de couleurs.
JSP	*Java Server Page*. Technologie Java de création de pages dynamique en HTML.
Machine virtuelle Java	Environnement permettant d'exécuter le byte code généré lors de la compilation d'une application Java. Le byte code étant portable, la machine virtuelle le transforme en code machine dans l'objectif d'être exécuté.
MIME	*Multipurpose Internet Mail Extensions*. Format conçu pour transférer par e-mail des données codées qui ne sont pas du texte (images, programmes, etc.).
Outil auteur	Application permettant de créer un environnement d'expérimentation Web.
Outil de publication (renderer)	Application serveur servant à publier un environnement d'expérimentation Web (cockpit) créé avec un outil auteur.
Plug-in	Petit module qui s'installe sur un navigateur pour lui apporter des fonctionnalités supplémentaires.
PNG	*Portable Network Graphic*. Format d'image libre de droits offrant une qualité supérieure au format GIF pour une meilleure compression.
Pop-up	Fenêtre d'un navigateur Internet s'ouvrant automatiquement lors de la visite d'un site Internet.
Service Web	Fonctions particulières offertes par une application distante et quelconque s'exécutant sur un serveur distant. Comparable à une '*boîte noire*' transformant une information reçue en entrée, puis renvoyant le résultat à l'expéditeur.
Servlet	Programme java s'exécutant sur un serveur Web pour communiquer avec un navigateur Internet en utilisant le protocole HTTP.

SMIL	*Synchronized Multimedia Integration Language*. Langage de création de présentation multimédia basé sur XML et normalisé par le W3C.
SVG	*Scalable Vector Graphics*. Langage permettant la description des graphiques bidimensionnels en XML, concurrent de Flash.
Tomcat	Serveur Web prenant également en charge des composant Web, JSP et servlet.
VBScript	*Visual Basic Scripting*. Script étant une extension du langage Visual Basic, développé par Microsoft. Peut être utilisé comme script dans Internet Explorer.
W3C	*World Wide Web Consortium.* Fondé en 1994 par Tim Berners-Lee, son principal objectif est la mise au point de normes et de protocoles ouverts et libres, dans un souci d'interopérabilité maximale.
WSDL	*Web Service Definition Language*, document XML décrivant le fonctionnement d'un service Web (WS) particulier.
XML	*eXtensible Markup Language*. Document structuré utilisant le concept de balises pour décrire un ensemble d'informations spécifiques.
XML-Schema	Document XML servant de 'grammaire' à une structuration XML particulière; un XML-Schema définit le cadre autorisé que doit respecter un document XML affilié.
XSL, XSLT	*eXtensible Stylesheet Language*. Feuille de style permettant d'effectuer des transformations particulières sur un document de type XML vers un nouveau document de type HTML, XML ou texte quelconque.
ZIP	Format de fichiers compressés.

Annexes

Annexe A
Installation de l'outil auteur

A-I Introduction

Nous présentons ci-dessous comment installer l'outil auteur de manière assez succincte. Pour de plus amples détails sur les procédures d'installation, prière de se référer au rapport traitant de la conception de l'outil auteur [Pas04] et qui traite en détails les procédures d'installation de cet outil.

A-II Procédures d'installation

Dans les fichiers que nous rendons avec ce rapport, nous fournissons deux archives ZIP contenant l'outil auteur que nous avons modifié.

1. La première archive nommée '`cockpitComposerPlugin.zip`' contient uniquement le plugin de l'outil auteur. Pour installer ce plugin, il est nécessaire d'avoir déjà installé préalablement eClipse [10] sur sa machine. Si tel est le cas, il suffit de décompresser le contenu de cette archive dans le répertoire de base d'eClipse. Les fichiers du plugin seront ainsi copiés dans le sous répertoire nommé 'plugins'.
2. La deuxième archive contient une version d'eClipse Plateforme déjà configurée avec le plugin cockpitComposer et prêt à l'emploi. Pour cela, il suffit de décompresser cette archive sur son disque et de démarrer l'outil auteur par le fichier '`cockpitcomposer.bat`'.

Remarque :

Lors du démarrage de l'outil auteur, il est possible que la barre d'outil et le menu se référant au cockpit ne soient pas affichés dans la perspective 'cockpit' comme cela est représenté dans la figure ci-dessous.

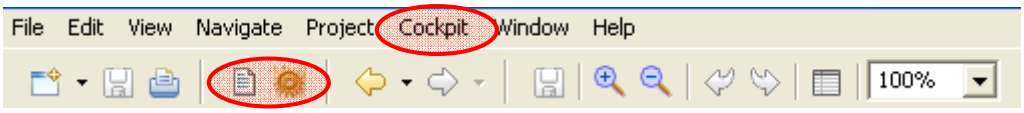


Figure A-0-1 Menu de l'outil de publication

Si le menu 'Cockpit' et les deux boutons ne sont pas affichés, il faut suivre la procédure décrite ci-dessous. Lorsque l'outil auteur sera relancé les fois suivantes, la barre d'outil et le menu resteront correctement configurés.

1. Faire un clique droit avec la souris en positionnant le curseur à droite de la barre d'outils. La fenêtre suivante apparaît :

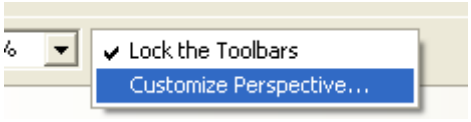


Figure A-0-2 Clique droit de la souris à droite de la barre d'outils

2. Cliquer sur le champ '*Customize Perspective…*'. La fenêtre suivante apparaît :

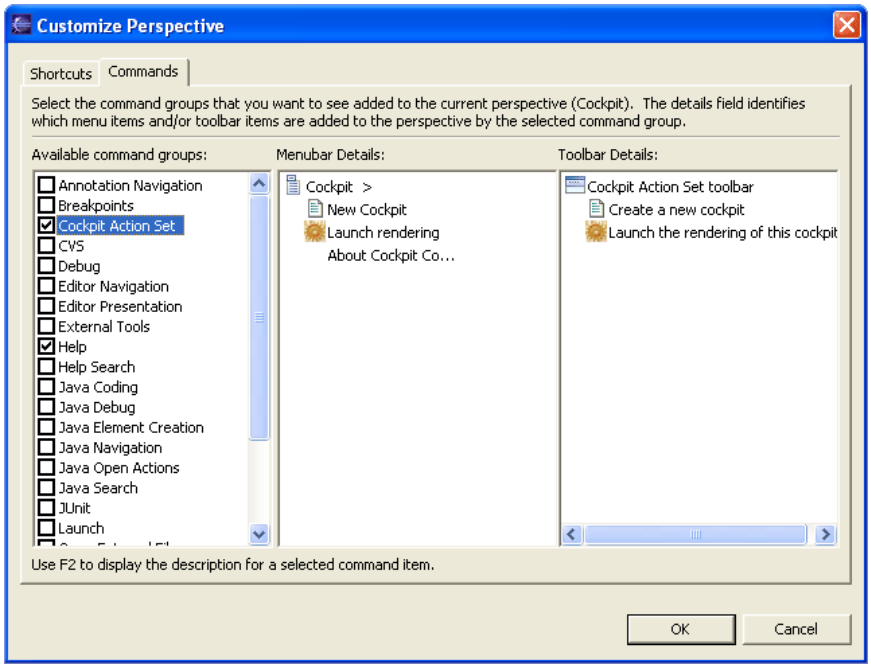


Figure A-0-3 Fenêtre de personnalisation des barres de menus et d'outils

3. Cliquer sur l'onglet '*Commands*' et cocher la case '*Cockpit Action Set*'. Finalement, en cliquant sur le bouton '*OK*', les barres de menus et de titres permettront d'accéder aux opérations de traitement sur les cockpits.

A-III Manuel d'utilisation de l'outil auteur

Concernant le manuel utilisateur de l'outil auteur, prière de se référer au manuel utilisateur annexé au rapport de l'outil auteur.

Annexe B
Installation de l'outil de publication

B-I Installation du serveur Tomcat

L'outil de publication fonctionne sur un serveur Tomcat. Pour cela, il est donc nécessaire que Java soit installé. Ensuite, il faut installer le serveur Tomcat avant d'installer l'outil de publication. Il faut donc télécharger une version de Tomcat disponible à l'adresse http://jakarta.apache.org/tomcat/ ou installer la version fournie avec les fichiers de ce projet. Nous avons utilisé la version 5.0.28.

Lors de l'installation, il est possible de configurer le port par défaut pour accéder au serveur en mode local. Nous avons décidé de conserver la valeur par défaut qui utilise le port 8080. Ainsi, une fois installé, l'accès en local à la page d'accueil du serveur Tomcat se fait par l'adresse 'http://localhost:8080/'.

Sous Windows, le serveur Tomcat est installé dans le répertoire 'C:\Program Files\Apache Software Foundation\Tomcat 5.0'. Ce répertoire de base est nommé '`CATALINA_HOME`'.

Remarques :

- Afin d'assurer le bon fonctionnement du serveur Tomcat, il est nécessaire de spécifier une nouvelle variable d'environnement nommée `JAVA_HOME`. Cette variable d'environnement doit pointer sur le répertoire de base où a été installé Java.
  Par exemple, sous Windows XP, pour rajouter une nouvelle variable d'environnement, il faut cliquer avec le bouton droit de la souris sur l'icône '*Poste de travail*' dans le menu '*démarrer*'. Dans le menu qui apparaît, il faut sélectionner '*Propriétés*'. La fenêtre '*Propriétés système*' apparaît. Il faut sélectionner l'onglet '*Avancé*', puis cliquer sur le bouton '*Variables d'environnement*'. Dans la fenêtre qui apparaît, il faut cliquer sur le bouton '*Nouveau*' du champ '*Variables système*'. Dans le champ '*Nom de la variable*', entrer `JAVA_HOME`, et dans le champ '*Valeur de la variable*' entrer le chemin vers le répertoire de base de Java qui serait par exemple `C:\Java\j2sdk1.4.2_05`.

- La configuration des différents paramètres d'exécution du serveur Tomcat est définie dans le fichier `%CATALINA_HOME%\conf\server.xml`. Dans ce fichier, un paramètre permet de gérer la mise en place dynamique de nouvelles applications Web, évitant ainsi le redémarrage du serveur. Pour cela, il est nécessaire que l'attribut '`autoDeploy`' de l'élément `<Host>` ait la valeur '`true`'. Ceci permettra le déploiement dynamique des nouveaux cockpits. Normalement la valeur de cet attribut est déjà définie à '`true`' lors de l'installation du serveur.

B-II Installation de l'outil de publication

Les fichiers de l'outil de publication sont contenus dans l'archive ZIP nommée cockpitDeploy.zip. Il faut décompresser cette archive dans le répertoire de base du serveur Tomcat '`CATALINA_HOME`'. Cette archive contient l'ensemble des librairies nécessaires à la publication de nouveaux cockpits. Ces fichiers seront copiés dans le répertoire `%CATALINA_HOME%\common\lib\`.

Cette archive contient finalement tous les fichiers nécessaires à l'exécution de l'outil de publication et aux nouveaux cockpits. Ces fichiers seront copiés dans les répertoires `%CATALINA_HOME%\webapps\cockpitDeploy\` et `%CATALINA_HOME%\common\_linkedFiles\`.

Les nouveaux cockpits seront tous installés dans le répertoire `%CATALINA_HOME%\webapps\`.

Pour démarrer l'outil de publication, il faut démarrer un navigateur Web à l'adresse http://localhost:8080/cockpitDeploy, puis suivre les instructions présentées dans le chapitre suivant qui explique comment utiliser l'outil de publication.

Annexe C
Manuel utilisateur de l'outil de publication

C-I Démarrage du serveur Tomcat

Etant donné que l'outil de publication fonctionne sur un serveur Tomcat, pour exécuter l'outil de publication en mode local, il est nécessaire de s'assurer que le serveur a été démarré avant de publier un nouvel environnement. Pour cela, il est possible de démarrer le serveur à partir du '*Monitor Tomcat*' sous Windows à partir du menu de démarrage.

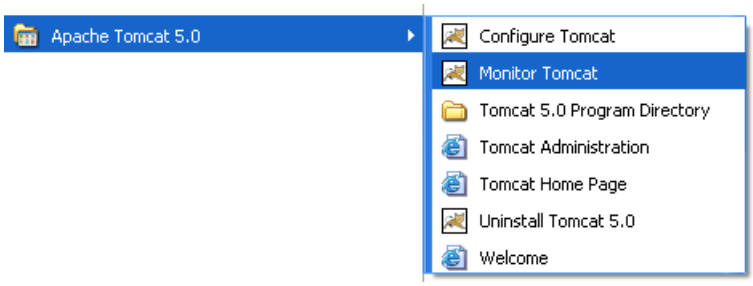


Figure C-0-4 Lancement de l'application 'Monitor Tomcat' depuis le menu de démarrage

Lorsque l'application '*Monitor Tomcat*' est lancée, une icône apparaît dans la barre des tâches. En cliquant avec le bouton droit de la souris sur cette icône, un menu apparaît, dans lequel il faut cliquer sur '*Start service*'. Cette action aura pour effet de démarrer le serveur Tomcat.

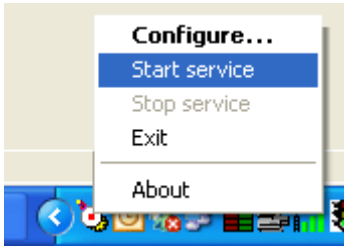


Figure C-0-5 Démarrage du serveur Tomcat depuis l'icône de la barre des tâches

Sinon, il est aussi possible de démarrer manuellement le serveur directement à partir d'un terminal. En se rendant dans le répertoire `%CATALINA_HOME%\common\`, il faut exécuter la commande '`startup`' pour démarrer le serveur. Pour stopper le serveur, il faut exécuter la commande '`shutdown`'.

C-II Lancement depuis l'outil auteur

Lorsque le concepteur d'un nouveau cockpit a achevé sa création et souhaite le publier à partir de l'outil auteur, il doit cliquer sur le bouton de publication dans la barre d'outil comme présenté dans la figure ci-dessous.

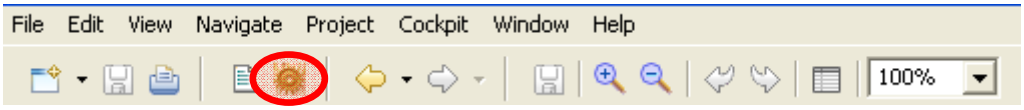


Figure C-0-6 Lancement de l'outil de publication à partir de l'outil auteur

Après avoir cliqué sur ce bouton, une nouvelle fenêtre apparaît avec la liste des cockpits qu'il est possible de publier. Il faut donc sélectionner le cockpit que l'on souhaite publier. Par ailleurs, l'adresse du serveur s'inscrit au bas de la fenêtre. Si le serveur ne se trouve pas à cette adresse qui est inscrite, il faut la modifier.

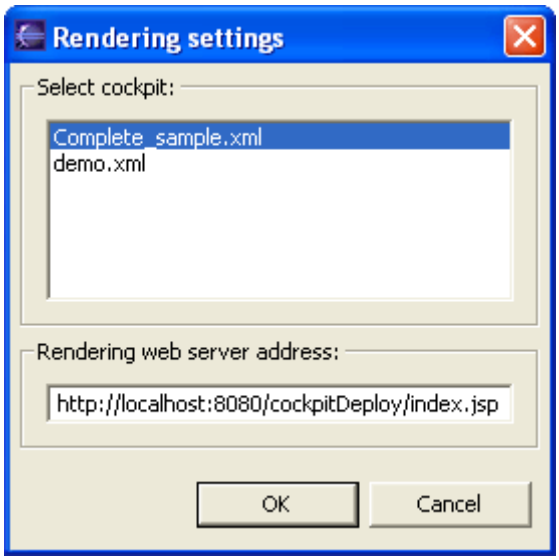


Figure C-0-7 Fenêtre de sélection du cockpit à publier avec l'adresse de l'outil de publication

C-III Utilisation de l'outil de publication

Après avoir cliqué sur le bouton 'OK' de la fenêtre de sélection du cockpit à publier depuis l'outil auteur, le navigateur par défaut est démarré à l'adresse spécifiée.

Le lien de l'archive du cockpit à publier est affiché sur la page. L'utilisateur est ainsi invité à sélectionner ce fichier su son disque local. Nous pouvons remarquer que si l'outil de publication est démarré manuellement, sans passer par l'outil auteur, aucun lien ne sera affiché.

Seule une archive ZIP est acceptée. Dans le cas contraire un message d'alerte est affiché pour inviter l'utilisateur à reconsidérer son choix.

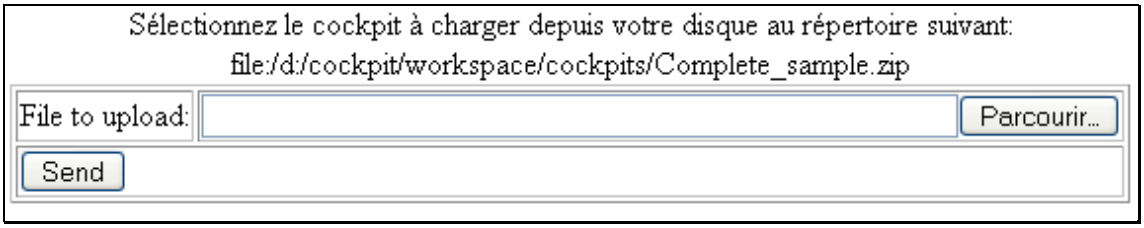


Figure C-0-8 Sélection du cockpit dans l'outil de publication

Une fois que le cockpit à publier a été sélectionné, le chargement de l'archive sur le serveur peut démarrer. Durant le chargement, une fenêtre de progression du

chargement est affichée à l'écran pour indiquer à l'utilisateur le temps restant jusqu'à la fin du chargement de l'archive du cockpit.

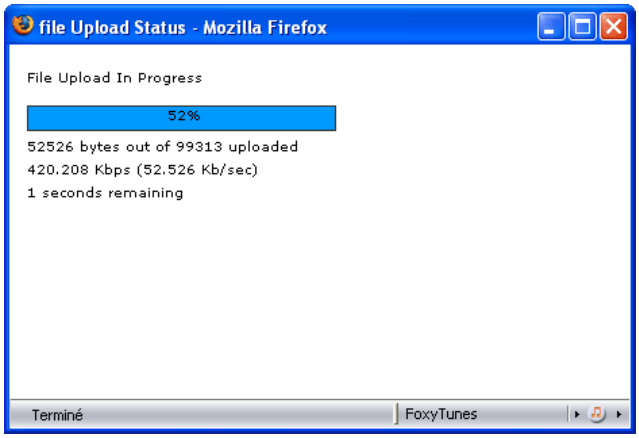


Figure C-0-9 Fenêtre indiquant la progression du chargement de l'archive du cockpit

Une fois le chargement du cockpit effectué, il est demandé à l'utilisateur de modifier ou de confirmer le nom qui sera attribué à ce cockpit. Ce nom servira également pour la page principale d'accueil du nouveau cockpit.

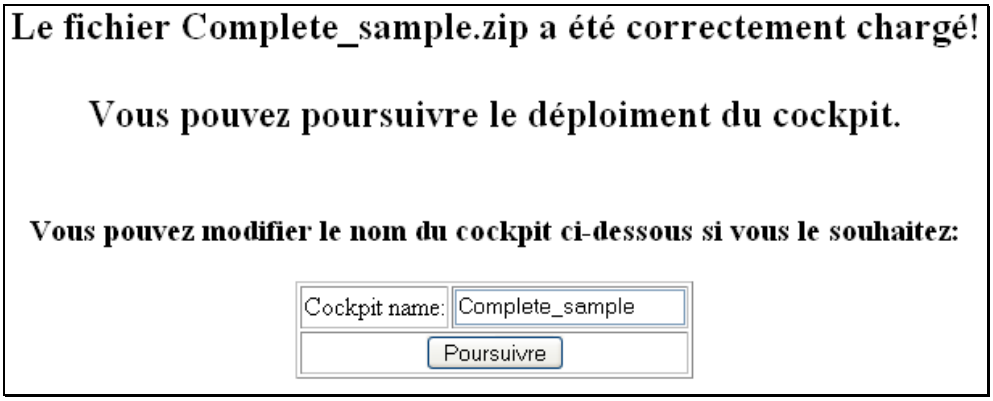


Figure C-0-10 Confirmation du nom du cockpit avant son déploiement

L'étape suivante consiste au déploiement de tous les fichiers constituant le nouveau cockpit. Lorsque ce déploiement est achevé, un rapport de déploiement est affiché. Celui-ci résume les différentes opérations qui ont été effectuées, ainsi que les éventuelles erreurs qui se sont produites. Finalement, un lien est affiché pour permettre à l'utilisateur de se rendre directement sur la page du cockpit qui vient d'être généré.

Le fichier a été décompressé avec succès!

Nom du fichier xml du cockpit: **Complete_sample.xml**

Il n'y a aucun document structuré de type XML dans le cockpit.

Les documents html internes et structurés ont été générés avec succès.

Tous les document ont été générés avec succès.

Le cockpit est prêt à être utilisé! Lien vers les fichiers du cockpit

Figure C-0-11 Rapport de publication du cockpit

Finalement, lorsque l'utilisateur clique sur le lien du nouveau cockpit, celui-ci est affiché dans la fenêtre du navigateur.

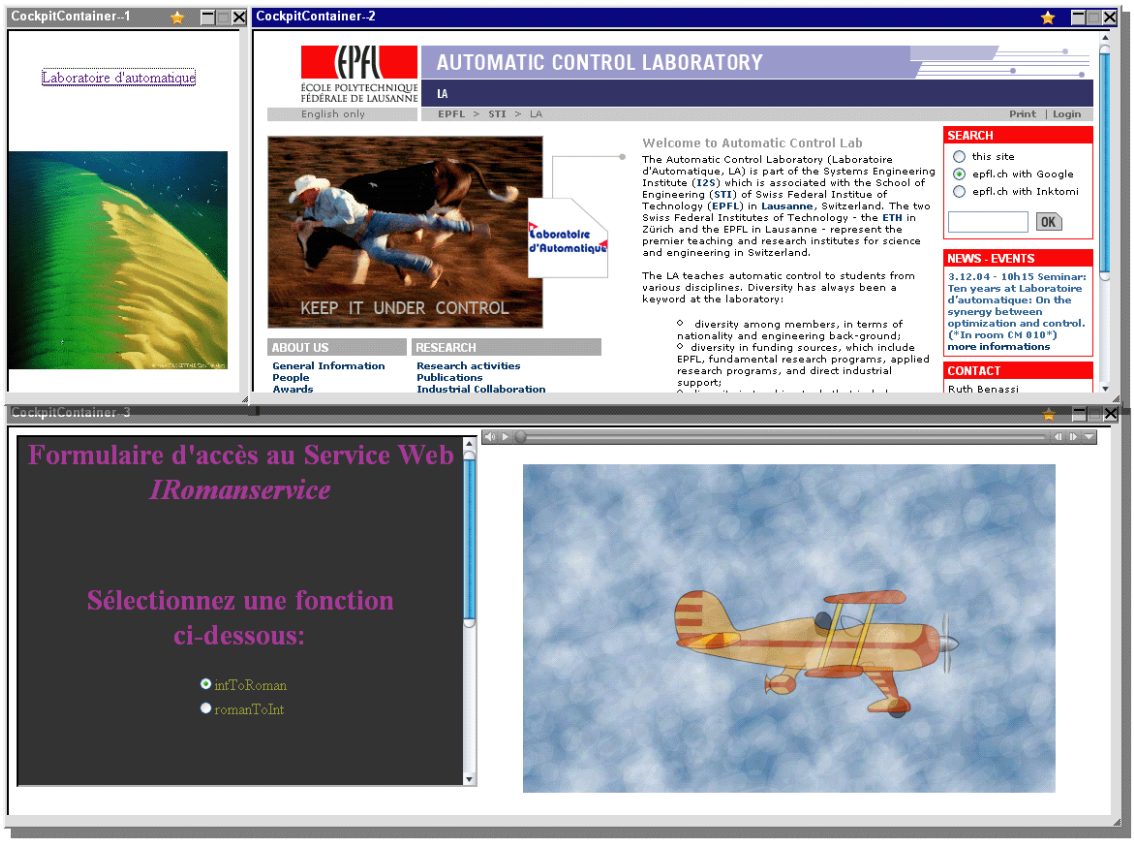


Figure C-0-12 Affichage du nouveau cockpit dans un navigateur

Annexe D
Manuel utilisateur des outils complémentaires

D-I Communication entre composants

L'exemple représentant la communication entre deux composants par le biais d'applets Java est contenu dans l'archive ZIP 'appletsCommunication.zip' fournies dans les fichiers rendus avec ce rapport. Cet exemple utilisant des servlets nécessite donc également le serveur Tomcat. Il faudra donc décompresser cette archive de la même manière que l'outil de publication, dans le répertoire `%CATALINA_HOME%\webapps\`.

Une fois cette opération effectuée, il faut s'assurer que le serveur Tomcat est démarré ; sinon il faudra le faire comme cela est indiqué dans le chapitre C-I traitant du démarrage du serveur Tomcat.

Lorsque le serveur fonctionne, il suffit de rentrer l'adresse 'http://localhost:8080/appletsCommunication/' dans un navigateur Web. L'exemple est alors présenté dans la page principale liée à cette adresse.

D-II Utilisation des outils complémentaires

Finalement, le dossier '05-Cockpit-tools' contenu dans les fichiers rendus avec ce rapport contient trois dossier représentant les trois outils complémentaires qui ont été implémentés dans le cadre de ce projet.

1. **Speed Test**

   Le dossier nommé '01_speedTest' contient une page html 'speedtest.htm' qui permet d'évaluer la vitesse de connexion à un serveur. Il suffit de lancer cette page directement depuis un navigateur Web.

2. **Text to Image**

   Le dossier nommé '02_Text_To_Image' contient un programme Java qui permet de créer une image à partir d'un texte passé en paramètre. Pour l'exécuter, il faut ouvrir un terminal pour exécuter l'application Java nommée CreateImage.java en rentrant la commande `java CreateImage <filename> <fontname> <fontsize> <colour> <message>`.

   - `<filename>` correspond au nom du fichier image PNG qui va être généré.
   - `<fontname>` correspond au nom de la police utilisé pour le style du texte de l'image générée. La liste des polices se trouve dans le sous répertoire 'fonts'.
   - `<fontsize>` correspond à la taille de la police.
   - `<colour>` est la couleur de la police sous la forme ffffff.
   - `<message>` correspond au texte qui sera écrit dans l'image. Si plus d'un mot se trouvent dans le message, il faut l'écrire entre des trémas "mon texte".

3. Image resizer

Le dossier nommé '03_Image_Resizer' contient un programme Java qui permet de redimensionner une image passée en paramètre. Pour l'exécuter, il faut ouvrir un terminal pour exécuter l'application Java nommée Resize.java en rentrant la commande `java -cp ImageEncoder.jar; Resize original_file new_filename {resize_factor | new_Width new_Height}`.

Si un seul paramètre est entré `<resize_factor>`, il s'agit du facteur de redimensionnement ; les proportions sont conservées. Par contre si deux paramètres sont entrés `< new_Width new_Height >`, il s'agit de la taille de la nouvelle image, et les proportions ne sont alors plus conservées.

Bibliographie

Ouvrages consultés lors de ce travail :

[Ave01] K. AVEDAL, JSP Professionnel. Paris: Wrox Press, 2001.

[Ber04] D.BERKOVITS, Projet de semestre - Outil de publication pour la gestion de l'emplacment spatial au sein d'environnements d'expérimentation Web. Lausanne :LA-EPFL, 2004.

[Chab04] E. CHABERT, Les JSP avec Struts, Eclipse et Tomcat. Paris: Dunod, 2004.

[Chau02] J.-M. CHAUVET, Services Web avec SOAP, WSDL, UDDI, ebXML... Paris: Eyrolles, 2002.

[Dan03] J. DANIEL, Services Web: Concepts, techniques et outils. Paris: Vuibert, 2003.

[Gil03] D. GILLET, F.GEOFFROY, K. ZERAMDINI, A.V. NGUYEN, Y. REKIK, AND Y. PIGUET, The Cockpit, An effective Metaphor for Web-based Experimentation in Engineering Education. In the International Journal of Engineering Education (IJEE). Vol. 19, No. 3, p. 389-397, 2003.
[http://emersion.epfl.ch/publications/IJEE1414.pdf]

[Gio00] GILORIEN, DHTML & JavaScript. Upper Saddle River: Prentice Hall, 2000.

[Hol02] S. HOLZNER, XSLT par la pratique. Paris: Eyrolles, 2002.

[Hor01] H. HORTON, Maîtrisez Java 2. Paris: Wrox Press, 2001.

[Pas04] S. PASCHE, Projet de diplôme - Outil auteur pour la production de documents actifs composites structurés. Lausanne :LA-EPFL, 2004.

[Ste99] R. STEYER, HTML - Guide de l'utilisateur. Paris: Micro Application, 1999.

[Tha01] D. THAU, JavaScript par la pratique. Paris: Eyrolles, 2001.

Sites Internet consultés lors de ce travail :

[1] Xerces2 – Java Parser :
http://xml.apache.org/xerces2-j/index.html

[2] SAXON – The XSLT and XQuery Processor (saxon 8) :
http://saxon.sourceforge.net/

[3] Evaluation de la vitesse de connexion par JavaScript:
http://www.toutjavascript.com/source/countkos.zip

[4] Txt2PngFactory.java :
http://chimpen.com/things/archives/001139.php

[5] Acme package :
http://www.acme.com/resources/classes/Acme.tar.gz

[6] ImageInfo.java, Marco Schmidt :
http://www.geocities.com/marcoschmidt.geo/image-info.html

[7] Resize.java, Randy Belknap :
http://www.devpapers.com/article/161

[8] Popup Windows, Brian Gosselin :
http://scriptasylum.com/misc/popup/popup.html

[9] The Apache Software Foundation
http://www.apache.org/
http://www.apachefrance.com/

[10] The Eclipse Project
http://www.eclipse.org

[11] BrowserLauncher :
http://browserlauncher.sourceforge.net/

[12] BrowserControl :
http://www.javaworld.com/javaworld/javatips/jw-javatip66.html

[13] JARV :
http://iso-relax.sourceforge.net/JARV/

[14] Sun-multiSchéma :
http://wwws.sun.com/software/xml/developers/multischema/

[15] Mega Upload :
http://www.raditha.com/megaupload/jsp.php

[16] Dictionnaire de l'informatique et d'Internet :
http://www.dicofr.com/

[17] Use a Java applet to access remote Web services :
http://www-106.ibm.com/developerworks/webservices/library/x-jappws/

[18] Java Web Services Developer Pack - JWSDP 1.2 :
http://java.sun.com/webservices/downloads/webservicespack.html

[19] SOAP and WSDL with Java and Axis :
http://www.itu.dk/courses/IWSJ/E2004/slides/axis.pdf

[20] Axis (Apache <Web Services/> Project) :
http://ws.apache.org/axis/

[21] Mozilla Firefox :
http://www.mozilla-europe.org/fr/products/firefox/

[22] Java Technology :
http://java.sun.com/

[23] Le projet eMersion
http://emersion.epfl.ch/

[24] Description de l'outil auteur actuellement utilisé
http://emersion.epfl.ch/ cockpit.html#cockpit

[25] World Wide Web Consortium
http://www.w3.org/XML/

[26] Extensible Markup Language (XML)
http://www.w3.org/XML/

[27] Laboratoire d'Automatique (LA)
http://lawww.epfl.ch/

[28] Computer Science Theory Laboratory (LITH)
http://lithwww.epfl.ch/

[29] XML Schema
http://www.w3.org/XML/Schema

[30] Web Services Activity
http://www.w3.org/2002/ws/

[31] Web Service Definition Language (WSDL)
http://www.w3.org/TR/wsdl

[32] Web The Extensible Stylesheet Language Family (XSL)
http://www.w3.org/Style/XSL/

[33] Scalable Vector Graphics (SVG)
http://www.w3.org/Graphics/SVG/

[34] Macromedia Flash Player
http://www.macromedia.com/software/flashplayer/

[35] Adobe SVG Zone
http://www.adobe.com/svg/

[36] Synchronized Multimedia Integration Language (SMIL)
http://www.w3.org/AudioVideo/

[37] Apple - QuickTime
http://www.apple.com/quicktime/

[38] Document Type Definition (DTD)
http://www.w3.org/TR/REC-xml/#dt-doctype

[39] SOAP Specifications
http://www.w3.org/TR/soap/