Automatiser le passage des Données aux Objets Java...

... sur plate-forme Mobile Java J2ME

Stage de Fin d'Etudes

Spécialité Programmation

PastaGames – ExpWay

15 Juillet - 15 Novembre 2004

Eric Bréchemier
DESS Jeux Vidéo et Média Interactifs
Décembre 2004

Résumé

Au cours des quatre mois de mon stage de fin d'études, de juillet à novembre 2004, j'ai développé l'outil Binary 4 Java (B4J). Il s'agit d'un outil de productivité dans la gestion des données, destiné aux développeurs de jeux vidéo pour téléphones portables.

PastaGames est un éditeur de jeux en Java pour téléphones portables. Un jeu comporte de nombreux éléments à paramétrer. Des fichiers de données XML regroupent les valeurs de ces paramètres pour chaque version des jeux. *La norme XML, eXtensible Markup Language, définit une syntaxe à base de <balises et="attributs">qui entourent le texte</balises> pour représenter des données de manière semi-structurée.*

Or, ces fichiers XML ne peuvent être exploités directement sur le téléphone portable. Ils doivent être encodés dans un format binaire côté serveur, puis décodés sur le téléphone pour créer les différents objets du jeu. Il s'agit d'une tâche contraignante et fastidieuse. Contraignante car il faut synchroniser le code de l'encodeur et du décodeur à chaque modification du format des données, et fastidieuse car c'est une tâche répétitive, prévisible et très similaire d'un projet à l'autre. Cette tâche se présente donc comme un bon candidat pour une automatisation.

L'outil B4J répond à ce besoin en permettant aux développeurs de travailler sur les données de leurs jeux à un niveau d'abstraction plus élevé : il automatise le passage des données XML saisies sur PC vers les objets Java exécutés sur le téléphone.

Grâce au soutien de la société ExpWay, spécialiste de la binarisation de données XML et partenaire de PastaGames pour ce projet, j'ai pu concevoir, développer et mettre en place cet outil, le tout sur une durée très courte. A l'heure des bilans, il s'agit d'un outil assez simple du point de vue fonctionnel, que PastaGames a commencé à utiliser dans le développement de son prochain jeu.

Mots-Clefs

Jeux Vidéo, Données de Description des Niveaux et Objets du Jeu
Encodeur, Décodeur, Codec
Génération de code, Compilation
Java J2ME MIDP, XML, XSLT, XML Schema

Remerciements

Je remercie chaleureusement les équipes de PastaGames et d'ExpWay qui m'ont accueilli, et je remercie plus particulièrement Fabien Delpiano (PastaGames), mon maître de stage, à l'origine de ce projet, Thibault Franchini (ExpWay) avec qui j'ai eu la chance de collaborer, sans oublier Claude Seyrat et Cédric Thiénot (ExpWay) pour l'opportunité qu'ils m'ont offerte de travailler sur de nouveaux projets passionnants au sein de leur société.

Table des matières

I. CONTEXTE ET OBJECTIFS

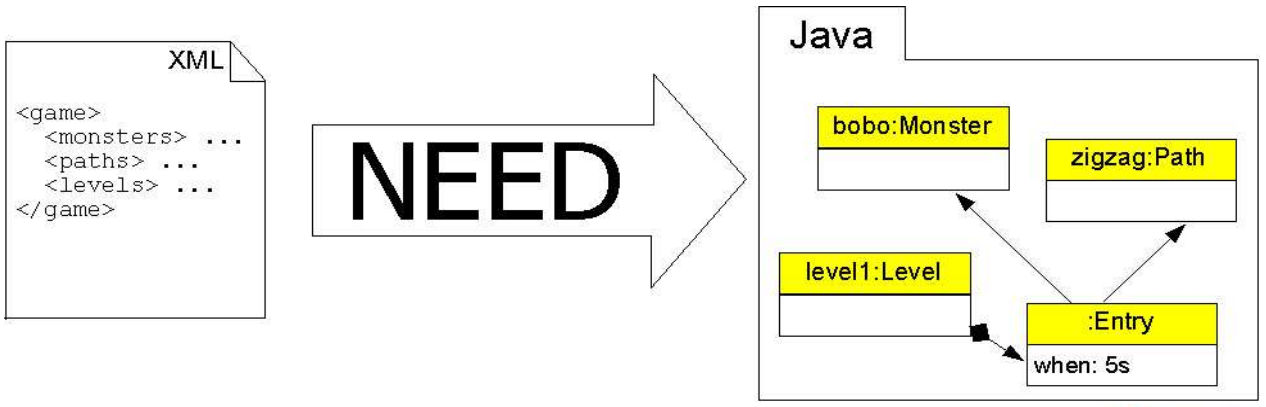
LE PROJET

Fabien Delpiano est intervenant pour la spécialité programmation du DESS Jeux Vidéo et Média Interactifs. Je l'ai rencontré lors de sa journée de cours et travaux pratiques sur le développement d'applications pour téléphones portables.

En aparté de cette conférence, il évoqua avec nous un problème pratique récurrent dans les projets de jeux vidéo pour téléphones portables : la gestion des fichiers de données qui servent à paramétrer la construction des objets Java du jeu (les monstres, le joueur, etc...).

Dans l'idéal, il utiliserait le format XML, standard de fait pour l'échange et la représentation structurée de données ; malheureusement, les fichiers XML sont trop volumineux, et les librairies de traitement de ces fichiers ne sont pas intégrées à la plate-forme J2ME (Java 2 Mobile Edition). Il est donc obligé de convertir les données XML qu'il saisit sur PC :

- dans un format binaire qui dépend de chaque jeu,
- dans un encodage spécifique à Java pour pouvoir le décoder avec des classes déjà présentes sur le téléphone (ByteArrayInputStream, DataInputStream).



Fabien Delpiano nous a alors exposé son idée : il devrait être possible de développer un outil qui se baserait sur une définition de la structure des données XML en XML Schema (définition de grammaire XML en XML), et génèrerait :

- le code source de l'encodeur pour transformer les données XML valides en données binaires
- le code source Java du décodeur pour créer les objets Java du jeu à partir des données binaires.

Fabien Delpiano a proposé aux étudiants de l'assistance un stage pour la réalisation de cet outil. Fortement intéressé par ce projet qui rejoignait mes préoccupations personnelles, j'ai alors pris contact avec lui lors de la pause déjeuner, et nous avons commencé à deviser des solutions techniques qui pourraient être mises en oeuvre.

Cette démarche de génération de code spécifique à partir d'un modèle abstrait rejoint en effet l'approche MDA, Model Driven Architecture. *Il s'agit d'une réflexion actuelle de l'OMG, l'Object Management Group. L'OMG est l'organisme qui définit la norme UML, Unified Modeling Language, la notation la plus utilisée pour modéliser des systèmes d'information*.

MDA définit un processus de développement en trois étapes :

- La définition de modèles abstraits indépendants de la plate-forme (PIM ou Plateform Independent Models). Dans notre exemple, il s'agit du schéma XML qui définit le format des données du jeu.
- La définition de modèles concrets (PSM ou Plateform Specific Models) ; spécifiques à chaque plate-forme cible, ils sont dérivés des modèles abstraits. On retrouvera bien cette étape intermédiaire, mais seulement de manière partielle, dans la mise en oeuvre qui sera présentée par la suite.
- La génération automatique du code exécutable du programme sur la plate-forme cible à partir des modèles concrets. Dans notre exemple, le but est de générer le code de l'encodeur sur plate-forme PC-Windows, dans le langage de notre choix, et le code du décodeur, en Java, sur plate-forme téléphone portable MIDP/Doja.

Pendant les semaines suivantes, nous avons réfléchi chacun de notre côté aux solutions possibles pour résoudre le problème posé et organiser le stage d'un point de vue matériel sachant que Fabien Delpiano n'avait que peu de temps pour m'encadrer et n'avait pas la place suffisante pour m'accueillir de manière satisfaisante dans les locaux de PastaGames.

Le projet ne s'est donc vraiment concrétisé que lorsque la société ExpWay, en la personne de Claude Seyrat, s'est montrée intéressée. En tant que spécialiste de la binarisation XML, ExpWay allait mettre à notre disposition son expertise et financer une partie du développement de l'outil afin de pouvoir le mettre en oeuvre par la suite sur ses propres projets. Nous allions également participer au développement d'un moteur graphique vectoriel 2D pour téléphone portable, dans le cadre d'une démonstration de portail de téléchargement de séquences vidéo.

Je remercie à nouveau la société ExpWay qui m'a accueillie dans ses locaux pendant les quatre mois de ce stage, et m'a permis de poser toutes les questions possibles à ses experts en technologies XML et binarisation XML.

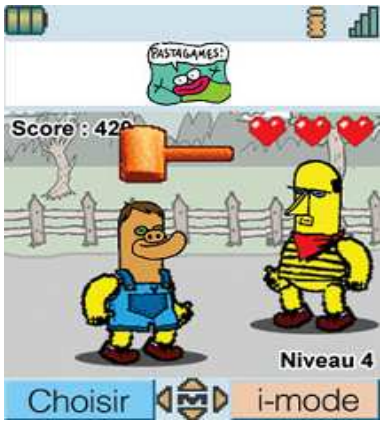
La Société PastaGames

PastaGames est une société de développement de jeux vidéo pour téléphones portables. Fondée en juillet 2000 par le dessinateur Joan Sfar, l'éditeur Guy Delcourt et deux programmeurs, elle réalise aujourd'hui un chiffre d'affaire annuel de l'ordre de 100.000€ et compte 3 employés.

Son activité se partage entre :
- le développement de jeux pour mobile,
- le portage de jeux pour mobiles développés par d'autres sociétés,
- ainsi que le développement d'autres applications pour mobile,
  dans des proportions respectives de 40%, 40% et 20% du temps de développement.

PastaGames est présente sur le portail iMode de Bouygues depuis 2003 avec deux sites : PastaGames pour les jeux vidéo et BéDé pour les bandes dessinées.

Son catalogue comporte déjà 8 jeux. Ce sont pour la plupart des jeux de plate-forme, inspirés par des classiques ou bien expérimentant des concepts de jeu plus novateurs. Il va s'enrichir prochainement d'un jeu de gestion, Kouïz, dont le but est de faire croître et se multiplier de sympathiques créatures très colorées.



La Société ExpWay

ExpWay est une start-up, fondée en 2000 par Claude Seyrat et Cédric Thiénot, deux experts des technologies XML. Leur vision est devenue la vocation de l'entreprise : rendre XML plus efficace, quels que soient les données et l'environnement logiciel ou matériel.

Rescapée de l'ère des dotcoms suite à l'explosion de la bulle Internet, elle a su s'appuyer sur la force de ses actionnaires historiques,

- **C-Source** (CDC-Entreprises, AXA Private Equities, INRIA-Transfert et Science Pratique),
- **TechFund Capital Europe** (Thomson, Thalès, EDF)
- et **Champagne Ardenne Croissance** (Région Champagne Ardennes et Caisse des Dépôts et Consignations).

Une extension de capital de 3,5 millions d'euros en octobre 2004, avec l'arrivée d'un nouvel investisseur, le **FCJE** (Fonds de co-investissement pour les jeunes entreprises : Etat Français, Fond d'investissement Européen et CDC), va financer la croissance d'ExpWay dans le domaine des guides de services.

Son activité de développement et d'édition de logiciels s'articule autour de trois grands pôles de compétences : les plates-formes de développement de binarisation XML, la télévision numérique et l'ADSL, et la téléphonie mobile.

La recherche et l'innovation sont les moteurs de la performance d'ExpWay. Elle a déposé plusieurs brevets qui mettent en valeur son expertise dans l'encodage binaire et le décodage de données XML.

ExpWay participe à de nombreux comités de normalisation : MPEG-7, TV-Anytime, DVB-(Digital Video Broadcast), et est membre du W3C. Robin Berjon (ExpWay) participe au groupe de travail SVG. Il dirige également la définition d'un nouveau groupe de travail sur la binarisation de données XML.

ExpWay fête aujourd'hui l'arrivée de son 15e employé.

EXPWAY
MAKING XML EFFICIENT™

GROS PLAN SUR... LE W3C

Le W3C, **World Wide Web Consortium**, est l'organisme à l'origine de la plupart des technologies qui définissent le Web actuel : HTML, XML, PNG, DTD et XML Schema, et aujourd'hui SVG et XForms.

Fondé en octobre 1994 par Tim Berners-Lee, inventeur et promoteur du Web, ce consortium international a pour but de regrouper et de canaliser les initiatives de chacun de ses membres vers la définition de standards communs. Initialement soutenu par le CERN, le DARPA et la Commission Européeenne, le W3C est aujourd'hui hébergé conjointement par le MIT (USA), l'INRIA (France) et la Keio University (Japon).

Ce n'est pas à proprement parler un office de standardisation, à l'image de l'ISO, International Standardization Organisation ; le W3C émet seulement des **recommandations**, ratifiées par son directeur, Tim Berners-Lee. Néanmoins ces recommandations, qui sont le fruit des travaux conjoints des plus grandes entreprises et organisations utilisatrices de ces technologies (AOL, IBM, Microsoft, Sun, Adobe, Macromedia,...) deviennent rapidement des standards de fait dans l'industrie.

Le W3C collabore sur certaines normes avec l'IETF, Internet Engineering Task Force, responsable du développement des standards de l'Internet en général.

Comment devient-on membre ?

Les membres paient une cotisation annuelle de 5750$/an. Ils s'impliquent activement dans la définition des nouvelles technologies, au sein de groupes de travail (Working Group) sur un sujet donné. Seuls les membres peuvent accéder aux documents de travail et participer aux conférences du groupe.

Les groupes de travail suivent un processus bien rôdé, et publient régulièrement des documents intermédiaires, pour montrer l'avancement de leurs travaux et recevoir les commentaires du public :

- **Working Drafts :** ébauches de recommandations, remises à jour tous les trois mois environ. Les "Last Call" Working Drafts précèdent de peu une recommandation.
- **Candidate Recommandation :** une recommandation en phase d'évaluation. Pendant cette phase, les développeurs essaient d'implémenter la recommandation et rapportent leurs difficultés. La recommandation est encore susceptible d'évoluer.
- **Proposed Recommandation :** cette "version finale" est l'objet des derniers joutes orales des membres du groupe avant d'être proposée au directeur du W3C qui a le dernier mot.
- **W3C Recommandation :** considérées stables (jusqu'à la version suivante), elles émergent d'un processus de plusieurs années, avec la garantie de pouvoir être implémentées correctement.

Les membres cèdent la propriété intellectuelle de leurs travaux au W3C, et offrent ainsi à tous la libre utilisation de ces technologies. Toutes les recommandations du W3C peuvent être implémentées gratuitement, sans besoin de licences ni de limitations par des brevets.

II. DÉROULEMENT DU STAGE

Ce stage s'est scindé en deux périodes :

- Pendant la première moitié, j'ai travaillé sur le développement d'un **guide électronique de services** pour téléphones portables en Java ; dans ce projet, PastaGames travaillait en sous-traitance pour le compte de la société ExpWay.
- Pendant les deux derniers mois, je me suis concentré sur le développement de l'**outil Binary 4 Java**, en tirant parti de l'expérience acquise dans la première partie de mon stage.

Un guide électronique de services est une appellation générale pour différentes applications qui présentent à l'utilisateur des contenus ou des services organisés de manière hiérarchique. Les programmes télé d'une ou de plusieurs chaînes, un portail de téléchargement de jeux, un magasin en ligne sont des exemples de guides de services.

Dans le cas présent, il s'agissait d'un prototype d'application de téléchargement de vidéos à la demande, développé pour la société TDF (Télé-Diffusion de France). La filiale Cognacq Jay Image de TDF a développé un service de numérisation, d'indexation et d'archivages de contenus multimédia (séquences vidéo, sons, ...). TDF souhaitait disposer d'une plate-forme de démonstration de téléchargement de fichiers vidéo à la demande, afin de valoriser ces compétences et commercialiser ce service auprès de diffuseurs de contenus sur les réseaux de téléphonie mobile.

En tant qu'employé de PastaGames détaché au sein de la société ExpWay, j'assurais un rôle de coordination entre les développeurs des deux entreprises. PastaGames se chargeait de la partie graphique en développant un moteur de présentation basé sur des graphismes vectoriels SVG. *SVG, Scalable Vector Graphics, est une norme du W3C qui définit des images vectorielles et des animations, à la manière de Macromedia Flash.* Fabien Delpiano avait également défini l'architecture générale de l'application. ExpWay s'occupait de son côté de la gestion du serveur hébergeant les vidéos, des méta-données décrivant les fichiers vidéo, de la gestion du cache des données téléchargées sur le téléphone, et de la gestion des données de session, répertoriant l'itinéraire de l'utilisateur au fur et à mesure de ses choix dans l'arborescence.

Dans le cadre de ce premier projet, j'étais plus particulièrement responsable de l'encodage binaire et du décodage des données SVG. Schématiquement, mon rôle était de transformer les fichiers SVG correspondant aux graphismes des différentes pages de l'application, d'abord en un condensé binaire, puis en objets Java Texte, Image, Rectangle, etc... gérant l'affichage des graphismes correspondants.

En suivant l'ordre chronologique, je vais commencer par présenter le travail que j'ai réalisé pour le guide de services électroniques (ESG, Electronic Service Guide), puis pour le développement de l'outil Binary 4 Java.

PROJET ESG : ELECTRONIC SERVICE GUIDE

Développement d'un Moteur de présentation 2D pour téléphones portables

Fiche produit

Il s'agit d'un prototype de démonstration d'une plate-forme de téléchargement de vidéos à la demande. Le client, TDF, qui souhaite commercialiser une solution technique de gestion et de diffusion de programmes audio-visuels, souhaite étudier la faisabilité technique et la validité d'une approche de vente de vidéos à l'unité. Ces séquences vidéo de quelques minutes seraient commercialisées autour de 1 euro.

ExpWay a réalisé cette démonstration en parallèle sur deux canaux :

- la télévision numérique sur ADSL
- la téléphonie mobile ; le téléphone cible de cette démonstration était le Nokia 3660.

Le projet ESG-Mobile s'est étalé sur une durée de cinq mois, avec trois développeurs à temps partiel. Il s'agit, relativement bien sûr au téléphone portable, d'une très grosse application : avec 43 classes qui représentent 120Ko de code compressé, et plus de 700Ko de données de tests compressées (vidéos, méta-données, images et graphismes des pages). Une MIDlet (application Java MIDP pour téléphone portable) classique comporte uniquement 5 ou 6 classes, et se limite souvent à une taille de 30Ko au total pour le code et les données.

LES PRINCIPAUX COMPOSANTS DE L'APPLICATION ESG

Le système complet se décompose en un serveur de données, où sont stockées les vidéos, et d'un client en Java qui s'exécute sur le téléphone portable Nokia 3660.

La communication aérienne entre un téléphone portable et un serveur utilise le standard HTTP classique. Pour la démonstration, le serveur était un simple serveur HTTP Apache, s'exécutant sur une machine Linux dédiée.

Sur le téléphone portable, l'application ESG en Java permet à l'utilisateur de parcourir les vidéos disponibles, classées par genre et par sous-genre. Au fur et à mesure de sa navigation, le téléphone stocke les méta-données (données de description) qui listent les vidéos disponibles dans une catégorie donnée, leur titre, leur durée, leur coût, etc...

Les composants de la MIDlet Java

Voici une description des principaux éléments du navigateur ESG. Chaque élément correspond grosso modo à une classe différente.

Le **Contrôleur** est responsable de la navigation à proprement parler : il gère le passage d'une page à une autre et l'historique des pages. Lorsque l'utilisateur appuie sur le bouton de direction gauche, il retourne sur la page précédente ; en appuyant sur la direction droite, il retourne vers la page suivante.

La **Session** mémorise les choix de l'utilisateur dans chaque menu au fur et à mesure de sa navigation. La Session est transmise de page en page par le Contrôleur. Dans l'exemple ci-dessus, l'écran de détails, qui sait que l'utilisateur a sélectionné la vidéo Basket peut alors interroger une Requête pour obtenir les méta-données correspondantes.

Une **Requête** permet d'obtenir des informations stockées dans les méta-données téléchargées. Par exemple, l'écran de détails va interroger une Requête pour savoir quelles sont les informations (titre, prix, durée, résumé) associées à un identifiant de vidéo.

Chaque page est un **Document**, composé de différents contrôles graphiques : **Image**, **Texte**, **Rectangle**, **Ellipse**, **Liste**, **Lien Hypertexte**, ...

Les pages sont construites dynamiquement à partir des méta-données. C'est le rôle de l'**Assembleur de Pages**. Il part d'un modèle de présentation de page en SVG binarisé ; à partir de ce modèle, il construit les différents contrôles de la page (liste, texte, image, rectangle, ...) et leur associe les données dynamiques issues des méta-données (par exemple, des identifiants de vidéo et des titres pour la page TOP 10).

Répartition des tâches et Intervenants

- Thibault Franchini (ExpWay) s'est occupé du serveur de vidéos et de méta-données et des Requêtes manipulant les méta-données.

- Fabien Delpiano (PastaGames) a pris en charge la définition de l'architecture, le développement du squelette et des outils de base, ainsi que la majeure partie des contrôles graphiques, des transformations de repères 2D (à base de matrices de transformation) et des animations. Il a également développé "en dur" la version de référence des modèles de pages, que j'ai réencodées par la suite.

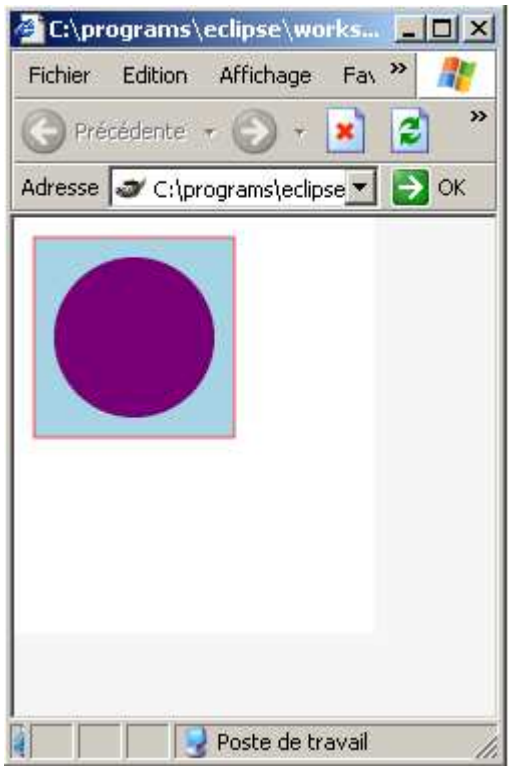
- Personnellement, j'ai développé l'encodeur des modèles de pages SVG, que je présente plus en détails ci-dessous, et le décodeur qui construit ces modèles de pages en Java sur le téléphone portable à partir des fichiers SVG encodés (Assembleur de Pages). Il s'agit d'un encodeur et d'un décodeur développés spécifiquement pour le sous-ensemble de SVG utilisé dans ce projet.
  J'ai également pris la suite de Fabien Delpiano en fin de projet sur le développement et l'optimisation de contrôles graphiques, ainsi que sur l'amélioration de l'ergonomie : animations d'attente, homogénéité des interfaces d'une page à l'autre, dans le respect d'une charte graphique commune.

MOTEUR SVG, LES ÉTAPES DE TRANSFORMATION

svgCircleAboveRect.svg (347 octets)

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<svg width='180' height='208' viewBox='0 0 180 208'
     xmlns='http://www.w3.org/2000/svg'
     xmlns:xlink='http://www.w3.org/1999/xlink'
>
   <rect x='10' y='10' width='100' height='100' fill='lightblue' stroke='red'/>
   <circle cx='60' cy='60' r='40' fill='purple' stroke='none' />
</svg>
```

Voici un exemple de fichier SVG très simple. Ce fichier respecte le standard défini par le W3C, et peut donc s'afficher correctement dans un visualiseur compatible ; en voici en aperçu donné par SVG Viewer de Adobe (installé comme plugin d'Internet Explorer).



Le but de l'encodage binaire est de réduire énormément la taille des données en supprimant toutes les **informations de structure** pour ne conserver que les **valeurs** (chiffres et textes), en **rouge** dans l'exemple précédent.

Après encodage, on obtient le fichier *svgCircleAboveRect.bin* qui ne pèse plus que 137octets, contre 347octets pour le fichier SVG *svgCircleAboveRect.svg*. Un fichier XML est en effet très redondant pour répondre au double besoin de relative lisibilité pour un humain et de facilité de parcours par un analyseur : par exemple, le nom des éléments est répété dans leur balise d'ouverture (`<élément>`) et dans leur balise de fermeture (`</élément>`).

Mais que deviennent les informations de structure ? Ces informations sont très importantes, il ne suffit donc pas de les supprimer, sinon le fichier binaire ne contiendrait plus que des valeurs vides de sens. Au niveau du décodage, la connaissance correspondant à ces informations de structure est répartie entre le fichier binaire et le décodeur :

- Dans le fichier binaire, on conserve de manière explicite mais très allégée des indications sur le type des éléments utilisés : on donnera par exemple un code à chaque élément, et on fera précéder les valeurs des attributs de cet élément par son code. Pour simplifier le décodage, on stockera également le nombre de ses éléments enfants (Dans l'exemple, l'élément `<svg>` a deux enfants, `<rect>` et `<circle>`).

- Dans l'encodeur, on incorpore de manière implicite la structure des données ; le fait par exemple qu'un élément `<rect>` possède des attributs x, y, width, height, fill et stroke est incorporé dans l'algorithme de décodage : le code Java ci-dessous indique que pour lire les valeurs d'un élément `<rect>` (reconnu après avoir lu son code), il faut lire dans cet ordre les valeurs correspondant à sa position horizontale, sa position verticale, sa largeur, sa hauteur, sa couleur de remplissage et sa couleur de trait.

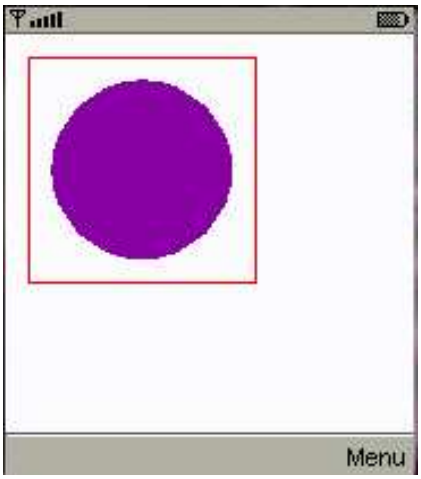
```
private void decodeRect(DataInputStream data) {

  int x = data.readInt();
  int y = data.readInt();
  int width = data.readInt();
  int height = data.readInt();
  int fillColor = data.readInt();
  int strokeColor = data.readInt();
  (...)
}
```

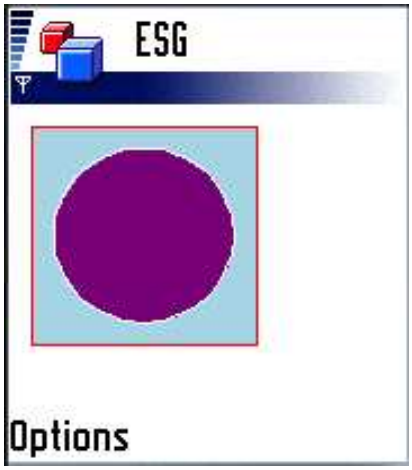
Plus précisément, on effectue d'abord un **prétraitement** des données SVG pour résoudre certaines ambiguïtés et simplifier les traitements ultérieurs (par exemple décomposer une animation complexe en animations plus simples). L'encodeur passe alors à l'action : il parcourt le fichier XML en entrée et encode les données de chaque élément à la suite pour obtenir le fichier binaire de sortie.

*On appelle **codec** l'ensemble de l'algorithme d'encodage d'un type de données et de l'algorithme de décodage inverse qui permet de retrouver la donnée de départ. Par abus de langage, on parle parfois de codec pour le code de décodage seul.*

Pour éviter de devoir ajouter des classes de décodages qui occuperaient une partie de l'espace mémoire limité réservé à notre application, on tire parti au maximum des fonctionnalités déjà présentes dans les bibliothèques Java du téléphone. Ainsi, on souhaite utiliser les méthodes `readByte`, `readShort`, `readInt`,... de la classe Java `DataInputStream` fournie. Pour cela, il est nécessaire d'utiliser l'autre partie des codecs lors de l'encodage : on utilise donc les méthodes `writeByte`, `writeShort`, `writeInt`,... de la classe Java `DataOutputStream` côté encodeur.

Les deux images ci-dessous illustrent le résultat du décodage sur deux configurations de téléphones *(émulateur MIDP 1.0 et émulateur Nokia Série 60)*.





Le résultat sur un téléphone Nokia est très proche de la version de référence dans le visualiseur d'Adobe, aux problèmes de crénelage près. Par contre, la version à gauche montre une limitation du profil Java MIDP 1.0 seul : il ne propose pas de librairie pour le remplissage des polygones ; ceci explique pourquoi l'intérieur du rectangle n'a pu être colorié.

Une plate-forme évolutive

Grâce à son architecture modulaire et la souplesse offerte par la définition des modèles de pages sous forme de données SVG binarisées, cette plate-forme de guide de services électroniques est très évolutive. Comme on peut le constater sur les exemples ci-dessous, il devient très simple, en suivant cette démarche, de modifier en profondeur l'apparence et même le comportement de l'application.

Les images ci-dessous illustrent différentes apparences pour notre premier scénario, le service de téléchargement de vidéos à la demande.

La souplesse de cette plate-forme nous permet d'implémenter simplement un scénario différent : un programme télé qui permet à l'utilisateur de sélectionner un programme et de déclencher son enregistrement, sur son magnétoscope numérique chez lui, à partir d'un téléphone portable.

Dans le scénario téléchargement de vidéos à la demande, l'utilisateur pouvait accéder à des listes de vidéos, sélectionnées par le fournisseur de contenus (Top 10 et Best Of) ou bien classées par genre/sous-genre, par mots-clefs, ou par collection. En sélectionnant une vidéo, il obtenait un résumé détaillé, son prix et sa durée, et pouvait la télécharger ; la vidéo serait alors facturée sur son forfait.

Dans le scénario Programme Télé, l'utilisateur peut accéder aux programmes télé à partir de la liste des chaînes et d'un calendrier avec les différentes périodes de la journée. Lorsqu'il sélectionne un programme télé, il obtient plus de détails sur ce programme ; il pourrait alors déclencher l'enregistrement du programme sur son magnétoscope connecté au boîtier ADSL de son décodeur de télé numérique (non implémenté dans la démonstration).

Ces deux applications sont différentes fonctionnellement, mais grâce à la souplesse de notre moteur de présentation, elles ont pu réutiliser la même technologie sous-jacente.

ETAPES DU DÉVELOPPEMENT DE L'OUTIL BINARY 4 JAVA

Je présente ici les grandes étapes du développement de l'outil Binary 4 Java, réalisé pendant la deuxième moitié de mon stage. Cette présentation met l'accent sur les aspects fonctionnels de l'outil, du point de vue de l'utilisateur.

J'aborderai à nouveau cet outil à l'occasion de la dernière partie de ce rapport, en me plaçant cette fois dans la perspective du développeur : démarche de conception, justification des choix technologiques et architecture.

Le développement de l'outil B4J a suivi les étapes suivantes, qui ne sont pas sans rappeler le processus de développement d'un jeu vidéo :

- Une phase de **pré-production**
  - analyse des besoins et réflexions préliminaires
  - premier prototype pour valider la faisabilité technique des solutions envisagées
  - suite aux développements d'encodeurs/décodeurs pour le guide de services, retours d'expérience et analyse des bonnes pratiques et des travers à éviter

- Une phase de **production**
  - génération de l'encodeur, du décodeur et d'un décodeur de test
    par itérations successives de **conception** et de **développement**, en améliorant l'architecture de l'outil à chaque itération

- Une phase de **test utilisateur**
  - l'utilisateur prend possession de l'outil et, face à ses difficultés,
  - l'ergonomie est repensée et améliorée

- Une phase de **distribution**
  - formation et documentation du produit (tutoriaux, références)
  - deux niveaux de packaging et de support, pour une version standard et une version étendue

Comme nous l'avons vu dans la première partie de ce rapport, Fabien Delpiano est à l'origine de l'outil, dont l'idée a germé à force d'écrire des classes d'encodage et des classes de décodage "jetables" d'un projet de jeu vidéo à l'autre.

En lui proposant ma candidature, j'avais dans l'a priori que la technologie XSLT, que j'avais eu l'occasion d'utiliser sur plusieurs projets importants auparavant, serait une solution efficace pour le développement de cet outil. En effet, XSLT est une technologie dédiée à la transformation de documents XML, et l'idée de base de ce projet était de transformer une grammaire XML en deux textes particuliers, le code source d'un encodeur d'une part, et le code source d'un décodeur en Java d'autre part.

Fabien Delpiano est resté très ouvert sur le choix de la technologie, et je le remercie de m'avoir laissé une très grande liberté dans la conception de cet outil.

RETOURS D'EXPÉRIENCE SUITE AUX DÉVELOPPEMENTS SPÉCIFIQUES

Avantages et Inconvénients de la génération de code sur l'écriture manuelle

L'écriture d'un encodeur et d'un décodeur de modèles de pages au format SVG dans le cadre du projet ESG m'a permis de découvrir par la pratique les contraintes de ces tâches sur un projet réel.

Le premier constat est que ce code est très compliqué et assez difficile à faire évoluer. En effet, lorsque l'on modifie le format des données en entrée, par exemple pour ajouter le support de nouveaux éléments SVG, il faut :

- modifier les données SVG d'exemple
- modifier les traitements d'encodage
- tester l'encodage
- modifier le décodeur
- tester le décodeur sur les nouvelles données
- documenter la modification du format de données binaires

En effet, je devais maintenir à des fins de documentation une spécification du sous-ensemble de SVG supporté et du format des fichiers binaires encodés.

A ce stade, il est important de s'interroger sur les avantages et inconvénients respectifs de la génération de code et de l'écriture manuelle. En effet, si les contraintes d'utilisation de l'outil sont plus importantes que ses bénéfices, par exemple s'il est plus compliqué de l'utiliser que d'écrire des encodeurs et décodeurs spécifiques, l'outil perd tout son intérêt.

Comparons le scénario précédent avec le scénario correspondant à l'utilisation de notre outil de génération de code. Il faut désormais :

- modifier les données SVG d'exemple
- modifier le schéma XML de définition des données supportées
- regénérer le code de l'encodeur et du décodeur, dont le fonctionnement est garanti car les données sont validées et le code est généré.

Alors que nous utilisions une description dans un format libre (texte) dans le premier cas, l'utilisation de l'outil nous force à formaliser la définition de nos données dans un schéma XML. Ce schéma nous permettra également de vérifier la validité des données d'exemple (et plus tard celle des données de production).

Si l'on fait le bilan, le code écrit à la main est souvent plus performant car l'humain tire mieux parti des spécificités des données pour les encoder en un minimum d'octets. Toutefois, il est aussi plus difficile à modifier car chaque modification est source d'erreurs potentielles, et il est beaucoup plus long à écrire.

Au contraire, le code généré peut contenir des erreurs aux débuts de l'outil, mais il est de plus en plus fiable au fur et à mesure de son développement car les erreurs ne sont pas corrigées une seule fois, mais pour toutes les regénérations de ce code ou d'un code semblable dans le futur.

A condition de veiller à la simplicité d'utilisation de l'outil, la génération de code se présente donc comme une approche très prometteuse pour la productivité des développeurs sur des tâches récurrentes et bien connues.

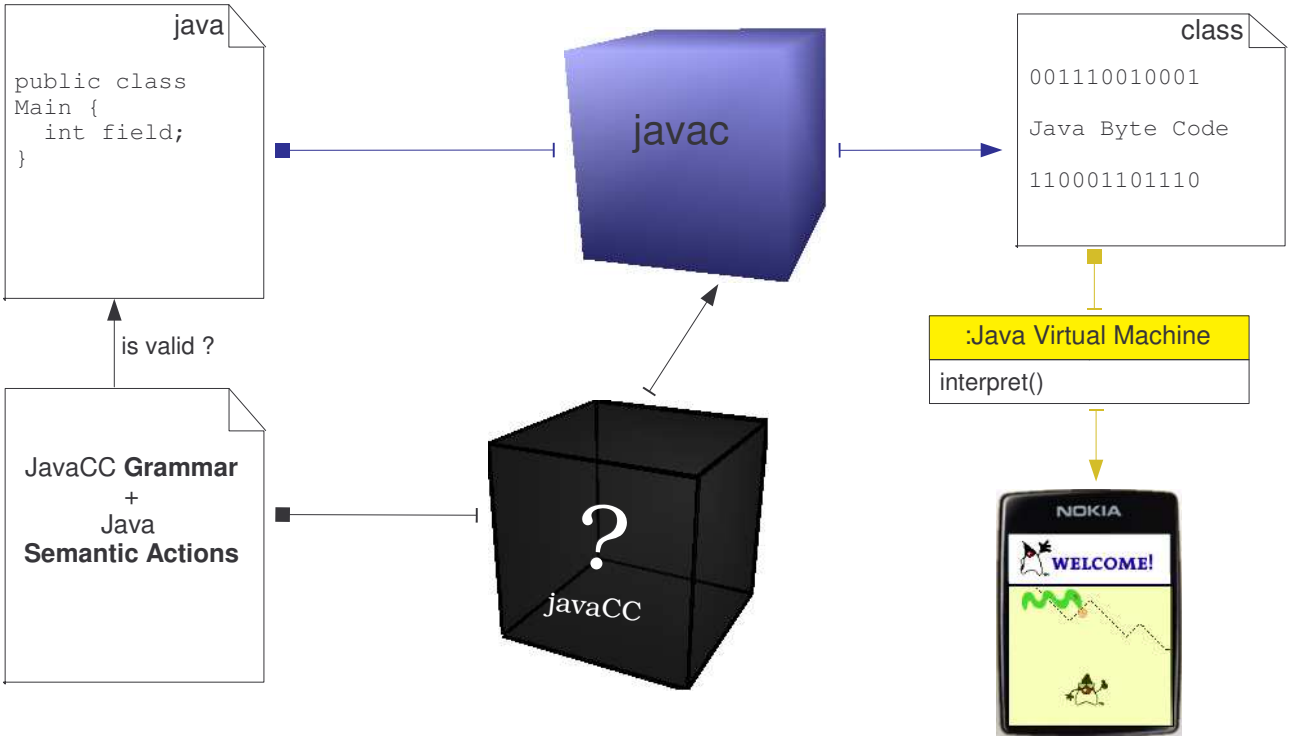
TESTS D'UTILISABILITÉ ET ERGONOMIE

Une fois l'outil pleinement fonctionnel, le développement est encore loin d'être terminé : reste à lui faire subir l'épreuve du feu, le test par l'utilisateur final. Or, il est inutile de tricher avec l'utilisabilité : dès les premiers contacts de l'utilisateur avec l'outil, on découvre les difficultés qu'il rencontre. Il est donc important d'intégrer cette démarche ergonomique dès le début du projet et de prévoir le temps nécessaire pour faire les ajustements nécessaires, même une fois que l'outil peut être considéré comme complet.

Après avoir validé la faisabilité, j'ai donc pris le temps, avec Fabien Delpiano, l'utilisateur final, de réfléchir à la manière dont il se servirait de l'outil avant même que la première ligne de code ne soit écrite.

L'utilisateur final de l'outil B4J est un développeur. Il va communiquer avec l'outil par le biais d'un ou de plusieurs fichiers textes qui doivent décrire :
- la grammaire XML (XML Schema) suivie par ses données
- les actions à faire lors du décodage, à savoir la création d'objets Java à partir des données décodées

Suite à cette discussion, Fabien Delpiano m'a aiguillé sur les compilateurs de compilateurs, en particulier JavaCC (Java Compiler Compiler), qui fonctionnent sur le même principe. Un compilateur est un programme qui transforme le code source d'un programme (un "simple" fichier texte) en un exécutable (les instructions machines sous forme binaire). De nos jours, les compilateurs ne sont plus écrits à la main, mais ils sont générés par un compilateur de plus haut niveau : le compilateur de compilateur. Le compilateur de compilateur accepte en entrée une définition de la syntaxe du langage source (C, Java, ...), annotée par des actions sémantiques, qui sont les actions à effectuer lors de la compilation lorsqu'un élément de la grammaire (une fonction, une variable, une classe, une boucle, ...) est reconnue. La syntaxe utilisée pour décrire la grammaire annotée varie d'un outil à l'autre, mais le principe reste le même.



Je me suis fortement inspiré de cet existant pour le modèle d'utilisation de l'outil B4J : XML Schema prévoit, de manière systématique, la possibilité d'ajouter des annotations libres sur chacune des définitions de la grammaire ; nous exploitons ce principe en ajoutant le code Java à exécuter lors du décodage (nos actions sémantiques) comme annotations de la grammaire XML Schema.

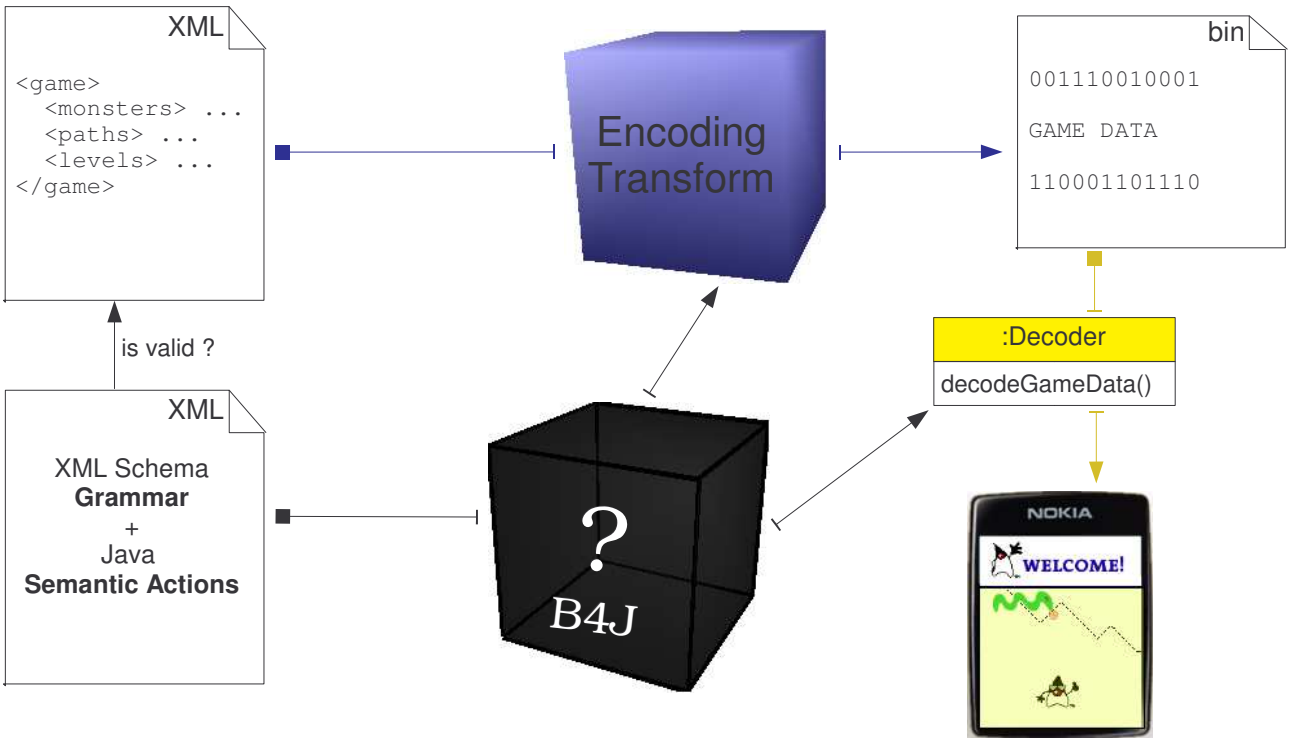
L'exemple ci-dessous présente un extrait de grammaire XML annotée. Il s'agit de la définition d'un type d'élément, "Quizz", qui possède un attribut "topic" et deux enfants "questions" et "result". Les actions sémantiques (en gras) sont insérées dans l'élément `<xsd:appinfo>` prévu pour les annotations destinées à un programme.

Les annotations sont elles-mêmes en XML ; nous avons défini nos propres balises, qui commencent par "a4j:" (il s'agit d'un espace de noms différents de celui du schéma, "xsd:"), pour structurer le code Java des actions sémantiques. Dans cet exemple, chaque balise délimite du code Java exécuté à des étapes différentes du décodage des éléments de type "Quizz".

GuessMeRight.xsd (extrait)

```
<xsd:complexType name="Quizz">
    <xsd:annotation>
      <xsd:appinfo>
        <a4j:events>
          (...)
          <a4j:onElementStart>
            // User Java Code
          </a4j:onElementStart>
          <a4j:onAttributesEnd>
            if (isAtTopicPresent) {
              System.out.println("Le thème de ce questionnaire est : "+atTopic);
            }
          </a4j:onAttributesEnd>
          (...)
          <a4j:onElementEnd>
            // User Java Code
          </a4j:onElementEnd>
        </a4j:events>
      </xsd:appinfo>
    </xsd:annotation>
    <xsd:sequence>
      <xsd:element ref="gmr:questions" minOccurs="1" maxOccurs="1"/>
      <xsd:element ref="gmr:result" minOccurs="1" maxOccurs="1"/>
    </xsd:sequence>
    <xsd:attribute name="topic" type="xsd:string"/>
  </xsd:complexType>
```

D'un point de vue conceptuel, l'outil B4J est donc un compilateur de compilateur, avec XML comme langage source et Java comme plate-forme d'exécution. En comparant le schéma suivant avec celui de la page précédente, on constate une différence remarquable de B4J avec les compilateurs de compilateurs classiques. Au contraire de javaCC qui ne génère que le compilateur, B4J génère à la fois le compilateur et l'interprêteur du langage dont on lui fournit une définition, à savoir l'encodeur et le décodeur des données.


XML
<game>
<monsters> ...
<paths> ...
<levels> ...
</game>
Encoding Transform
bin
001110010001
GAME DATA
110001101110
:Decoder
decodeGameData()
is valid ?
XML
XML Schema
Grammar
+
Java
Semantic Actions
?
B4J
NOKIA
WELCOME!

FORMATION ET DOCUMENTATION

Dans le développement d'un outil, la formation et la documentation jouent également un rôle capital. La question n'est donc pas de documenter ou pas, mais de savoir quoi documenter et comment ? Faute de temps, la documentation de l'outil B4J reste encore malheureusement assez sommaire ; il était donc fondamental pour moi de concentrer mes efforts sur la documentation indispensable.

J'ai choisi de peu documenter l'implémentation, mes choix d'architecture, les librairies utilisées pour le développement, mais de privilégier la documentation de l'utilisation de l'outil. J'ai en particulier mis l'accent sur la méthode qui me semble la plus efficace pour prendre en main l'outil rapidement :

1. Commencer par écrire un fichier XML de données d'exemple.

2. Ecrire le schéma XML qui permet de valider ces données d'exemple. L'outil B4J propose des fonctionnalités d'aide à l'édition de ce schéma ; il permet également de tester la validité du schéma par rapport au sous-ensemble de XML Schema supporté.

3. A ce stade, vérifier que le schéma valide bien le fichier XML d'exemple. Il est déjà possible de générer l'encodeur, mais pas encore le décodeur.

4. Utiliser B4J pour ajouter les annotations vides dans le schéma.

5. Compléter les annotations vides par les actions sémantiques en Java.

6. Générer l'encodeur et le décodeur, qui est prêt à être intégré au projet de jeu ou d'application mobile de l'utilisateur.

J'ai décrit ce processus d'utilisation recommandé dans un document synthétique d'une page, et je l'ai développé et appliqué à un exemple concret dans un tutoriel qui visite pas à pas les fonctionnalités principales de l'outil.

Ce tutoriel m'a ensuite servi de base pour la formation que j'ai dispensée à Fabien Delpiano (PastaGames) et Thibault Franchini (ExpWay) à la fin de mon stage.

J'ai pris comme exemple, pour ce tutoriel, le développement d'un jeu très simple, en mode texte, qui s'exécute directement sur PC et non sur le téléphone portable. En effet, si l'outil B4J a pour vocation de générer des décodeurs qui s'exécutent sur un téléphone portable, il ne s'y limite pas.

Ce jeu simple est un Quizz pour deux joueurs, qui s'inspire des tests de personnalités qu'on trouve fréquemment dans les magazines. Il s'agit d'un jeu collaboratif : le premier joueur ne répond pas au quizz, mais essaie de deviner quelles réponses son partenaire va choisir. Le deuxième répond sincèrement aux questions (ou pas, selon le mode de jeu).

Pour cet exemple, notre fichier XML de données va contenir la liste des questions du Quizz, avec pour chaque question la réponse choisie par le premier joueur. On suppose que les deux joueurs ne jouent pas en même temps, mais que le premier joueur a fini de jouer et invite le second joueur à répondre au questionnaire, par exemple en lui envoyant le fichier de données encodé sur son téléphone portable.

Voici une question, telle qu'elle apparaît dans le fichier de données XML :

```
<question title="Une porte..." prediction="2">
  <text>Vous êtes dans un donjon ; alors que vous arrivez devant une porte,
    vous entendez des craquements étranges de l'autre côté. Que faites-vous ?
  </text>
  <answer category="2">j'ouvre la porte, avec appréhension</answer>
  <answer category="1">je passe devant en sifflotant,
    et je cherche un autre chemin
  </answer>
  <answer category="0">je défonce la porte joyeusement</answer>
</question>
```

Comme on le voit, l'utilisateur peut choisir librement la représentation de ses données (choix des éléments et des attributs qui peuvent apparaître dans ses fichiers de données).

Voici à présent la définition de l'élément `<question>` en XML Schema annoté. Le code Java saisi par l'utilisateur au sein de ces annotations est en gras.

```
<xsd:complexType name="Question">
  <xsd:annotation>
    <xsd:appinfo>
      <a4j:events>
        <a4j:context>
          <a4j:childContinuation>categoryMatching, i</a4j:childContinuation>
        </a4j:context>
        <a4j:onElementStart>
          byte[] categoryMatching = new byte[3];
        </a4j:onElementStart>
        <a4j:onEachChildEnd fromChild="String text" position="i">
          if (text != null) {
            System.out.println("Q. "+i+") "+text);
          }
        </a4j:onEachChildEnd>
        <a4j:onElementEnd>
          String userChoice = readKeyBoardInput();
          while(
             !(
                  userChoice.equals("1")
               || userChoice.equals("2")
               || userChoice.equals("3")
              )
          ) {
            System.out.println("1 2 or 3 expected...");
            userChoice = readKeyBoardInput();
          }
          int choice = Integer.parseInt(userChoice);
          int category = categoryMatching[choice-1];
          categoryCount[ category ]++;

          if (category == atPrediction)
            teamScore++;
          categoryMatching = null;
        </a4j:onElementEnd>
      </a4j:events>
    </xsd:appinfo>
  </xsd:annotation>
  <xsd:sequence>
    <xsd:element ref="gmr:text" minOccurs="1" maxOccurs="1"/>
    <xsd:element ref="gmr:answer" minOccurs="3" maxOccurs="3"/>
  </xsd:sequence>
  <xsd:attribute name="title" type="xsd:string"/>
  <xsd:attribute name="prediction" type="xsd:byte" use="required"/>
</xsd:complexType>
```

A partir de la grammaire XML Schema définie par l'utilisateur, l'outil B4J génère un encodeur en XSLT et un décodeur en Java.

Voici d'abord l'extrait du décodeur Java qui correspond au décodage de l'élément `<question>` (le code saisi par l'utilisateur est à nouveau en gras) :

```
private static void decodeQuestion( DataInputStream data ) throws IOException {

    byte[] categoryMatching = new byte[3];
    byte presenceMaskFromTitle = data.readByte();
    boolean isAtTitlePresent = (  ( presenceMaskFromTitle&0x1 )!=0  );
    String atTitle = null;
    if ( isAtTitlePresent ) {
        atTitle = data.readUTF();
    }
    byte atPrediction = data.readByte();

    short __sequenceCount1 = 1;
    short __sequenceCount2 = 3;
    short childCount = (short)( __sequenceCount1 + __sequenceCount2 );

    {
        short i = 0;
        for (short __i=0; __i<__sequenceCount1; __i++) {
            String text = decodeText(data, categoryMatching, i);

            if (text != null) {
                System.out.println("Q. "+i+") "+text);
            }
            i++;
        }
        for (short __i=0; __i<__sequenceCount2; __i++) {
            String text = decodeAnswer(data, categoryMatching, i);

            if (text != null) {
                System.out.println("Q. "+i+") "+text);
            }
            i++;
        }
    }

    String userChoice = readKeyBoardInput();
    while(
             !(
                  userChoice.equals("1")
               || userChoice.equals("2")
               || userChoice.equals("3")
              )
          ) {
        System.out.println("1 2 or 3 expected...");
        userChoice = readKeyBoardInput();
    }
    int choice = Integer.parseInt(userChoice);
    int category = categoryMatching[choice-1];
    categoryCount[ category ]++;

    if (category == atPrediction)
        teamScore++;
    categoryMatching = null;

}
```

Et voici à présent l'extrait de l'encodeur généré correspondant :

```
<gxslt:template match="gmr:question">
  <isob4j:element description="question">
     <isob4j:mask>
        <gxslt:attribute name="bit1">
           <gxslt:call-template name="bitIsPresent">
              <gxslt:with-param name="node" select="@title"/>
           </gxslt:call-template>
        </gxslt:attribute>
        <gxslt:attribute name="bit1Description">is title present
        </gxslt:attribute>
     </isob4j:mask>
     <gxslt:if test="@title">
        <isob4j:string>
           <gxslt:attribute name="description">title</gxslt:attribute>
           <gxslt:value-of select="@title"/>
        </isob4j:string>
     </gxslt:if>
     <isob4j:byte>
        <gxslt:attribute name="description">prediction</gxslt:attribute>
        <gxslt:value-of select="@prediction"/>
     </isob4j:byte>
     <gxslt:apply-templates/>
  </isob4j:element>
</gxslt:template>
```

En fait, cet encodeur génère d'abord un fichier intermédiaire, dans un format spécifique de l'outil Binary 4 Java ; voici ce qu'on obtient en sortie de l'encodeur pour l'élément `<question>` que nous avons pris comme exemple :

```
<isob4j:element description="question">
  <isob4j:mask bit1="1" bit1Description="is title present"/>
  <isob4j:string description="title">Une porte...</isob4j:string>
  <isob4j:byte description="prediction">2</isob4j:byte>
  <isob4j:element description="text">
     <isob4j:string description="text value">Vous êtes dans un donjon ;
        alors que vous arrivez devant une porte, vous entendez des craquements
        étranges de l'autre côté. Que faites-vous ?
     </isob4j:string>
  </isob4j:element>
  <isob4j:element description="answer">
     <isob4j:byte description="category">2</isob4j:byte>
     <isob4j:string description="text value">j'ouvre la porte, avec appréhension
     </isob4j:string>
  </isob4j:element>
  <isob4j:element description="answer">
     <isob4j:byte description="category">1</isob4j:byte>
     <isob4j:string description="text value">je passe devant en sifflotant,
        et je cherche un autre chemin
     </isob4j:string>
  </isob4j:element>
  <isob4j:element description="answer">
     <isob4j:byte description="category">0</isob4j:byte>
     <isob4j:string description="text value">je défonce la porte
       joyeusement
     </isob4j:string>
  </isob4j:element>
</isob4j:element>
```

L'extrait ci-dessus décrit les instructions à destination d'un utilitaire d'encodage simple qui est indépendant du format des données de l'utilisateur.

PACKAGING ET SUPPORT À DEUX NIVEAUX

A l'issue des tests utilisateur, Fabien Delpiano m'a demandé le développement de fonctionnalités complémentaires pour gérer des problèmes plus particuliers :
- la résolution des références (mécanisme id/ref)
- l'inclusion de fichiers de données externes

Résolution des références

Certains attributs dans un fichier XML jouent un rôle particulier, il s'agit des identifiants et des références. Par exemple, en XHTML, on peut accéder à une partie du document en définissant une ancre avec un identifiant : `<a id='partie1' />`. Un lien peut ensuite référencer cette partie du document de la manière suivante
`<a href='#partie1'>voir la première partie</a>`.

Lors du décodage, l'information `'partie1'` lue dans l'attribut `id` et dans l'attribut `href` sera uniquement utilisée pour relier ces deux éléments ; elle n'a aucun autre intérêt, et pourrait être remplacé par une autre valeur, on aurait toujours la même liaison. La résolution des références consiste à masquer cette information, et remplacer la valeur de la référence par une référence à l'objet Java identifié (ici l'objet Java qui représenterait une ancre). *L'utilisateur n'a donc plus besoin de stocker explicitement chaque ancre dans une table de hachage, ce qu'il devait faire sinon pour retrouver l'élément Java correspondant en lisant la valeur d'une référence.*

Inclusion de fichiers de données externes

L'inclusion de données externes consiste simplement à inclure les données d'une image ou d'un autre fichier de données dans notre fichier de données. Les données incluses deviennent alors accessibles au sein des actions sémantiques de décodage, au même titre que les valeurs issues des données de l'utilisateur.

J'ai choisi de ne pas inclure ces fonctions aux côtés des autres fonctions du code, mais de distinguer un noyau de code fournissant les fonctionnalités fondamentales, et des extensions pour les fonctionnalités supplémentaires. Pour autoriser la définition de modules d'extension, j'ai dû adapter le code existant : il m'a fallu le découper en un peu plus de modules pour permettre aux fonctions d'extensions d'insérer leurs traitements entre deux traitements du noyau de base.

La version standard de l'outil B4J est utilisable indépendamment des extensions, et peut être déployée sans ces extensions. L'intérêt de ce découpage est double :
- d'un point de vue marketing, il permet de distinguer deux niveaux de service
- d'un point de vue technique, le noyau est plus stable car les derniers développements, non finalisés, sont ajoutés comme extensions sans influencer le noyau de base.

GROS PLAN SUR... le développement pour mobiles

Un marché dynamique, en forte progression

Le marché de la téléphonie mobile est très important : au deuxième trimestre 2004, la France comptait plus de 42 millions d'abonnés au téléphone mobile. Avec une augmentation annuelle du nombre d'abonnés de 7%, c'est un marché dynamique, en évolution technologie constante, et le taux de renouvellement des équipements est rapide.

Abonnés Français au Téléphone Mobile (Journal du Net)

Abonnés français au téléphone mobile en 2004			
Opérateur	**Abonnés**	**Croissance trimestrielle**	**PDM**
T2 2004			
Orange	20.395.800	24.600	48,3 %
SFR	14.942.200	114.400	35,4 %
Bouygues Télécom	6.903.500	163.100	16,3 %
Dauphin Télécom	1.900	500	0 %
Total	42.243.400	302.600	100 %
T1 2004			
Orange	20.371.200	42.600	48,6 %
SFR	14.827.800	103.400	35,4 %
Bouygues Télécom	6.740.400	110.300	16,0 %
Dauphin Télécom	1.400	--	0 %
Total	41.940.800	257.700	100 %
Source : ART			*Mis à jour le 30/09/2004*

Avec le renouvellement des infrastructures, les débits disponibles augmentent régulièrement et de nouveaux services font leur apparition. Des services dits de troisième génération (UMTS), quittent actuellement la phase d'expérimentation :

- visiophonie,
- téléchargement de vidéos à la demande,

ou sont en cours d'expérimentation :

- télévision, ...

Malheureusement, ce marché est très hétérogène ; les applications que l'on peut développer pour un portable de dernière génération qui supporte Java, possède un écran couleur et une caméra, lit les vidéos et les mp3, et les téléphones qui datent de quelques années n'ont pas grand chose de comparable.

Il convient également de relativiser l'intérêt des utilisateurs pour ces nouvelles fonctions. De nombreux utilisateurs déclarent en effet n'utiliser leur portable que comme téléphone, et taper leurs numéros systématiquement, sans même utiliser la fonction de répertoire.

Les contraintes du développement pour mobiles

Même en se limitant aux applications Java, supposées être des modèles de portabilité conformément au slogan de Sun : *"Write Once, Run Everywhere"*, de nombreuses contraintes pèsent sur les applications pour téléphones mobiles. En pratique, ces contraintes obligent les développeurs à décliner leurs applications en différentes versions en fonction des modèles de téléphone :

- Du point de vue **matériel** : la taille de l'écran, les fonctionnalités disponibles (appareil photo, rétro-éclairage, vibration, ...), les bugs d'une version spécifiques du téléphone. En particulier, comme les images ne peuvent être redimensionnées dynamiquement sur le téléphone, la plupart des images doivent être créées en différentes tailles en fonction du téléphone.

- Du point de vue **logiciel** : les bibliothèques disponibles sur le téléphone (profil MIDP 1.0 ou 2.0 de Sun, profil Doja 1.5 ou 2.5 de DoCoMo), la taille du code qui limite le nombre de classes et de méthodes que l'on va pouvoir définir.

- Du point de vue des **données**, la taille disponible pour les stocker et la manière dont elles doivent être fournies (empaquetées avec le code compressées dans une archive JAR ou bien téléchargée via HTTP). Il faut également tenir compte du coût du téléchargement de ces données pour l'utilisateur, de la latence des requêtes aériennes (OTA ou Over-The-Air) et des problèmes induits par la mobilité (coupures fréquentes de transmission, pause de l'application à la réception d'un appel).

Contraintes liées au développement sur mobiles (conférence de Fabien Delpiano)

	MIDP 1.0	MIDP 2.0	DoJa 1.5	DoJa 2.5
❖ **Taille du code**	Dépend du terminal 30ko < … < 4Mo avec les ressources	Dépend du terminal 30ko < … < 4Mo avec les ressources	30ko sans les ressources	30ko sans les ressources
❖ **Taille des données**	20ko < .. < 4Mo	Toute la mémoire disponible sur le téléphone	100ko	200ko
❖ **Divers petits écrans**	128x128 < < 208x276	176x208 (6600)	128x128 << 240x255	320 x 240
❖ **Petit footprint**	50Ko < … < 8mo	< 6mo	~ 1mo	?
❖ **Performance**	Très variables !	idem	idem	idem
❖ **Sécurité** • Accès ressources locales • Accès réseau	Non	Non	Non Uniquement sur le site de chargement de l'appli	Oui, mais sécurisé Uniquement sur le site de chargement de l'appli
❖ **Peu de protocoles réseau disponibles**	http 1.1 obligatoire, pas de https Parfois TCP et UDP	http 1.1obligatoire, https optionnel Parfois TCP et UDP	http & https	http & https
❖ **Passerelles opérateur contraignantes**	Proxy http Rigueur nécessaire	idem	Idem	Idem
❖ **Disparité des terminaux**	APIs supplémentaires non standard Bugs Firmware	Bugs Firmware	Peu de bugs firmware connus	

Source: http://kissen.cs.uni-dortmund.de:8080/devicedb/index.html

Outils utilisés et étapes de construction d'une application

Les outils utilisés pour le développement d'une application Java pour téléphone portable sont en partie communs avec ceux utilisés couramment pour le développement d'une application Java pour PC :

- environnement de développement (IDE) Eclipse
- outil d'automatisation des tâches de construction Apache Ant
- outil de tests unitaires JUnit,
- etc...

Toutefois, comme la machine virtuelle Java du téléphone est allégée et fait ainsi moins de vérifications (typage, ...) sur les classes qu'elle exécute, ces vérifications doivent être faites lors d'une étape supplémentaire après la compilation, dite **prévérification**, et des indications sont ajoutées aux fichiers des classes pour les rendre exécutables sur le téléphone.

Sun fournit une suite d'outils de développement pour la plate-forme MIDP, le Sun Wireless Toolkit, pour réaliser les différentes étapes de construction et de vérification d'une application MIDP Java (ou MIDLet). C'est l'équivalent du Kit de Développement Java (Java SDK) pour le développement PC.

Cette suite comporte également des **émulateurs** qui permettent de tester la MIDLet sur PC dans des conditions proches de celles du téléphone. Le test sur l'émulateur ne dispense malheureusement pas de tester la MIDLet une deuxième fois sur le téléphone cible, car l'émulateur et le téléphone ont leurs propres bugs, et fréquemment des comportements différents. Les fabricants de téléphones fournissent eux aussi des émulateurs, correspondant à leurs différents modèles.

Pour faciliter l'utilisation de ces outils, le projet Antenna définit des balises d'extension pour intégrer les différentes tâches de construction d'une application mobile (compilation, prévérification, packaging) au sein des scripts Ant de construction du projet. Il inclut également un **précompilateur** Java, qui s'avère utile pour sélectionner ou commenter des parties du code en fonction du téléphone cible.

Les outils d'**obfuscation**, qui remplacent les noms de toutes les classes et méthodes par des noms courts (A, B, C,...) sont utilisés pour réduire la place occupée par le code : on peut ainsi conserver des identifiants significatifs pendant le développement, et les remplacer par des identifiants très courts mais incompréhensibles dans la version finale du code. Ces outils sont habituellements utilisés pour rendre illisible le code Java obtenu par rétro-ingénierie des exécutables.

Enfin, pour **distribuer** la MIDlet, il faut regrouper toutes ses classes et ses données dans une archive zip particulière, ou JAR (Java Archive), qui contient en plus un fichier de description de son contenu. Il faut parfois ajouter un autre descripteur, le fichier JAD (Java Application Description), qui contient un lien vers l'archive JAR de l'application ; envoyé seul sur le téléphone portable dans un premier temps, il permet à l'utilisateur de lancer le téléchargement de l'application à partir d'un serveur HTTP.

III.PRÉSENTATION DU TRAVAIL RÉALISÉ

Cette partie sera plus particulièrement consacrée à la présentation de l'outil Binary for Java, en mettant l'accent sur son processus de développement, son architecture et les technologies utilisées pour son développement. Plus généralement, après avoir présenté cet outil du point de vue de l'utilisateur, nous prenons ici le point de vue du développeur.

Dans un premier temps, j'évoquerai ma réflexion sur le modèle mental de l'utilisateur, un point clef du développement de cet outil. En effet, il m'a semblé fondamental pour développer un outil efficace de prévoir la **manière** dont il allait être utilisé, et quelle **perception** un utilisateur pouvait en avoir. Le concept de modèle mental de l'utilisateur recoupe ces deux notions.

Suite à cette réflexion, le développement de l'outil B4J a été grandement influencé par la volonté de proposer un modèle simple et cohérent à l'utilisateur, et de le faire apparaître sans artifices dans l'"interface textuelle" à travers laquelle il utilise l'outil. Ceci a guidé en particulier le choix des balises XML, qui décrivent les données de l'utilisateur et les actions à réaliser lors du décodage de ces données.

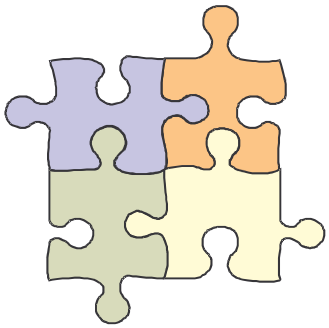
Le processus de développement de l'outil B4J a consisté à recenser les cas d'utilisation de l'outil à partir de l'analyse des besoins, à choisir des technologies puis plus précisément des techniques d'implémentation, à définir enfin une architecture technique compatible avec les implémentations choisies et propre à réaliser la satisfaction des besoins de l'utilisateur. Ces phases n'étaient pas distinguées de manière formelle et suivies séquentiellement, mais plutôt exécutées en parallèle avec des allers-retours entre les phases de conception et les phases de développement.

Le langage XSLT est le langage principal utilisé pour le développement de l'outil B4J. Je l'ai utilisé de manière assez originale pour faire un pont entre le monde des documents et le monde des objets, ce que j'évoquerai dans un dernier temps.

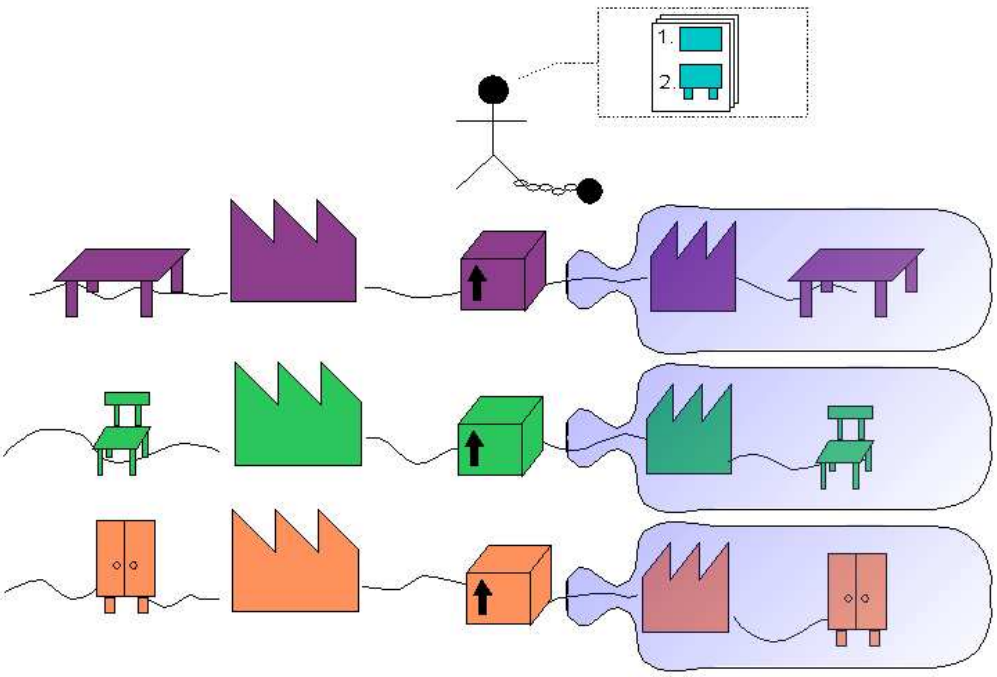
L'outil B4J a été déployé au sein de la société PastaGames. Dans sa dernière version, il intègre les fonctionnalités réclamées par Fabien Delpiano suite à ses premiers retours d'expérience. Des fonctionnalités avancées ont ainsi été ajoutées, en particulier la possibilité d'inclure un autre fichier (image, son, ou autre fichier de données).

Modèle mental de l'utilisateur

Un modèle est une représentation simplifiée d'un système, qui représente avec une précision plus ou moins grande un aspect donné de ce système. Au cours de l'utilisation d'un outil, un utilisateur se forge une représentation opérationnelle de cet outil, élaborée à partir de ses connaissances antérieures et des résultats de ses interactions avec l'outil ; il s'agit du modèle mental de l'utilisateur.

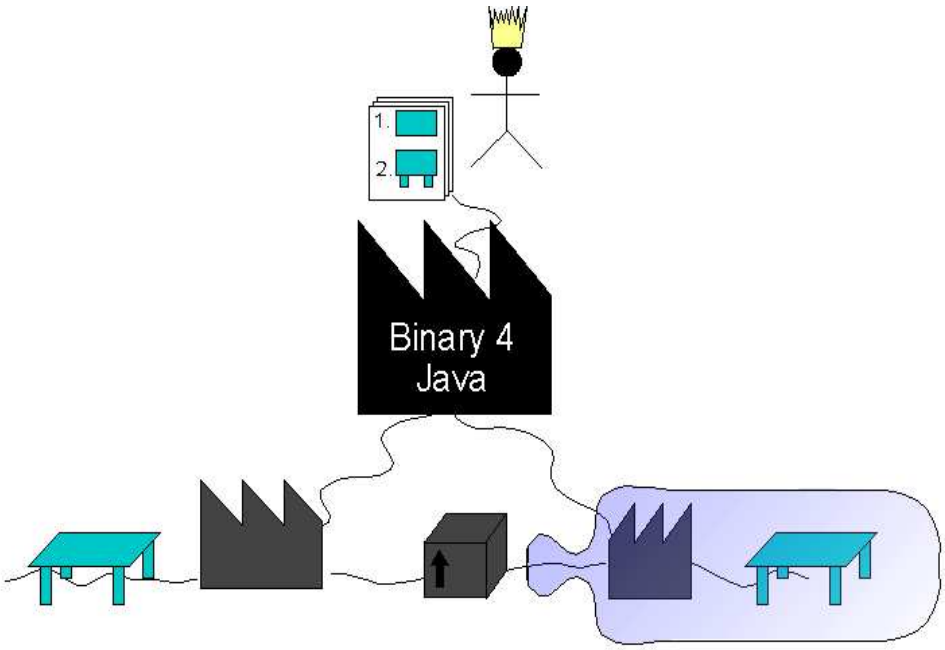


L'outil Binary 4 Java propose un modèle opérationnel basé sur la métaphore des meubles en kit. Dans cette métaphore, l'utilisateur veut construire des meubles dans une bouteille. Comme les meubles ne rentrent pas dans la bouteille, il doit d'abord les démonter, faire rentrer les pièces détachées dans la bouteille, puis réassembler le meuble à l'intérieur de la bouteille.

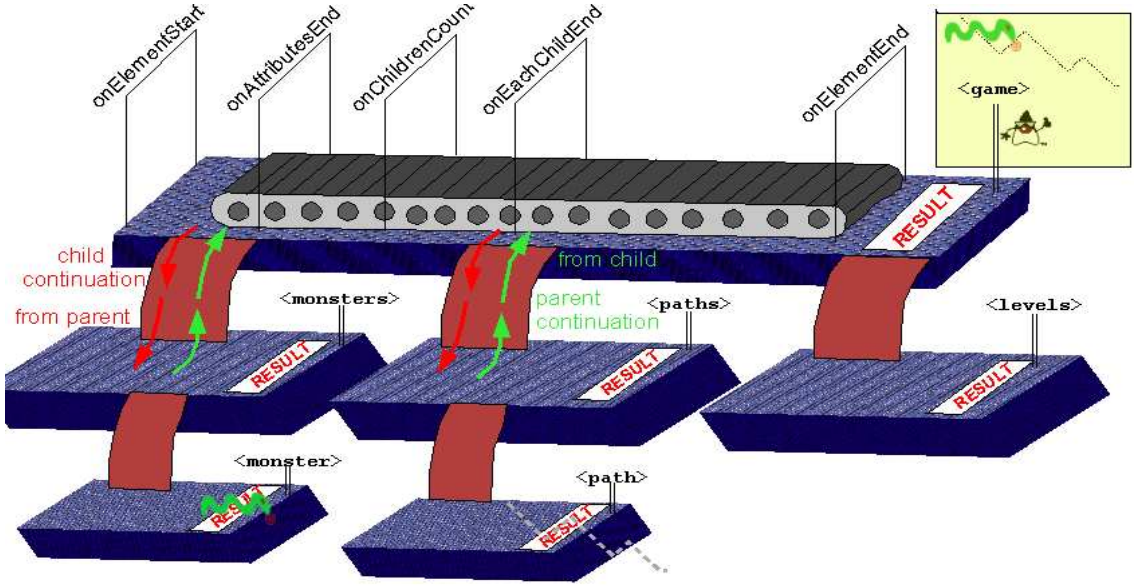


Le montage et le démontage des meubles sont confiés à deux usines. Mais chaque usine ne sait que respectivement monter et démonter un seul type de meuble. Pour chaque nouveau meuble, l'utilisateur doit donc faire construire une nouvelle usine de montage et de démontage. Malheureusement, il est le seul à connaître les plans de montage des meubles, et doit donc diriger personnellement et dans les détails la construction de toutes les usines. Il s'agit d'une tâche fastidieuse et répétitive.

Le but de l'outil Binary 4 Java est de décharger l'utilisateur de ce fardeau : l'utilisateur formalise sa connaissance du meuble à monter en écrivant un guide comprenant toutes les étapes du montage. A partir de là, l'outil Binary 4 Java prend en charge la construction des usines de montage et de démontage. Et pour modifier le design d'un meuble, l'utilisateur n'a plus qu'à mettre à jour son schéma de montage.



Rentrons désormais à l'intérieur de l'usine : on y trouve différentes chaînes de montage, correspondant aux différentes parties du meuble. Ces chaînes de montage sont assez indépendantes, mais s'échangent des pièces détachées ; par exemple, la chaîne chargée d'assembler un pied de la table reçoit des vis et les morceaux du pied de table, et renvoie le pied assemblé à la chaîne chargée de l'assemblage de la table complète.



Comme vous l'aurez compris, dans cette histoire, la bouteille représente le téléphone portable, une usine est un programme, le schéma de montage un schéma de grammaire XML, et un meuble correspond à des données en entrée et aux objets Java construits à partir de ces données en sortie.

Suite aux retours de Fabien Delpiano en utilisation sur des projets concrets, des fonctionnalités ont été ajoutées avec leur propre modèle utilisateur. Par exemple, pour la gestion des références entre les éléments, qui transforment l'arbre XML en graphe, nous utilisons le principe d'une **liste de publication**. Ce schéma (design pattern) bien connu des développeurs, consiste en une liste de messages classés par **sujet** (topic) sur lequel les **éditeurs** (publishers) publient des **messages** que les **abonnés** (subscribers) reçoivent.

Un noeud du graphe à l'origine d'un arc de référence joue le rôle d'abonné, le noeud destination celui d'éditeur et la liaison entre les deux noeuds correspond au sujet. Lors du décodage des différents noeuds, le premier noeud s'abonne à la liste de diffusion, le second y publie un message ; une fois les deux éléments décodés, le message arrive à l'abonné ce qui lui permet de créer une association entre les deux objets Java correspondant à lui-même et à l'éditeur.

Motivation du Choix de XSLT pour l'encodeur

En lisant *SVG Essentials* de J. David Eisenberg (O'Reilly, 2002) pour découvrir la technologie SVG que nous allions utiliser par la suite, j'ai découvert un exemple de transformation XSLT illustrant l'usage de classes d'extensions en Java. Il s'agissait de compléter les fonctionnalités disponibles en XSLT en écrivant des fonctions supplémentaires dans une classe Java et en utilisant une syntaxe particulière pour accéder à ces fonctions Java au sein de la transformation XSLT *(Chapter 12, Generating SVG, Using XSLT to Convert XML Data to SVG)*.

J'ai rapidement fait un parallèle avec notre problème d'encodage de données pour me rendre compte qu'en utilisant cette technique, nous pourrions écrire la majorité du code de l'encodeur en XSLT. A ce stade, j'avais déjà arrêté mon choix sur XSLT pour les générateurs de code, mais je ne pensais pas pouvoir écrire un encodeur de données Java en XSLT. En effet, XSLT manipule uniquement du texte, et les fichiers qu'il produit en sortie ne peuvent pas contenir de données binaires.

Dans notre cas, on contourne les limitations de XSLT en redirigeant la sortie standard des données vers notre classe Java grâce au mécanisme d'extension. Ce n'est plus XSLT qui écrit le fichier de sortie, mais une classe Java, ce qui permet non seulement d'écrire un fichier binaire, mais un fichier binaire basé sur les codecs Java disponibles sur le téléphone.

Voici un extrait de cette transformation XSLT montrant un exemple d'utilisation de l'extension Java (en gras):

```
<xsl:stylesheet version="1.0" xml:lang="en"
   xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
   xmlns:isob4j="http://eric.brechemier.name/2004/bin4java/isoXml"
   xmlns="http://www.w3.org/1999/xhtml"

   xmlns:java="http://xml.apache.org/xalan/java"
   xmlns:xmlbin="xalan://name.brechemier.eric"
>
  (...)
  <xsl:template match="isob4j:boolean">
    <xsl:variable name="Action_doEncode_boolean"
       select="xmlbin:JavaEncoder.writeBoolean(.)"
    />
  </xsl:template>
  (...)
</xsl:stylesheet>
```

Et voici l'extrait de la classe d'extension Java correspondante :

```
package name.brechemier.eric;
(...)

public class JavaEncoder
{
   protected DataOutputStream output;
   protected static JavaEncoder _singleton = new JavaEncoder();
   (...)

   public static void writeBoolean(boolean data) throws IOException
   {
      _singleton.output.writeBoolean(data);
   }
   (...)
}
```

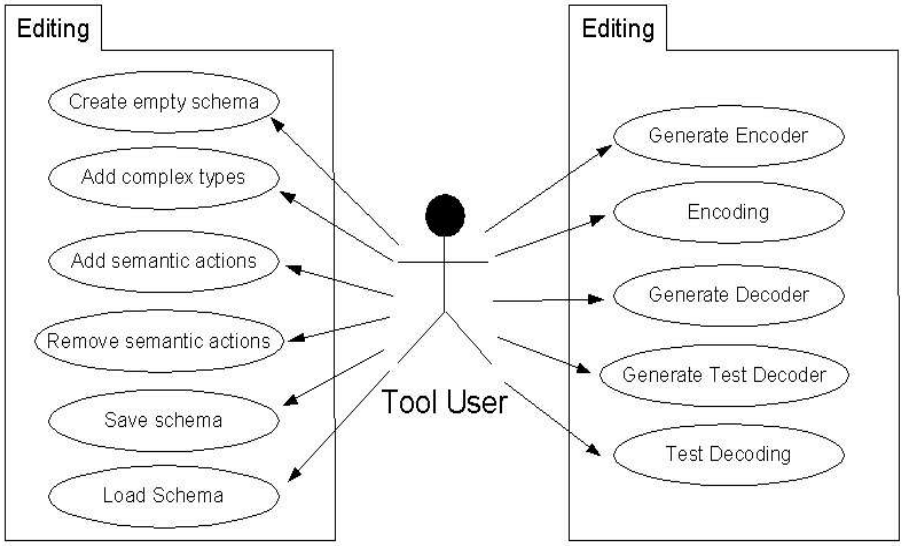
Situé "côté serveur", l'encodage des données est peu contraint, contrairement au décodage sur le téléphone portable. Le choix de la technologie se fait donc plus en fonction de la facilité de développement que de la performance pure. L'intérêt majeur d'un encodeur en XSLT est qu'il manipule de manière très naturelle les données XML en entrée, contrairement à Java qui s'appuie sur deux API concurrentes DOM et SAX, prévues pour la lecture de documents XML mais mal adaptées à leur transformation.

De plus, il est beaucoup plus simple de générer du code source XSLT, qui est un cas particulier de document XML, plutôt que du code source Java qui requiert des formatages textuels beaucoup plus complexes.

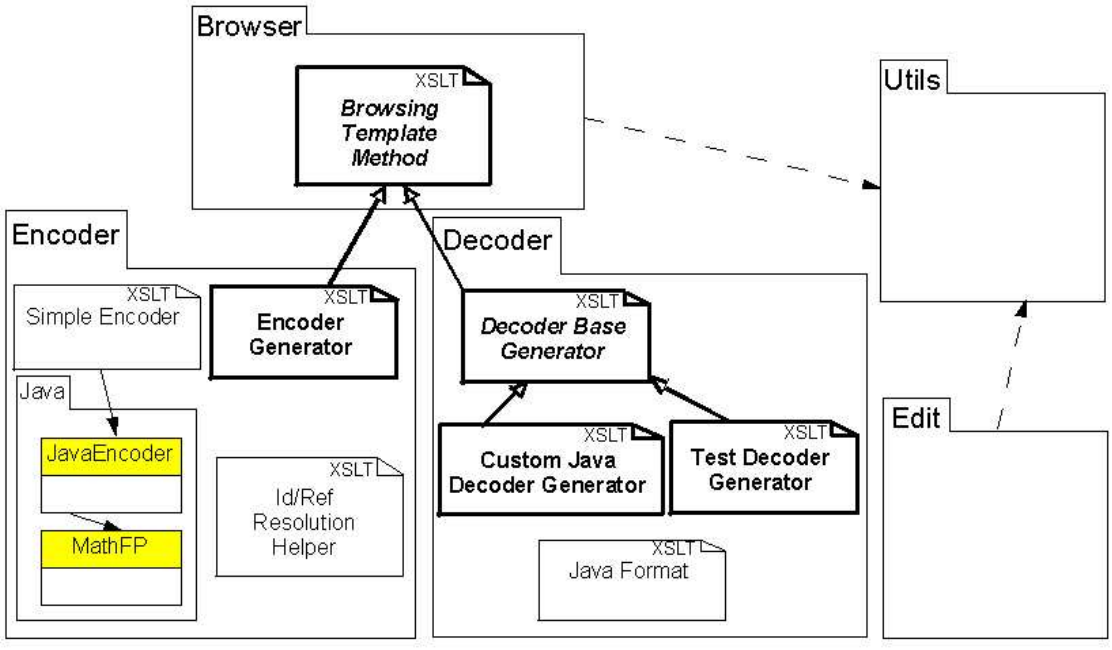
Afin de valider cette approche, j'ai mis en oeuvre rapidement un premier prototype de génération d'encodeur XSLT en XSLT sur des données d'exemple très simple. Ce premier prototype nous a permis de mettre en place une chaîne complète d'encodage/décodage, et nous a donc fourni, malgré sa simplicité, une base solide pour les développements ultérieurs.

ARCHITECTURE

Le diagramme de cas d'utilisations ci-dessous récapitule l'ensemble des tâches que l'utilisateur peut réaliser au moyen de l'outil Binary 4 Java. Les tâches de gauche correspondent aux différentes étapes de création du schéma XML, partiellement automatisées pour assister l'utilisateur. Celles de droite sont : générer l'encodeur et le décodeur et tester encodage et décodage d'un fichier de données utilisateur d'exemple.

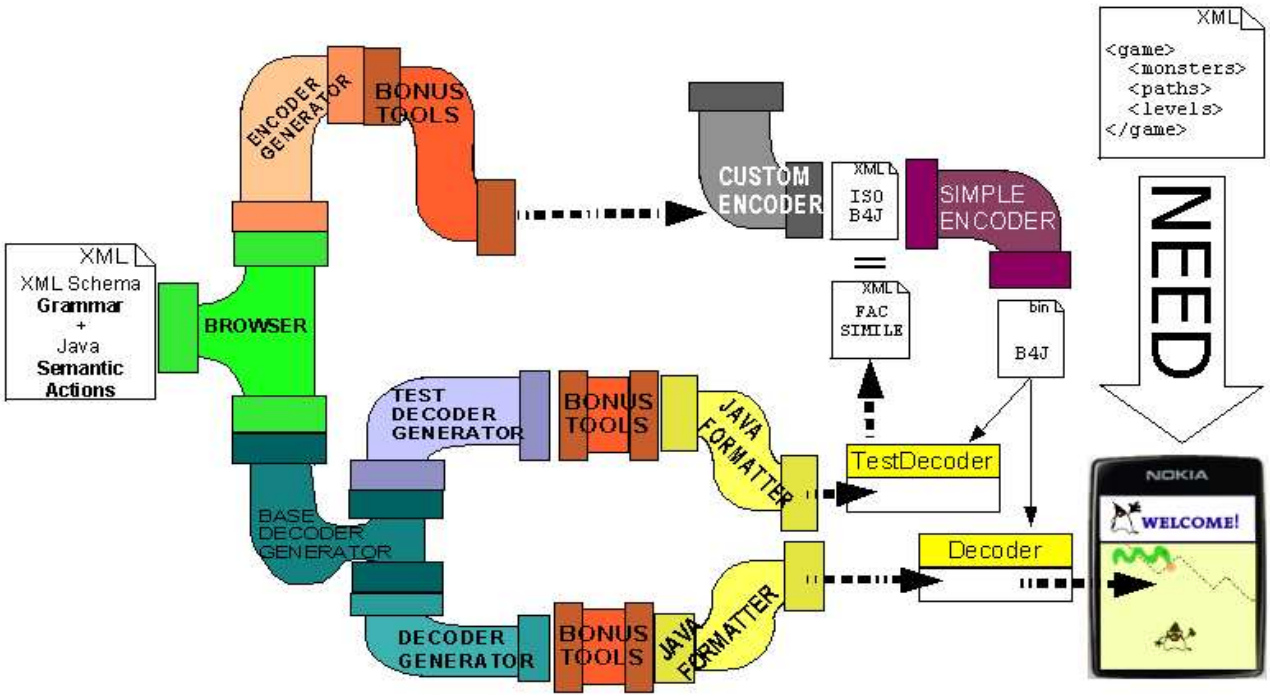


L'architecture de l'outil Binary 4 Java se base sur un principe simple : la factorisation des algorithmes communs à deux transformations dans une transformation plus abstraite. Les entités de base sont les paquetages **Encoder** (générateur d'encodeur) et **Decoder** (générateurs de décodeurs). Le paquetage **Utils** regroupe des librairies de fonctions de transformation utiles (indentation, formatage de texte, etc...). Le paquetage Edit regroupe les transformations qui génèrent les modèles de schémas XML et d'annotations que l'utilisateur va ensuite compléter.



Le paquetage **Browser** regroupe toute la logique de parcours de la grammaire XML Schéma. La transformation **Browsing Template Method** définit les étapes de l'algorithme et les actions à réaliser de manière abstraite : "processElement", "processAttribute" *(design pattern template method ou patron de méthode)*. Les transformations descendantes implémentent les actions concrètes correspondantes.

Le fait que le générateur d'encodeurs et le générateur de décodeurs suivent les mêmes étapes de traitement de la grammaire XML, définies au niveau du Browser, se reflète dans les encodeurs et décodeurs générés, qui utilisent eux aussi les mêmes étapes de traitement d'un fichier de données.



Le schéma ci-dessus représente, de manière imagée, l'emboîtement des différentes transformations qui constituent la chaîne complète des données de l'utilisateur jusqu'aux objets Java construits sur le téléphone portable. Trois maillons de cette chaîne sont générés automatiquement à partir de la grammaire annotée : **Custom Encoder** (Encodeur), **Decoder** (Décodeur), et **Test Decoder**, décodeur de test qui reconstitue les instructions d'encodage en entrée de **Simple Encoder** à partir d'un **fichier B4J.bin** de données encodées. Le décodeur de test est essentiellement utilisé à des fins de debuggage de l'outil.

Nous terminons ici le parallèle entre l'outil B4J et le processus MDA (Model Driven Architecture) :

- Les PIM (Plateform Independent Models) sont définis par l'utilisateur ; il s'agit des données XML et de la grammaire XML Schéma.

- Les PSM (Plateform Specific Models) correspondent du côté du générateur de décodeurs aux fichiers en entrée des transformations **Java Formatter** : elles génèrent le code source Java des décodeurs à partir de fichiers XML qui suivent la même structure qu'une classe Java. Il n'y a par contre pas vraiment de PSM de l'encodeur, sinon l'encodeur XSLT lui-même, que l'on peut considérer comme un PSM exécutable.

GROS PLAN SUR... XSLT ORIENTÉ OBJETS

XSLT est un langage de programmation qui ne s'écrit pas sous forme simplement textuelle, comme la grande majorité des langages de programmation, mais sous forme de balises XML.

Pour comparaison, voici un extrait de programme en XSLT et l'extrait équivalent dans le langage de programmation Java :

- en XSLT, pour exécuter un traitement particulier en fonction des différentes valeurs possibles de la variable $input, on écrira :

```
<xsl:choose>
  <xsl:when test="$input = 0">
    <xsl:call-template mode="handle0"/>
  </xsl:when>
  <xsl:when test="$input = 1">
    <xsl:call-template mode="handle1" />
  </xsl:when>
  <xsl:otherwise>
    <xsl:call-template mode="handleOther">
      <xsl:with-param name="otherValue" select="$input"/>
    </xsl:call-template>
  </xsl:otherwise >
</xsl:choose>
```

- en Java on écrira

```
switch(input) {
  case 0: {
    handle0();
    break;
  }
  case 1: {
    handle1();
    break;
  }
  default: {
    handleOther(input);
  }
}
```

XSLT définit un langage de **transformation de documents XML**. Sa première vocation est d'appliquer des feuilles des styles à des données XML provenant par exemple d'une base de données, pour permettre leur lecture sur différents types de terminaux : en HTML pour un navigateur Web, en SVG pour les graphiques ou en XSL-FO pour l'impression.

Toutefois, ses facultés ne se limitent pas aux feuilles de styles. XSLT est utilisé plus généralement pour transformer des données qui ne suivent pas la même logique, par exemple dans un dialogue entre deux applications.

Une transformation XSLT définit un ensemble de règles à appliquer au document d'entrée pour obtenir le document de sortie. Chaque règle s'applique sur un ou plusieurs éléments du document d'entrée et construit une partie du document de sortie.

Les données qui constituent les instructions d'un programme en XSLT sont elles-mêmes structurées et manipulables plus facilement par un autre programme. En effet, il existe de nombreuses librairies standard pour accéder, modifier et transformer les données qui suivent la syntaxe XML.

Ceci permet de générer plus facilement du code XSLT, ce qui était déterminant dans notre cas, et facilite par ailleurs le développement de processeurs XSLT, qui s'appuient sur des analyseurs XML standards développés indépendamment.

Ces analyseurs suivent principalement deux logiques :

- Une vision du document XML comme une arborescence que l'on peut parcourir à sa guise. Le standard DOM définit une API (Application Programming Interface, la définition d'un ensemble de fonctions dont plusieurs organisations vont pouvoir proposer des implémentations) d'accès au document sous la forme d'une arborescence de noeuds.

- Une vision plus réduite avec un curseur qui progresse toujours en avant, du début à la fin du document. Le modèle SAX définit une API événementielle : le document est parcouru en entier une seule fois, et des événements sont émis à toutes les étapes clefs de ce parcours.

De même, en XSLT, il existe deux approches principales pour aborder le problème de la transformation des données :

- une approche guidée par la logique du document d'entrée. Il s'agit de la technique la plus simple, qui consiste à traiter le document d'entrée dans l'ordre de ses éléments. C'est le mode par défaut des processeurs XSLT, qui appliquent les règles suivantes :

```
Pour traiter un élément
- si l'utilisateur a défini une règle pour cet élément, applique-la
- sinon traite chaque élément enfant dans l'ordre du document
et recopie tous les textes que tu trouves vers le document de sortie.
```

  Ce qui donne en XSLT :

```
<xsl:template match="*">
  <xsl:apply-templates />
</xsl:template>
<xsl:template match="text()">
  <xsl:value-of select="." />
</xsl:template>
```

  On utilise dans ce premier cas des règles (template) "match" qui s'appliquent à tous les noeuds qui respectent le motif défini par l'attribut match.

- une approche guidée par la logique du document de sortie. On définit dans ce cas des règles "name", qui ne se déclenchent pas directement, mais doivent être appelées explicitées. Il s'agit d'un mécanisme d'appel de fonction, utilisé classiquement dans la plupart des langages de programmation (voir l'exemple de la page précédente).

Dans le projet B4J, je me suis inspiré de l'approche objet pour définir une troisième voie. Du point de vue théorique, un objet est caractérisé par :

1. un état interne (ses données)
2. des opérations (ses méthodes)
3. une identité

Considérons une règle de transformation XSLT : elle s'applique sur des noeuds du document d'entrée qui sont d'un certain type, il s'agit de ses **données**. Elle représente une **opération** à effectuer sur ces données pour construire des données de sortie. Chaque élément d'information du document d'entrée est **identifié** par sa position dans ce document. Les briques de base des objets sont donc bien présentes.

Par ailleurs, la programmation orientée objets est caractérisée par trois concepts :

1. l'**encapsulation** : un objet est comme un iceberg, sa plus grande partie est immergée, (*son état interne est protégé, on ne peut le modifier directement ce qui garantit son intégrité et la possibilité de modifier sa représentation interne indépendamment des opérations proposées*), et on ne peut accéder qu'à la partie visible de l'iceberg (ses opérations).
2. **l'héritage** : un objet d'une classe fille hérite des caractéristiques des objets de la classe mère (état interne et opérations). Il peut redéfinir ces caractéristiques, par exemple pour compléter les traitements de base proposés par la classe mère.
3. **le polymorphisme** : un objet d'une classe fille peut toujours se "faire passer" pour un objet de la classe parente : il peut être utilisé partout à la place d'un objet de la classe parente.

Les données manipulées par une transformation XSLT sont **encapsulées** au sein du document d'entrée, qui n'est jamais modifié directement par la transformation ; ses données sont seulement recopiées pour construire un autre document en sortie.

Le mécanisme d'imports de XSLT correspond au concept d'**héritage**. Il ressemble à l'inclusion d'une feuille de style dans une autre, en y ajoutant la gestion des priorités entre les règles de la transformation principale et celles de la transformation importée ; les règles de la transformation principale s'appliquent prioritairement sur celles de la transformation importée, de la même manière que les méthodes d'une classe fille sont appelées prioritairement sur celles de sa classe mère. De la même manière qu'en Java, il est possible en XSLT d'appeler la méthode de base de la classe mère à partir de la méthode plus spécialisée de la classe fille en utilisant `<xsl:apply-imports />`. La seule limite par rapport à Java est qu'il est impossible de transmettre des arguments à cette méthode (limitation corrigée dans la deuxième version de XSLT, en cours d'élaboration au W3C).

Enfin, la transformation principale peut s'appliquer sur tous les fichiers de données sur lesquels s'appliquait la transformation de base, et les règles qu'elle définit **remplacent de manière transparente** les règles correspondantes de la transformation importée.

On peut donc programmer en XSLT en suivant une philosophie objet. Le tableau suivant récapitule les correspondances entre les concepts XSLT et les concepts objets :

Objet	XSLT
Objet	Transformation (correspond à un fichier, comme en Java)
Etat interne	Données du document d'entrée
Opération	Règle de transformation
Encapsulation	Transformation des donnés
Héritage et Polymorphisme	Import et Priorités

IV.BILAN ET CONCLUSION

BINARY 4 JAVA

Le développement de l'outil Binary 4 Java est terminé. C'est la société ExpWay qui va en poursuivre le développement ; il va ainsi compléter sa ligne de produit actuel, qui ne comportait pas jusqu'alors d'outil de génération de code.

B4J est actuellement disponible en deux versions : une version standard et une version étendue (power pack) qui propose en plus des fonctionnalités avancées. Ces deux versions sont le résultat d'une réflexion commerciale stratégique sur ce produit : afin de promouvoir cette technologie, la **version standard** devrait être publiée en Open Source avec une licence permettant l'utilisation gratuite du produit, sa modification et la redistribution du produit modifié (modèle de licence GPL, GNU General Public Licence, ou bien LGPL, GNU Library ou Lesser General Public Licence).

Ce type de produit aurait sans doute sa place parmi les projets de la fondation Apache (parmi lesquels le célèbre serveur Web Apache, l'outil Ant d'automatisation des tâches de construction de programmes, le serveur d'applications Tomcat, etc...). Il pourrait également être publié dans le cadre du futur groupe de travail du W3C sur la binarisation standardisée de données XML, dont ExpWay dirige la préparation actuellement.

La deuxième version, **power pack**, maintient une réelle plus-value dans l'offre d'ExpWay par rapport à la version standard : les fonctionnalités avancées permettent de gérer les données de plus gros projets, ajoutent la gestion de graphes, et permettent l'inclusion de fichiers de données externes à l'outil au sein des fichiers de données générés.

L'architecture de cet outil offre de nombreux avantagent sur la durée : le découpage entre version de base et version étendue est fait de manière à pouvoir faire évoluer les deux versions indépendamment l'une de l'autre. On pourrait ainsi intégrer à la version de base des contributions externes en provenance d'une future communauté d'utilisateurs, tout en ajoutant en interne chez ExpWay les fonctionnalités spécifiques nécessitées par des projets particuliers.

PastaGames

Le prochain jeu de PastaGames, Kouïz, est un jeu de gestion animalier. Le but du jeu est d'élever des créatures, de les nourrir pour les faire grandir, puis de les laisser se reproduire pour accroître la taille de l'élevage.

Le graphisme de chaque créature est déterminé par la combinaison aléatoire de sa couleur, sa taille, sa forme, la forme de ses yeux, de sa bouche, etc... Les enfants d'un couple héritent d'un mélange des caractéristiques de leurs parents.

Ces créatures ont également des traits de caractères différents. Certaines n'aiment que les aliments rouges, d'autres ont peur des petites créatures jaunes. Tous ces éléments de variété sont des paramètres à configurer pour équilibrer le jeu et le rendre intéressant.

Kouïz se base sur B4J pour l'encodage des caractéristiques des différents personnages et des objets du jeu :

- l'éleveur, l'avatar du joueur, possède une vitesse et une image
- certaines créatures peuvent être prédéfinies ; pour les créatures aléatoires il faut définir les plages de variation des valeurs de leurs caractéristiques (couleur, appétit, aggressivité, ...)
- les objets sont plus ou moins nourrissants ; ils peuvent être toxiques ou avoir d'autres propriétés. Ils ont aussi une fréquence d'apparition.

A l'image des tamagoshis, le joueur devrait peu à peu s'attacher à ces petites créatures indisciplinées. La diversité des créatures, toujours différentes, et la possibilité de suivre la destinée de ces créatures dont on découvre peu à peu le caractère en les voyant évoluer, sont deux aspects clefs pour l'intérêt du jeu, qui attisent la curiosité du joueur.

Dans une future version, les joueurs devraient pouvoir s'échanger leurs créatures, ou bien les envoyer rendre visite au téléphone portable d'un de leurs amis.

ExpWay

A l'issue de ce stage, Claude Seyrat et Cédric Thiénot m'ont proposé un poste d'Ingénieur Recherche & Développement chez ExpWay. Intégré à la division Mobile, je participerais au développement de tous types d'applications pour téléphones portables, et je poursuivrais le développement de l'outil Binary 4 Java.

J'ai accepté leur proposition avec plaisir, et je travaille chez ExpWay depuis le 1er décembre. Après avoir côtoyé leurs équipes pendant quatre mois dans leurs locaux, la transition fut assez facile. Mes premières tâches ont consisté à préparer la distribution d'une application de binarisation de graphismes vectoriels SVG pour les opérateurs de téléphonie mobiles, qui pourrait être déployée sur plusieurs millions de téléphones portables dans les prochains mois.

J'ai la chance de pouvoir collaborer à nouveau avec Fabien Delpiano à l'occasion d'un nouveau projet : le développement d'une architecture évolutive de guide de services intégrée à la future plate-forme Doja de NTT DoCoMo. Il s'agit d'un projet de grande envergure, qui s'appuie sur les projets auxquels j'ai participé au cours de mon stage, et pourrait conduire l'outil B4J à être intégré à une future suite d'outils de développement Doja.

Expway est la première société occidentale à participer à un Joint Development Projet (Projet développé conjointement) avec NTT. Expway entame actuellement une mutation, du rang de start-up à celui d'entreprise à vocation internationale, à laquelle je suis très heureux de pouvoir prendre part.

Je profite de l'occasion pour remercier le lecteur de m'avoir lu jusqu'au bout, et remercier Fabien Delpiano et tous les intervenants du DESS Jeux Vidéo et Média Interactifs pour nous avoir donné un aperçu passionnant du monde du jeu vidéo ; enfin, je réserve ces dernières lignes pour remercier tous les encadrants administratifs et la direction du DESS, en leur souhaitant toute la réussite possible pour la nouvelle école nationale du jeu vidéo.