La plasticité des I.H.M. ou comment développer un projet avec un code unique pour des interfaces multiples

Mathieu Humetz
Loïc Pénet

Dess MICE 2003

Université des Sciences et Technologies de Lille.

Remerciements.

Merci à tous ceux qui nous ont soutenus durant ce projet et qui ne sont pas cités.

Et plus particulièrement à :

José Rouillard : pour ses conseils eclairés et le suivi.

Jean Claude Tarby : pour le temps consacré.

Xavier LePallec : pour sa bonne humeur , sa réactivité et sa disponibilité.

Ludovic : pour sa sympathie.

Les épouses : pour nous avoir supportés.

La machine à café…

1 Table des matières.

Remerciements. 2
1 Table des matières. 3
2 Résumé. 6
2.1 Buts de PlasticML. 6
2.2 Plateforme requise pour l' implémentation. 6
2.3 PlasticML 1.1 un standard ouvert. 6
2.4 Outils relatifs à plasticML. 7
3 Introduction. 8
3.1 L' existant. 8
3.2 Le but de notre mission. 8
3.3 Notre challenge. 8
3.3.1 Concrétiser. 9
3.3.2 Transmettre. 9
3.4 Le contexte. 9
4 Etat de l' art. 10
4.1 Besoins de créations de méthodes et d' outils. 10
4.2 Historique de la recherche. 10
4.2.1 Projet AID. 10
4.2.2 Travaux de Dieterich. 10
4.3 Les langages de modélisation. 11
5 Notre vision. 13
5.1 Structure et contenus. 13
6 Le langage PlasticML. 14
6.1 Introduction. 14
6.2 Principe de fonctionnement. 14
6.3 Le fichier DB.xml. 14
6.4 Vue d' ensemple des éléments PlasticML. 15
6.4.1 L' élément <task> 18
6.4.2 L' élément <serveur> 18
6.4.3 L' élément <read> 19
6.4.4 L' élément <write> 20
6.4.5 L' élément <arg> 20
6.4.6 L' élément <table> 21
6.4.7 L' élément <target> 21

6.4.8 L' élément <request> 22
6.4.9 L' élément <sequence> 23
6.4.10 L' élément <head> 23
6.4.11 L' élément <meta> 24
6.4.12 L' élément <content/> 25
6.4.13 L' élément <message> 26
6.4.14 L' élément <entry> 27
6.4.15 L' élément <value> 29
6.4.16 L' élément <texte> 30
6.4.17 L' élément <media> 31
6.4.18 L' élément <information> 32
6.4.19 L' élément <control> 33
6.5 Intégration des éléments de contexte : 35
7 Les feuilles de transformation XSLT et les sorties. 36
7.1 XSLT vers html. 36
7.1.1 Les difficultés. 36
7.1.2 Notre sortie HTML : 38
7.2 XSLT vers wml. 39
7.2.1 Les difficultés : 39
7.2.2 Les solutions : 39
7.3 XSLT vers Voice XML 45
7.3.1 Les difficultés : 45
7.3.2 Les solutions : 45
7.3.3 Les options de tranformation retenues. 48
7.3.4 La transformation avec le code XSLT. 48
8 Les outils de développement. 49
8.1 Le PlasticML Tooolkit. 49
8.1.1 Principe et philosophie. 49
8.1.2 Génération automatique d' une interface à partir d' une DTD. 55
8.1.3 Création d'un nouveau document.. 57
8.1.4 Chargement d'un document existant. 58
8.1.5 Création des composants PlasticML. 58
8.1.6 Les possibilités d'exportations de documents 61
8.2 L' Application PlasticML Servlet. 63
8.2.1 Principe de fonctionnement. 63
8.2.2 Les données extérieures vers des SGBD multiples. 63
8.2.3 Mécanismes détaillés ou lecture et écriture dans les sources externes 67
9 Conclusions. 74
9.1 PlasticML et la mise en page. 74
9.2 La transmission du projet. 74
9.3 Développements futurs… 74

9.4 Les regrêts. 74
9.5 Les succès. 74
10 Sources de documentation. 75
11 Annexes 78
11.1 DTD de PlasticML 1.1 78
11.2 Feuille xslt vers HTML. 81
11.3 Feuille xslt vers WML. 87
11.4 Feuille xslt vers VoiceXML. 94
12 Notes 103

2 Résumé.

PlasticML 1.1 est un langage et une méthode de description d' interfaces indépendant des sorties possibles. Il a pour axe de recherche la tâche effectuée par un utilisateur dans le cadre d' une session en tenant compte d' éléments de contexte d' interaction, des contrôles à effectuer côté client et serveur, linguistique, ….

PlasticML 1.1 est un langage **basé sur XML** développé au sein de **l' université de Lille 1** par l' équipe de **José Rouillard, Maître de conférences** : cette version augmente la portée de la version réalisée en 2002 par la même équipe. Ce projet en cours de développement, vise dans un premier temps la production d' interfaces cohérentes au sens de la réalisation de la tâche. Les données informationnelles séparées des descriptions de structures de réalisation des tâches seront traitées au niveau de la mise en forme graphique pour les «sorties visuelles» dans un second temps. A cet effet, les contenus d' informations sont séparés des contenus de srtucture afin de les traiter différemment.

PlasticML 1.1 est donc **un métalangage de représentation abstraite d' interfaces** destiné à être « mappé ou parsé » vers des langages existants comme HTML, XHTML, WML, VoiceXML… La combinaison des éléments de PlasticML 1.1 offre une diversité de possibilités qui alliés à la puissance de XSLT **permet de générer pour un code plusieurs sorties** (devices).

2.1 Buts de PlasticML.

Spécifier une fois pour toute et obtenir des sorties diverses.

2.2 Plateforme requise pour l' implémentation.

Toute plateforme supportant « java »

2.3 PlasticML 1.1 un futur standard ouvert ?

Dans cette phase de réflexion ce langage n' est pas ouvert librement pour une implémentation : celui-ci fait l' obet d' une **soumission en cours par le laboratoire Trigone** de l' Université de Lille1.

2.4 Outils relatifs à plasticML.

-PlasticML ToolKit est un éditeur de code plasticML.

-PlasticML Servlet est une servlet « java » pour visualiser le rendu de feuilles PlasticML. Notez bien que ce servlet n'est en fait qu'un implémentation temporaire de ce que pourrait êtte un serveur supportant le PlasticML.

3 Introduction.

3.1 L' existant.

Dans le cadre d'étude de la plasticité des IHM, **le laboratoire Trigone** a développé **un langage de description abstraite** d'interaction Homme-Machine basé sur XML : PlasticML. Ce langage permet de décrire des éléments d'entrées et de sorties d'interfaces, à un haut niveau, c'est-à-dire sans faire référence au périphérique qui sera utilisé lors de l'interaction. Grâce à des transformations avec XSLT, les documents en PlasticML sont traduits automatiquement vers des langages idoines (HTML pour le Web, WML pour le WAP, VoiceXML pour le téléphone ...).

Un modeleur graphique (similaire à FrontPage ou Dreamweaver pour HTML) permet de créer des documents en PlasticML, grâce à des manipulations directes (clavier/souris), sans avoir à écrire le code correspondant.

3.2 Le but de notre mission.

Étudier les différents langages existants pour décrire des interactions de manière abstraite (Xforms, XIML, UIML, Sun.ML, CC/PP, etc.) et en synthétiser les éléments communs (et aussi ceux qui ne le sont pas, mais paraissent intéressants à intégrer dans le projet).

Étudier la DTD existante et **proposer des améliorations** (ajouter des balises pour les entrées et les sorties de données) en s'inspirant de langages similaires (Xforms notamment) ,

Recréer un modeleur graphique (PlasticML Toolkit) en JAVA, pour ajouter les éléments proposés dans la 2 ème étape. **Améliorer l'ergonomie du logiciel**, pour qu'il soit utilisable par des non spécialistes.

Proposer une application serveur chargée de transformer des documents originaux en PlasticML à la volée vers les devices appropriés. Cette application doit aussi mettre en avant le caractère dynamique du langage dans le cas d'interrogation de source de données externes.

3.3 Notre challenge.

Nous avons choisi de concert ce projet pour sa **dimension et ses potentialités** mais aussi pour ses enjeux.

3.3.1 Concrétiser.

Pour augmenter la portée de la mission, nous proposons une application serveur chargée de transformer des documents originaux en PlasticML à la volée vers les sorties appropriées. Cette application devra gérer les données externes stockées dans des systèmes de bases de données multiples et différentes.

3.3.2 Transmettre.

Nous réalisons un document de travail afin que des équipes futures puissent comprendre et reprendre facilement ce projet.

3.4 Le contexte.

Le **besoin d'un accès ubiquitaire** à l'information (au bureau, chez soi, dans le train, etc.) se fait sentir de plus en plus avec la venue de nouveaux matériels et le succès des ordinateurs de poche ou des téléphones portables, les avancées des technologies réseaux et du sans fil, offrent **de nouvelles voies de recherche** pour la communauté en I.H.M. 1 . Ces nouvelles formes d'interaction se heurtent à de nombreuses difficultés parmi lesquelles, le développement et le maintient des interfaces homme-machine sur chaque système.

Pour être utilisable, l'interface utilisateur **doit être adaptée** aux dispositifs d' interaction, par exemple à la taille de l'écran ou à la présence ou non d'un clavier ou à d' autres contraintes comme les profils ou les préférences.

4 Etat de l' art.

4.1 Besoins de créations de méthodes et d' outils.

Dans un souci d' **économie financière,** lors de la création d' un programme ou d' un service orienté réseaux, une nécessité s'impose: **optimiser la production** des interfaces:

- réduire les coûts,
- assurer une réutilisabilité du code,
- faciliter la maintenance et la cohérence des objets,
- capitaliser le développement initial.

4.2 Historique de la recherche.

La recherche autour du thème de la plasticité **émerge doucement** et les documents, articles ou travaux sont encore rares sur le réseau. Pourtant **des pools de réflexion émergent** en Belgique autour du projet Mozart, à Grenoble et bien entendu à Lille dans le laboratoire Trigone de l' Université de Lille 1.

Une IHM est évaluée comme:

- **adaptative**: elle réagit à une situation prévue par le concepteur**,**
- **autorégulatrice**: elle procède par le mode *essais , erreurs , apprentissage* ,
- **auto médiatrice:** elle anticipe la réaction de l' utilisateur,
- **auto modificatrice:** elle a le pouvoir de "muter".

Nous vous indiquons par ordre chronologique les différentes étapes de l' état de la recherche:

4.2.1 Projet AID.

Ce projet a pour objet de résoudre le cas de systèmes adaptables à de multiples utilisateurs. Cette **adaptation est automatique** pour l' aide, les messages d' erreurs, les langages de commandes, les styles de dialogues. Il n' y a pas d' adaptation au contexte malgré un début de méthodologie.

4.2.2 Travaux de Dieterich.

Dieterich établit une notion **de déclencheurs** de modifications des I.H.M. qui peuvent être les concepteurs , les administrateurs , les experts locaux, l' utilisateur, le sys-

tème. Le processus de l' adaptativité découle d' actions comme: l' Initiative, la suggestion, la décision, l' exécution. Une question se pose: **quand faut-il adapter?** Est-ce le concepteur? nous parlerons alors d' une **interface précalculée**. Est-ce pendant l' exécution ou entre les utilisations par l' administrateur?

Il existe plusieurs niveaux d' adaptation:

- **Syntaxique**: Goûts et expertise de l' utilisateur.
- **Heuristique**: optimisation par le système du meilleur choix.
- **Sémantique**: Aide pour faire l' action ou la tâche.

Que faut-il adapter? Les présentations , la navigation, les contenus...
Quels sont les facteurs possibles d' influence sur
l' Adaptativité d'une IHM?

- Les profil utilisateur,
- l' environnement contexte (bureau, gare),
- Le modèle du domaine et les fonctions « métier »,
- Les modèles du dialogue parlé, écrit ou autres...,
- Le modèle de l' application,
- Les modèles d' ergonomie, les règles et euristique d' ergonomie.
- Les modèles de présentation,
- Les modèles de comportement,
- Les modèle de plates-formes.

4.3 Les langages de modélisation.

Afin d' étudier l' état de l' art dans ce domaine, nous nous sommes penchés sur les langages de modélisation disponibles sur le marché.

UIML3:Ce langage propose une spécification par cible: il a été amélioré par UIML2 qui est encore en travaux et propose des interfaces calculées.

AUIML :La spécification est Indépendante de la cible. Il possède un gestionnaire de navigation qui construit la structure de l' interface et détermine les choix de navigation. Il est peu diffusé et pas utilisé. Les interfaces sont précalculées.

Qtx: :Ce langage propose une séparation de la présentation et des données et une présentation en fonction de la surface d' affichage. Le concepteur définit les vues lui-même.

XIML :XIML est une méta syntaxe qui décrit les tâches, attributs de propriétés de composants. Il gère des préférences utilisateur; son outil de développement est "MOBI-D".

Xforms:Le langage est destiné à fournir des document en xhtml. Il propose de séparer les data, la forme et la structure. Il est assez abscons et peu de documentation existe. Il est très peu utilisé car les navigateurs qui ne supportent pas encore tous le xhtml.

SunML:Le langage SUNML est un langage balisé inspiré du langage UIML. Dans sa version actuelle, le langage comporte 19 balises. Il n' est pas distribué.

CC/PP :Cette application de XML dont le consortium publie aujourd'hui un guide d'implémentations permet de décrire les caractéristiques logicielles et matérielles d'un terminal (Web ou WAP, capacité machine, etc.). Mais également les informations relatives au profil de son utilisateur.

5 Notre vision.

Le langage utilisé doit s**' abstraire des séquences** de présentations possibles pour s' orienter sur la description formelle d' interfaces au niveau d'un ensemble de tâches réalisées au sein d' une session . Ce langage doit être suffisamment abstrait pour anticiper **la venue de nouvelles cibles** pour ne plus en modifier le code ultérieurement. Il représentera une vue abstraite de l' interface utilisateur. Les termes employés **ne devront pas** faire référence ou même **s' identifier à une cible** afin de ne pas entraîner de confusions. L' élément *<page>* par exemple nous semble inapproprié dans la mesure ou il réfère sémantiquement à un espace et un surface qui n' a aucun sens en VoiceXML.

5.1 Structure et contenus.

Dans la pratique, **un contenu informatif peut se situer à n' importe quel endroit** ou moment dans la présentation d' une séquence sur une cible donnée. Ces contenus devront dans le cas d' une interface écran, avoir une présentation dotée d' un parti pris graphique répondant à des besoins **de layout** et de mise en page determinés par les concepteurs.
Il nous apparaît comme obligatoire **de tenir compte à l' origine du projet** de ce fait, et de séparer dans le langage la structure de l' information. L' information doit pour une flexibilité maximale de développement se situer à l' extérieur du document XML de structure de <message>.

6 Le langage PlasticML.

6.1 Introduction.

A la reprise de la mission la DTD contiend **cinq éléments** (page, output, form, field, value). Nons portons le nombre d' éléments à quinze afin de traiter la totalité des possibilités offertes sur les langages de sortie.

Les tests de transformation réalisés avec les feuilles XSL que nous avons conçues nous ont amenés à **modifier les éléments PlasticML afin de les améliorer** et de les affiner. Le langage n' est certainement pas fixé ni même abouti à la rédaction de ce mémoire car des aspects n' ont pas été traités comme la mise en page. Nos efforts se sont concentrés sur la réalisation d' une tâche.
La réalisation de feuilles XSL pour obtenir les différentes sorties nous a amenés à modifier les éléments PlasticML.

6.2 Principe de fonctionnement.

Sur le schéma ci-dessous vous pouvez voir le principe général de transformation d'un document rédigé en PlasticML vers les sorties cibles. Pour réaliser cette transformation, le document PlasticML est trans formé grâce à une feuille XSLT correspondant à la sortie demandée .

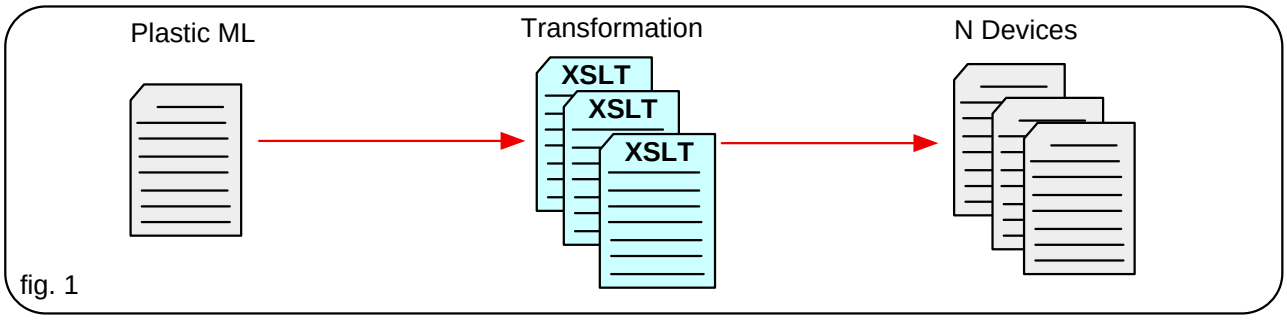


fig. 1

6.3 Le fichier DB.xml.

Pour récupérer des informations externes , un fichier de description de sources de données est nécessaire. Nous expliquons au chapitre 6.3.2 le fonctionnement du serveur d' application avec ce fichier. Sa structure d' information permet de définir plusieurs sources de données : elle se présente comme ci-dessous :

```
<db>
<database id="message" pilote="Xindice">
<url>Url de la source</url>
<user> Pasword</user>
<login> Nom de l' utilisateur </login>
</database>
</db>
```

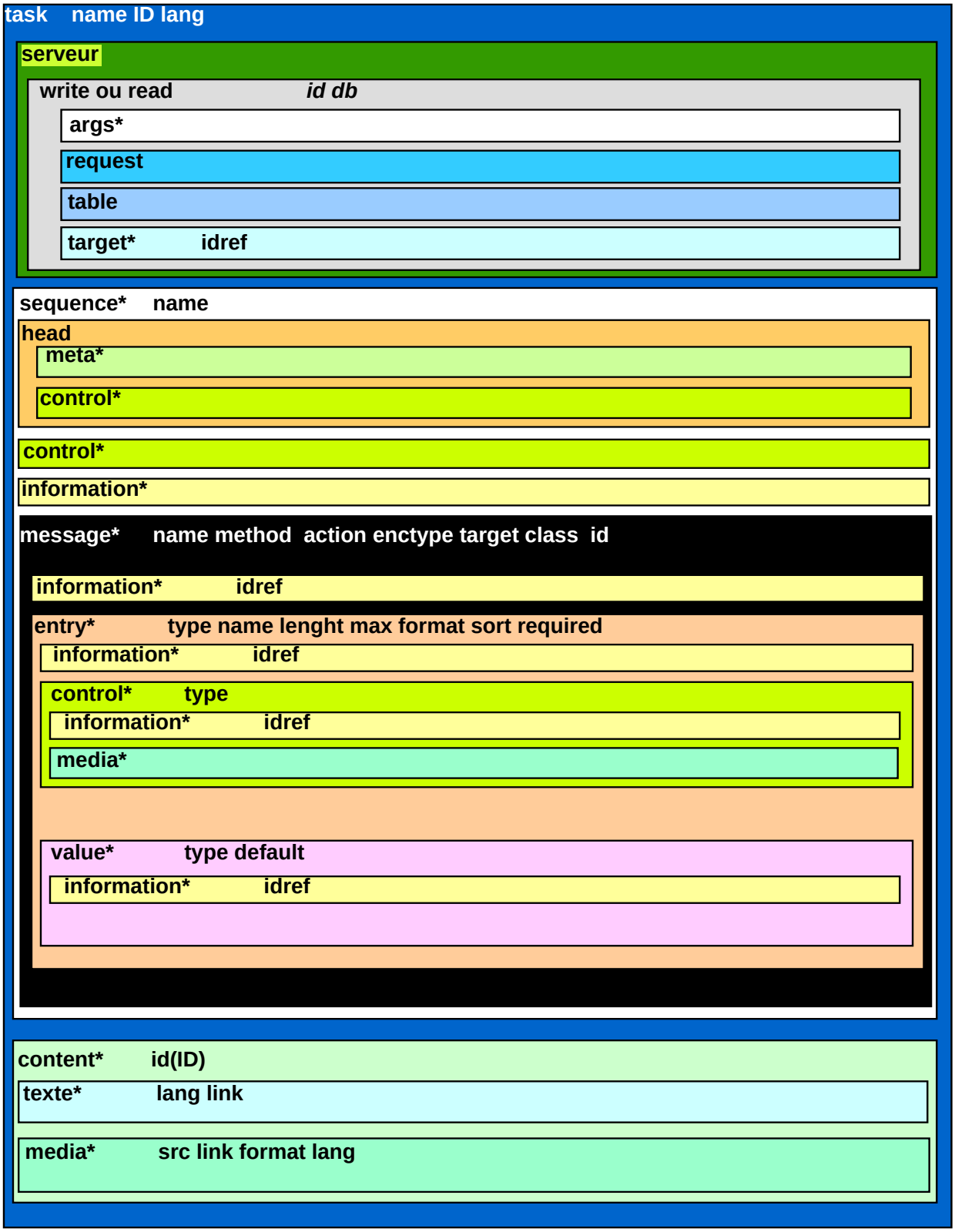
6.4 Vue d' ensemble des éléments PlasticML.

Le tableau ci-dessous présente les principaux éléments du langage :

Elements :	*Description :*
<task/>	La Tâche que doit effectuer l' utilisateur.
<serveur/>	Les traitements opérés par le serveur sur les post ou get arguments.
<read/>	Action de lecture dans une source de données externes (database ou fichiers) définie dans le fichier de description de bases de données db.xml.
<write/>	Action d' écriture dans une source de données externes (database ou fichiers) définie dans le fichier de description de bases de données db.xml.
<arg/>	Valeur des noms de champs pour redéposer l' information dans le fichier Plastic-ML ou écrire dans une base de donnée.
<table/>	Nom de table d ' une source externe.
<target/>	L' identifiant des éléments où doivent être redéposés les données.
<request/>	Chaîne de caractères (SQL ou XPATH).
<sequence/>	Une partie de la tâche qu' il est possible de découper.
<head/>	Délimite la partie de la sortie non appréhendée , vue ou entendue par l' utilisateur.
<meta/>	Contenus d' entête .
<content/>	Pointeur sur les éléments d' information.
<message/>	Indique un ensemble d' entrées utilisateurs et pointe sur un URI. L' élément <entry type= 'submit'> declenchera la sortie d' un message contenant les valeurs des <entry> renseignées par l' utilisateur.
<entry/>	Les champs que l' utilisateur informe pour envoyer le message.
<value/>	Valeur par défaut des entrées.

<texte/>	Informations textuelles dans le code destinées à être entendues ou vues.
<media/>	Tout contenu exterieur au code (images, CSS, scripts, grammaire, « include »…)
<information/>	Contenus hors structure de la tâche destinés à être mis en page.
<control/>	Vérification des entrées utilisateur côté client ; par exemple une grammaire un script. Cet élément contient <media> si le contenu de contrôle est externe.

Représentation visuelle des éléments du langage:



6.4.1 L' élément <task>

Description: l' élément task est la racine de la feuille XML. Il correspond à la tâche que doit effectuer l' utilisateur.

Parents: aucun (élément racine).

Attributs:

name	Nom de la tâche.	requis
ID	Identifiant.	facultatif
lang	Langue par défaut du document	Défaut :fr

Remarque : le typage des langues utilisées s' effectue avec les items de la norme « ISO ».

Enfants: sequence* content* serveur.

6.4.2 L' élément <serveur>

Description: Description des opérations à effectuer avant de « parser » le document dans la sortie choisie. <serveur> est un élément de regroupement. Cet élément est indispensable pour créer un document PlasticML.

Parents: task.

Attributs: aucun.

Enfants: write*, read.

Exemple :

```
<serveur>
   <write id="w1" db="addressbook" >
//une action d' écriture  dans une source.
</write>
</serveur>
```

6.4.3 L' élément <read>

Description:.Indique une lecture dans un fichier ou une source externe.

Parents: serveur.

Attributs:

id	ID identifiant	requis
db	La référence de l' élément < database> dans le fichier de description de bases de données db.xml	requis

Enfants: arg+ (au moins 1) , table , target.

Exemple :

```
<read id="w1" db="message" >
<request>Insert into person (nom, mail) values(« ***», « ***») </request>
<arg>mail</arg><arg>nom</arg>
<table> person</table>
</read>
```

L' application serveur :

1. cherche l' élément 'message' dans le fichier db.xml ,
2. cherche la présence d' arguments post ou get de nom='mail' et 'nom' ,
3. lit dans la source les données de la requête ,
4. remplace dans les éléments <target>.

6.4.4 L' élément <write>

Description:.Indique une écriture dans un fichier ou une source externe.

Parents: serveur.

Attributs:

id	ID identifiant	requis
db	La référence de l' élément < database> dans le fichier de description de bases de données db.xml	requis

Enfants: arg+ (au moins 1) , table.

Exemple :

<write id="w1" db="addressbook" >
<request>Insert into person (nom, mail) values(« ***», « ***») </request>
<arg>mail</arg><arg>nom</arg>
<table> person</table>
</write>

L' application serveur ;

1. *cherche l' élément '*addressbook*' dans le fichier db.xml ,*
2. *cherche la présence d' arguments post ou get de nom='mail' ,*
3. *met la valeur de 'mail' dans la source externe.*

6.4.5 L' élément <arg>

Description : nom des paramètres rentrants en post ou get dont le serveur d' application doit chercher les valeurs à écrire ou lire dans un fichier ou une source externe.

Parents: write.

Attributs: aucun.

Enfants: non.

Exemple : voir <write>.

6.4.6 L' élément <table>

Description : nom de la table de la source de données concernée par une action read ou write.

Parents: write.

Attributs: aucun.

Enfants: non.

Exemple : voir <request>.

6.4.7 L' élément <target>

Description: L' identifiant des éléments où doivent être redéposés les données.

Parents: read, write.

Attributs:

idref	ID identifiant (IDREF)	requis

Enfants: non

Exemple : ci-dessous , l' application redéposera les valeurs trouvées dans la balise : <content id="bienvenue" type="T1"><texte lang="fr">**ICI**</texte></content>

```
<read>
        /…..   …./
        <target idref="T1" ></target>
        <target idref="T2" ></target>
</ read >
```

6.4.8 L' élément <request>

Description: La chaîne de caractères utilisée pour effectuer une action de lecture ou d' écriture dans une source externe.

Parents: read, write.

Attributs: non.

Enfants: non

Exemple : ci-dessous , l' application remplacera les « *** » par les valeurs des arguments mail et nom

```
<read>
        /…..   …./
        <arg>mail</arg><arg>nom</arg>
        <request>
        Insert into person (mail, nom) values(« ***», »***»)
        </request>
</ read >

http://www.monserveur.com?mail=monnom@serv.com&nom=dupont
donnera la requête
```

Insert into person (mail, nom) values(« monnom@serv.com», « dupont»)

6.4.9 L' élément <sequence>

Description: une tâche peut être redivisée en plusieurs séquences: pour l' instant; il n' y a pas d' automatisme de détection pour obtenir des séquences en fonction par exemple des en-tête http ou de toute forme de détection d' un contexte donné. L' élément <sequence> indique que la tâche peut être si besoin découpée à cet endroit. *Cet élément n' est pas utilisé à ce stade du projet.*

Parents: task.

Attributs:

name	Nom de la séquence.	requis

Enfants: head? control* information* message*.

6.4.10 L' élément <head>

Description: délimite la partie de la sortie non appréendée par l' utilisateur.

Parents: sequence.

Attributs: aucun.

Enfants:
meta* : Balises d' entêtes.
control* : scripts.

Exemple :

deviendra après transformation
HTML :<*head*></*head*>
WML *: (non défini).*
VOICEWXML *: (non défini).*

6.4.11 L' élément <meta>

Description: Contenus d' entête (non finalisé).

Attributs: aucun.

Enfants: non.

Exemple :

Deviendrait après transformation :
HTML :
<meta name="description" content="…….">
<meta name="keywords" content=" …">
< meta name ="Revisit-After" CONTENT="…">
<meta name="robots" content="index,follow">
<meta name="Publication-Date" content="14 septembre 2001">
WML *: (non défini).*
VOICEWXML *: (non défini).*

Remarque: élément à étudier et compléter.

6.4.12 L' élément <content/>

Description: pointe sur tous les contenus d' information : l' élément est vide.

Parents: task.

Attributs:

id	(ID) Identifiant de l' élément information correspondant. Prend le nom du paramètre post ou get à recupérer dans le cas d' un type 'in'.	requis
type	*Internal* : dans le code source *Out* : remplacement de valeur depuis un post ou get argument par une valeur venant d' unne source externe. *In* : remplacement de valeur depuis un post ou get argument par la valeur du post argument.	Défaut : 'internal'
db	La référence de l' élément < database> dans le fichier de description de bases de données db.xml.	facultatif

Enfants:
media* : images ou autres.
texte* :

Exemple :

<content id="mail" type="in"/>
Supposons un paramètre post ou get 'mail=monnom@monadresse.com'

La valeur deviendra avant transformation :
<content id="mail" type="in">
<texte>monnom@monadresse.com</texte>
</content>

6.4.13 L' élément <message>

Description: Un message est un lien vers le serveur avec en général un passage de paramètres entrés par l' utilisateur.

Parents: sequence.

Attributs:

name	Nom de l' élément.	requis
method	*post* (par défaut) *get* : mode de transmission des valeurs. *put* : mode de transmission des valeurs.	post
action	URL de destination.	requis
enctype	Encryptage des données.	facultatif
target	Une autre fenêtre si la sortie le permet.	facultatif
class	Groupage des eléments par catégorie.	facultatif
id	Identifiant unique.	facultatif

Enfants:
information* :
entry* .

Exemple :

```
<message action="http://www.xx.com/monfichierhtml" name="le_formulaire_test"/>

deviendra après transformation :
HTML :
<form name="le_formulaire_test" action=" http://www.xx.com/monfichierhtml"
method="post" target="" enctype="">

WML :
<do type="accept" label="10champs_submit">
<go href=" http://www.xx.com/monfichierhtml " method="post" sendreferer="true" class=""
id="">
<postfield name="1champs_cache" value="$(1champs_cache)"/>
</go>
</do>
```

VOICEWXML :
<block>
<submit next=" http://www.xx.com/monfichierhtml *" namelist="…"/>*
</block>

6.4.14 L' élément <entry>

Description: champs d' entrée utilisateur.

Parents: message.

Attributs:

type	*choice* : il y a plusieurs propositions. *reset* : remise à « null » de champs. *submit* : acceptation et validation pour envoi. *secret* : caché à la saisie des personnes. *file* : pointer un fichier externe. *text* : chaîne de caractères. *hidden* : caché.	facultatif
name	Nom du champs. (string).	requis
lenth	Longueur physique du champs. (int).	facultatif
max	Nombre de caractères autorisés (int).	facultatif
format	*string* : chaîne de caractères *booleen* : booléen. *number* : Int.	facultatif
sort	*checkbox* : possibilité de sélectionner plusieurs choix *radio* : choix exclusif. *multiple* : possibilité de sélectionner plusieurs choix. *area* : multiligne.	facultatif
required	*true :*obligatoire *false :* détermine si le champs doit être rempli par l' utilisateur	true

Exemple de sortie en fonction d' un choix d' attributs :

Attribut :	***HTML :***	***WML***	***VOICEWXML***
type="hidden"	caché	Postfield caché	Variable déclarée
type="text"	texte	texte	Question sans choix
type="text" sort="area"	texte area	texte	Question sans choix
type="secret"	password	password	
type="choice"	Liste à sélection unique	Liste à sélection unique	Liste à sélection unique
type="choice" sort=" multiple"	Liste à sélection multiple	Liste à sélection multiples	Liste à selection unique et possibilité de revenir sur le champs
type="choice" sort=" checkbox"	Choix checkbox	Liste à sélection multiples	Liste à sélection unique et possibilité de revenir sur le champs
type="choice" sort=" radio"	Choix radio exclusif	Liste à sélection unique	Liste à sélection unique
type="file"	file	*Pas prévu*	*Pas prévu*
type="submit"	submit	submit	submit
type="reset"	reset	reset	*Pas prévu pour l' instant*

Enfants: value*, information*, control*

Exemple :

<entry name="choix_unique" sort="choice" type="choice">

deviendra après transformation :

HTML :
<select name="choix_unique" size="1">
<option value="default_fr">…</option>
<option value="valeur_gb">…</option>
</select>

WML :

```
<select name="5choix_unique" value="valeur_gb" title="" multiple="true" xml:lang="" id="" class="">
<option value="default_fr">…</option>
<option value="valeur_gb">…</option>
</select>

VOICEWXML :
<field name="choix_unique">
<prompt>…  </prompt>
<option dtmf="1" value="valeur_gb">…</option>
<option dtmf="2" value="ddd">…</option>
</field>
```

6.4.15 L' élément <value>

Description: contient les valeurs des <entry>.

Parents: entry.

Attributs:

id	identificateur	requis
type	*Internal :* valeur dans le code. *Out* : la valeur doit être prise dans une source externe avant la transformation. *In* : la valeur du paramètre post ou get de même nom que l' <entry> parente doit être redéposé dans la valeur.	Défaut : 'inter-nal'
default	*yes* : valeur par défaut si le <value> n' est pas renseigné. *no* :	Défaut : 'no'
db	La référence de l' élément < database> dans le fichier de description de bases de données db.xml	facultatif

Enfants: information*.

Remarque :

Une <value default="yes"> sera cochée par défaut et lue en premier dans le cas du **VOICEXML**.
Pour les type="in" et type="out" voir la section sur « les données extérieures ».

La figure 2 montre le fonctionnement des attributs 'in' ou 'out' pour les éléments <content> et <value>.

6.4.16 L' élément <texte>

Description: chaîne de caractères destinés à être affichée ou parlée.

Parents: content.

Attributs:

link	CDATA : un URL ou un URI	facultatif
lang	Choix dans la liste ISO	facultatif

Enfants: non.

Exemple :

<texte lang="en">KINDNESS</texte>
deviendra après transformation :

HTML *:*
* * KINDNESS* *

WML *:*
* * KINDNESS* *

VOICEWXML *:*
<form>
<block> KINDNESS *<goto next="#formsuivant"/>*
</block>
</form>

<texte lang="en" link="http://wwwmonserveur.com/fichier.htm" >
KINDNESS
</texte>
deviendra après transformation :

HTML *:*
** KINDNESS **

WML *:*
<do type="accept">
KINDNESS
<go href=" http://wwwmonserveur.com/fichier.htm*"/>*
</do>

VOICEWXML *:*
<form>
<block> KINDNESS *<goto next=* « http://wwwmonserveur.com/fichier.htm*"/>*
</block>
</form>

6.4.17 L' élément <media>

Description: Description générique d' un média externe à la sortie , nécessaire à sa représentation : par exemple une CSS , un fichier externe ou une image , un son, un script ou une grammaire…..

Parents: control, content.

Attributs:

lang	Choix dans la liste « ISO »	facultatif
src	CDATA : un URL ou un URI	requis
link	CDATA : un URL ou un URI : decrit un lien vers un autre fichier	facultatif
format	NOTATION (jpeg | png)	facultatif

Enfants: non.

notation:

jpeg	SYSTEM "image/jpeg"	facultatif
png	SYSTEM "image/png">	facultatif
gif	SYSTEM "image/gif">	facultatif

Exemple :

<control type="grammar">
<media lang="fr" format="application/srgs+xml" src="ma_grammaire.grxml"/>
</control>
deviendra après transformation :

VOICEWXML :
<grammar type="application/srgs+xml" src="ma_grammaire.grxml"/>

<control type="script"><media src lang=" JavaScript " ="monscript.js"/></control>
deviendra après transformation :

HTML :
<SCRIPT LANGAGE ="JavaScript" SRC="monscript.js"></SCRIPT>

6.4.18 L' élément <information>

Description : une information peut être partout.Cet élément est un pointeur sur un élément <content>

Parents: value, entry, message, sequence.

Attributs:

idref	Référence unique chaîne de caractères de l' élément <content>	requis

Enfants: non.

Exemple :

<information idref="bienvenue"/>

renvoie sur l' élément :

<content id="bienvenue" type="out">
<texte lang="fr"> momo Plastic m l de la promotion 2003 </texte>
</content>

6.4.19 L' élément <control>

Description: Représentation des vérifications côté client.

Parents: entry, head, sequence.

Attributs:

type	*filled* : action à réaliser si une <entry> est renseignée. *help* : action pour aider l' utliisateur. *nomatch* : action à réaliser si l' entrée est mal renseignée. *noinput* : action à réaliser si l' entrée n' est pas renseignée. *script* : code côté client pour contrôler…	facultatif

Enfants: information*, media*.

Remarque: lorsque le type est « *help »,* il sera utile de trouver une transformation par exemple des ballons d' aide pour « HTML » et à définir pour la sortie « WML ».

Exemple :

```
<control type="grammar">
<media lang="fr" format="application/srgs+xml"  src="ma_grammaire.grxml"/>
</control>
<control type="nomatch">
<information idref="nm1"/>
</control>
 <control type="help">
<information idref="nm2"/>
</control>
<control type="noinput">
<information idref="nm3"/>
</control>
```

deviendra par exemple après transformation :

VOICEWXML :

```
<gramar type="application/srgs+xml" src="ma_grammaire.grxml"/>
<catch event="nomatch">
Mauvaise reponse recommencez
</catch>
<help>
Pour vous aider
</help>
<catch event="noinput">
Je n ai pas entendu votre reponse
</catch>
```

6.5 Intégration des éléments de contexte :

La langue utilisée doit être récupérée dans les entêtes http. PlasticML est prévu pour implémenter toutes les langues.

```
<content id="entry7">
   <texte lang="gb">BANANA</texte>
   <texte lang="fr">BANANE</texte>
….
<texte lang="codeISO">…….</texte>

</content>
```

Une langue par défaut est indiquée en haut du fichier dans l' élément <task lang="fr">. si le « parser » ne trouve pas de contenu en anglais dans ce cas il utilisera les éléments du type de langue par défaut.

7 Les feuilles de transformation XSLT et les sorties.

A cette étape, le but est de pouvoir transformer un document basé sur les balises vues dans la partie précédente en des documents utilisant des langages existants comme l'HTML, le WML, ou le VoiceXML. Ces derniers sont en effet les trois langages que nous avons retenu actuellement pour mettre en avant les possibilités de PlasticML.

7.1 XSLT vers html.

Cette feuille de transformation axée dans un premier temps sur une tâche de formulaire contient toutes les routines pour transformer la totalité des champs de valeur d' un formulaire en « HTML ».

Le format « HTML » est le plus aisé à transformer. Notons ci-dessous la création d' une variable pour pointer sur les éléments <information> de la feuille PlasticMl.

```
<xsl:template match="information">
      <xsl:variable name="cible">
         <xsl:value-of select="@idref"/>
      </xsl:variable>
      <xsl:for-each select="id($cible)/texte[@lang=$param1]">
      <br/>
      <b>
         <xsl:value-of select="id($cible)/texte[@lang=$param1]"/>
      </b>
      <br/>
      </xsl:for-each>
   </xsl:template>
```

7.1.1 Les difficultés.

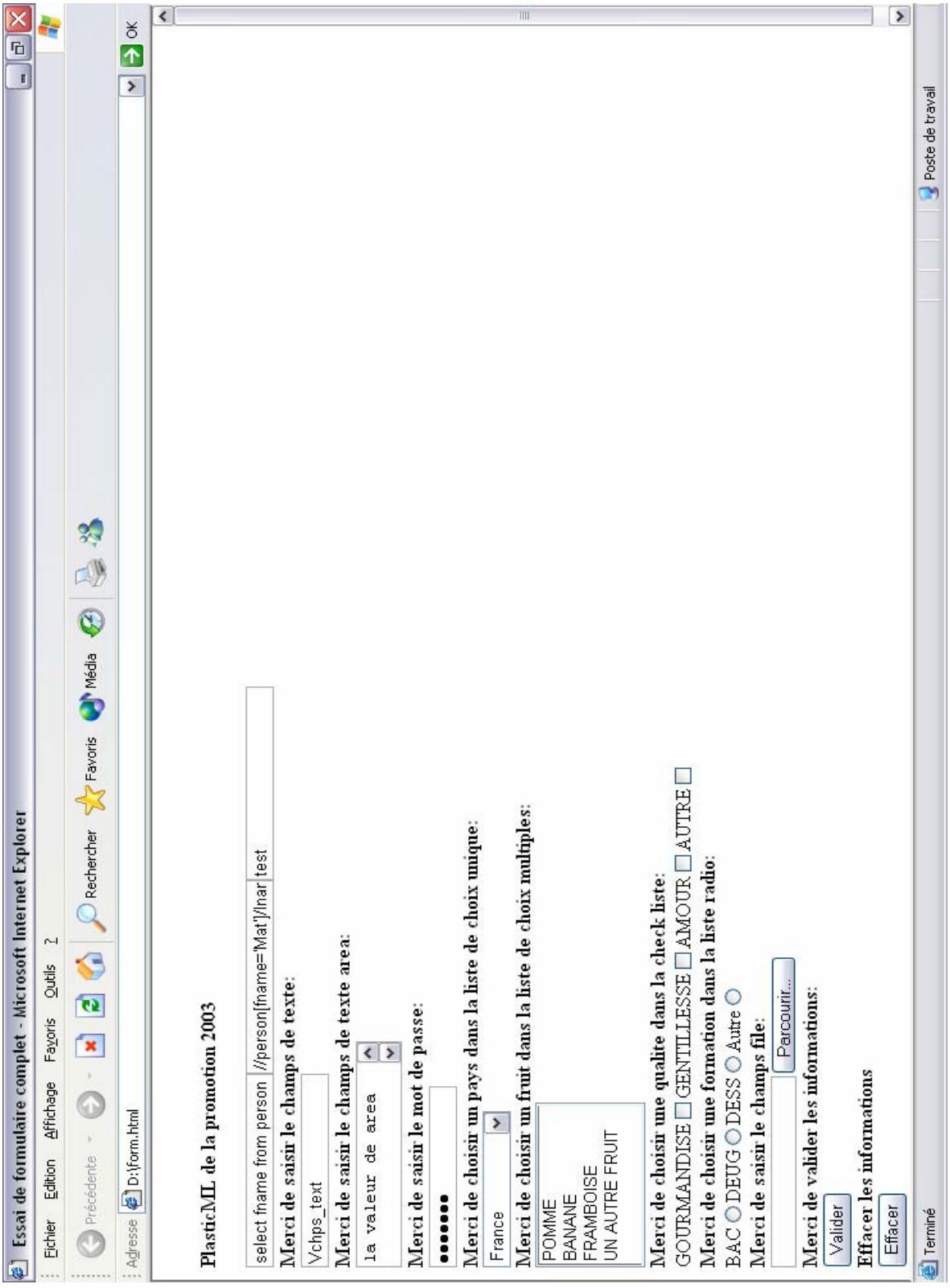
Notre ambition est de traiter la totalité des champs que l'on peut rencontrer dans un formulaire HTML, ce langage étant par essence le plus complet et le plus riche.

Certaines balises nous ont posé plus de problèmes que d'autres. Pour pallier ces difficultés nous avons dû utiliser et explorer le langage XSLT de manière détaillée car nous ne possédions pas les connaissances essentielles pour parvenir à un résultat tangible.

De plus, par souci d'honnêteté intellectuelle, nous décidons de ne mettre dans nos feuilles « XSLT » aucune balise du langage cible, ce qui nous a obligés à nous abstraire totalement de ce dernier pour pouvoir effectuer notre transformation.

Note : Le code de la feuille de transformation est dans le livret Document de travail en annexe.

7.1.2 Notre sortie HTML :



7.2 XSLT vers wml.

7.2.1 Les difficultés :

Le wml est destiné à des appareils aux écrans minuscules. Un fichier WML appelé **« deck » contient plusieurs cartes** qui sont des affichages écran. La transformation d' un fichier plasticML conséquent entraîne un dépassement des limites de l' écran.

7.2.2 Les solutions :

La solution idéale est de **récupérer dans les entêtes http** les valeurs de la taille de l' écran de l' utilisateur (ceci n' est pas implémenté aujourd' hui dans les normes).
L' utilisation de ces valeurs permettrait de découper notre <task> en ajoutant des éléments <sequence>. Ceux-ci indiqueraient en effet un découpage possible de la tâche en sous-tâches. Nous aurions alors **une automatisation de la détection du contexte** de surface d' affichage.
Dans l' immédiat, nous précalculons un découpage sur la base **de 1 champs=1 écran.**

- Lorsque l' utilisateur est sur un champs , il peut revenir à la carte precédente ou aller à la carte suivante.
- Lorsque l' utilisateur arrive au dernier champs qui est <submit> il peut revenir à la carte 1 ou valider

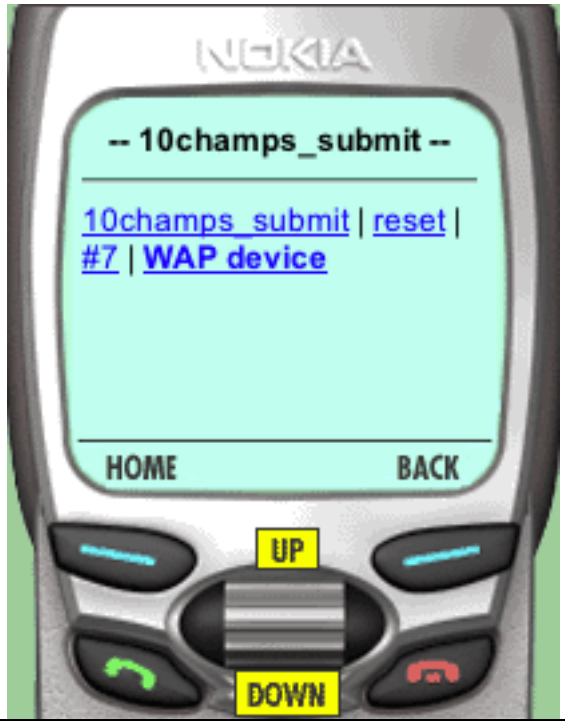
Déroulement de la séquence « WAP » :

Carte de bienvennue.

Le « texteArea » est identique au champs texte.

Le « checkbox»

La carte avec le champs submit permet de revenir à la première carte.



La méthode dans la feuille XSL pour opérer est la suivante :

Compter le nombre de champs <entry> que nous mettons dans une variable :

```
<xsl:variable name="card">
      <xsl:number value="count(/task/sequence/message/Entry)"/>
  </xsl:variable>
```

Compter le nombre de champs <entry> qui ne sont pas intéressants :

```
<xsl:variable name="nbhidden">
<xsl:number value="count(/task/sequence/message/entry[@type='hidden' or
@type='file' or @type='submit'])"/>
  </xsl:variable>
```

Créer une carte pour les nœuds **<entry>** présents à l' ecran :

Appliquer deux fois les templates sur <entry> pour placer les champs dans l' ordre choisi :

1. Les champs présents à l' ecran ,
2. Le champs submit.

```
<xsl:apply-templates select="entry[not(@type='hidden') and not(@type='file') and
not(@type='submit') and not(@type='reset')]"/>
    <xsl:apply-templates select="entry[@type='submit']"/>
```

Il ne reste plus qu' à **indiquer un id à chaque carte**….
(la fonction position() indique la position que l' on a choisi)

```
<xsl:when test="$letype='submit'">
  <xsl:value-of select="$loo"/>
</xsl:when>
<xsl:otherwise>
  <xsl:value-of select="position()"/>
</xsl:otherwise>
```

Et de **lier les cartes par un lien :**

```
<xsl:attribute name="label">
  <xsl:value-of select="'suivant'"/>
</xsl:attribute>
<go>
  <xsl:attribute name="href">#<xsl:number value="position()+1"/>
  </xsl:attribute>
</go>
```

Note : Le code de la feuille de transformation « wml » est dans le livret Document de travail en annexe.

7.3 XSLT vers Voice XML

Le déroulement d' une séquence voice est très contraignante parce qu' elle est linéaire : lorsqu' un champs est rempli , le processus passe automatiquement au champs suivant dont la valeur est nulle.

7.3.1 Les difficultés :

Avec champs à plusieurs sélections: l 'utilisateur ne peut indiquer qu' une valeur par champs et donc selectionner deux valeurs est virtuellement impossible.

7.3.2 Les solutions :

Le VoiceXML accepte les variables : nous créons une variable temporaire dans laquelle nous mettrons la selection.

7.3.2.1 Le dialogue obtenu en sortie :

Serveur : Merci de choisir un fruit dans la liste DE choix multiples: 1 POMME 2 BANANE 3 FRAMBOISE 4 UN AUTRE FRUIT
Réponse : *1 ou POMME*
Serveur : Desirez vous ajouter un nouveau choix dans cette liste (oui non)
Réponse : *1 ou oui*
Traitement : 1. la première réponse est sockée dans la variable « temp » , 2. la valeur du champs multiselection est remis à « null » , 3. le serveur voice revient donc sur le champs à plusieurs sélections.
Serveur : Merci de choisir un fruit dans la liste DE choix multiples: 1 POMME 2 BANANE 3 FRAMBOISE 4 UN AUTRE FRUIT
Réponse : *2 ou BANANE*
Serveur : Desirez vous ajouter un nouveau choix dans cette liste (oui non)
Réponse : *2 ou non*

Traitement :

1. la deuxième réponse est concaténée avec la première et remise dans la variable du champs à plusieurs sélections ,
2. la variable « temp » est remise à « null » ,
3. le serveur voice passe au champs suivant.

Le code voiceXML doit être le suivant avec <var name="temp" expr="0"/>
déclaré en tête du fichier:

7.3.2.2 Le code VOICEXML du dialogue :

```
<field name="Liste_choix_multiple">
<prompt>Merci de choisir un fruit dans la liste DE choix multiples:
   1 POMME 2 BANANE 3 FRAMBOISE 4 UN AUTRE FRUIT </prompt>
<option dtmf="1" value="valeur_POMME">POMME</option>
<option dtmf="2" value="valeur2">BANANE</option>
<option dtmf="3" value="valeur3">FRAMBOISE</option>
<option dtmf="4" value="valeur4">UN AUTRE FRUIT</option>
<filled>
<!--____multiple____-->
<goto nextitem="new_choice _F"/>
</filled>
</field>
<field name="new_choice_F">
<prompt> (new_choice6) desirez vous ajouter un nouveau
            choix dans cette liste</prompt>
<option value="1" dtmf="1">oui</option>
<option value="0" dtmf="2">non </option>
<filled>
<if cond="new_choice_F==1">
<if cond="temp==0">
<assign expr="Liste_choix_multiple" name="temp"/>
<else/>
<assign expr="temp +','+ Liste_choix_multiple" name="temp"/>
</if>
<clear namelist="Liste_choix_multiple"/>
<clear namelist="new_choice_F"/>
<prompt> la valeur de temp est <value expr="temp"/>
</prompt>
<else/>
<if cond="temp==0">
<else/>
<assign expr="temp +','+new_choice_F" name="new_choice_F"/>
</if>
<assign name="temp" expr="0"/>
</if>
</filled>
</field>
```

Deux champs « VoiceXML» dans le même document ne peuvent avoir le même nom. Nous avons crée une routine pour **allouer dynamiquement** une valeur au champs « new_choice » dans le cas ou nous aurions plusieurs champs à multiples sélections. Cependant , lors du retour sur le champs initial pour effectuer un deuxième choix , nous n' avons pas trouvé de solution pour retirer les choix déjà effectués.

7.3.3 Les options de transformation retenues.

Tous les champs sont transformés avec une **valeur « DTMF » imputée automatiquement** en plus de la valeur normale. Lors d' une selection multiple la liste des items ainsi que les numéros « DTMF »sont calculés automatiquement pour être dictés au premier <prompt>.

Exemple :

```
<prompt>
Merci de choisir un fruit dans la liste DE choix multiples:
   1   POMME  2  BANANE  3  FRAMBOISE  4  UN AUTRE FRUIT
</prompt>
```

7.3.4 La transformation avec le code XSLT.

Note : Le code de la feuille de transformation « VoiceXML » est dans le livret Document de travail en annexe.

8 Les outils de développement.

La phase de réflexion et de conception du langage, qui est encore en cours de développement, aboutit paralèllement à la **réalisation d' un éditeur** de PlasticML et **d' une servlet de traitement** pour accèder à des sources de données externes selon les requêtes et les sorties au travers du langage PlasticML

8.1 Le PlasticML Tooolkit.

Pour faciliter la création de pages en PlasticML, nous avons réalisé un outil de développement. Cet outil se veut **simple et le plus riche** possible en fonctionnalités. Le développement de ce PlasticML ToolKit s'est réalisé de manière très itérative, ponctuée de plusieurs réflexions sur les fonctions à rendre disponible en priorité.

8.1.1 Principe et philosophie.

Cet outil se veut être le petit « Dreamweaver » de PlasticML : il permet de manière simple et rapide la **création de pages en PlasticML** pour une personne qui ne maîtrise pas parfaitement le langage. Néanmoins une connaissance minimale est nécessaire pour pouvoir profiter de toutes les possibilités de ce dernier.

Pour faciliter l'utilisation, nous avons intégré :

- un éditeur « Wysiwyg »,
- un éditeur texte,
- un aperçu HTML
- ainsi que les visualisation des trois langages supportés (HTML, WML, VoiceXML).

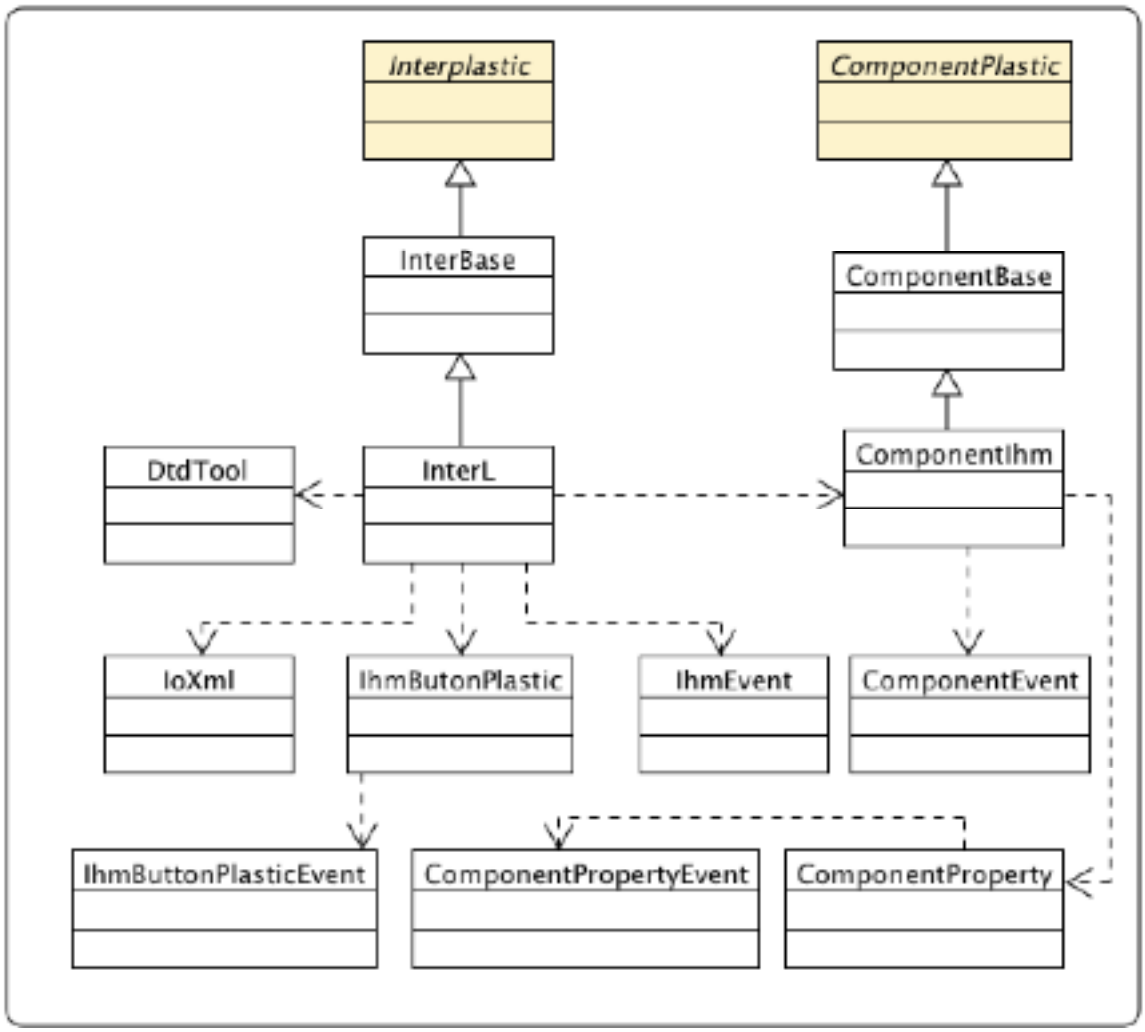
Nous avons également intégré une fonction d'évolution de l'interface. En effet celle-ci est basée sur la DTD PlasticML : **si la DTD actuelle évolue**, l'interface et ses possibilités de création **sont également mises à jour**.

Notre Toolkit adopte un fonctionnement simple avec une séparation de l'IHM et des actions que celle-ci peut réaliser. (**modèle MVC**) Nous avons d'ailleurs adopté ce principe de fonctionnement pour l'ensemble des classes d'IHM que nous manipulons. En fait nous pouvons distinguer trois types de classes dans notre application :

- Les classes qui gèrent l'interface générale
- Les classes qui gèrent les boutons de création de composants PlasticML
- Les classes qui gèrent les composants PlasticML à l'écran.

Chaque type de classe est composé d'une classe gérant l'IHM et d'une classe qui contrôle ses actions.

Le diagramme de classe simplifié suivant permet de réprésenter ces classes :



La classe *Interf* permet **de créer l'interface** graphique du ToolKit

La classe *IhmButonPlastic* permet de **créer les boutons** qui permettront d'ajouter des composants PlasticML

La classe *ComponentIhm* permet de **créer les composants** PlasticML qui seront accueillis par l'interface générale

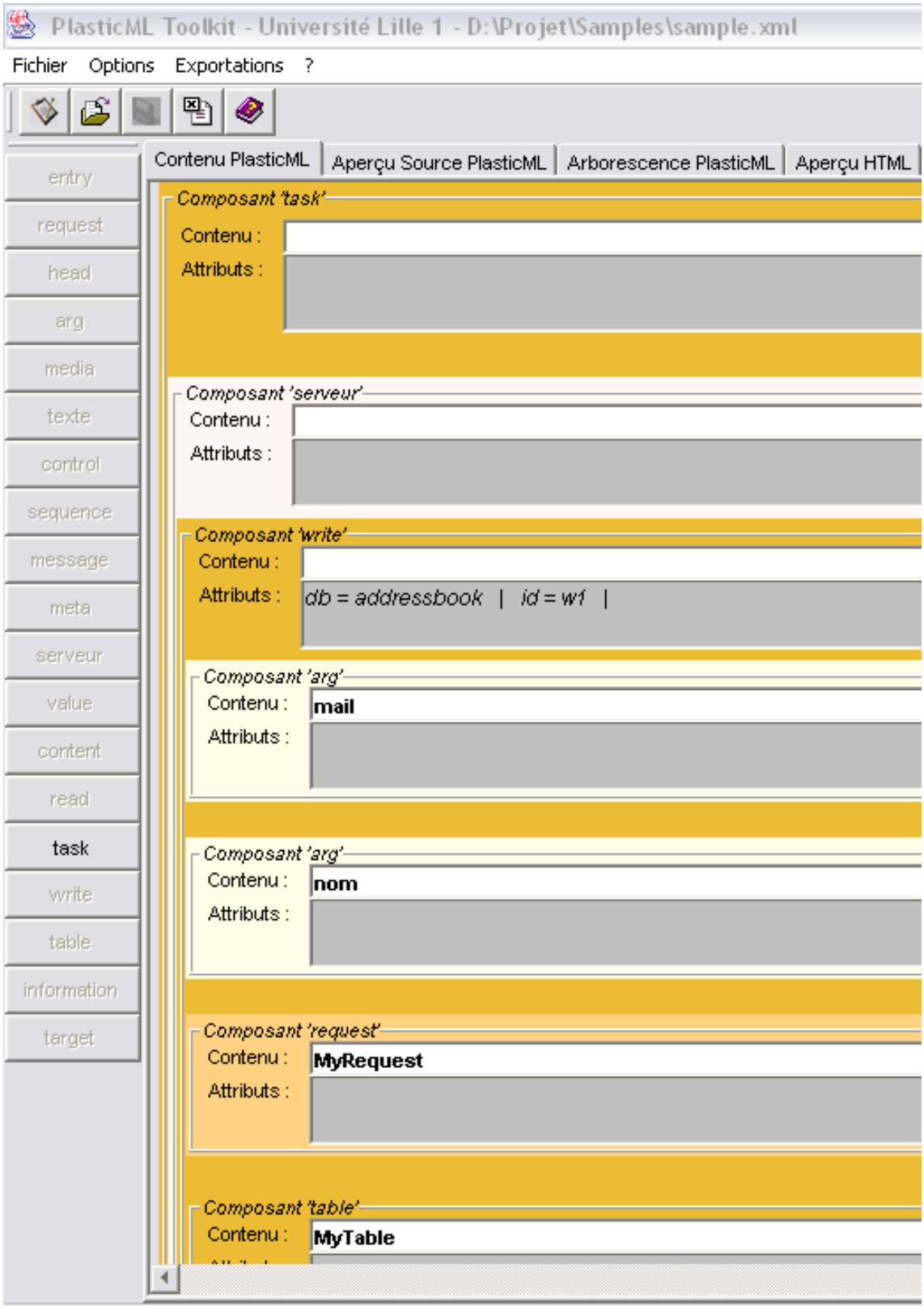
Il existe également deux classes qui nous permettent de gérer **les relations avec les document XML et XSL**. Ces deux classes sont *IoXml* et *DtdTool* . La première de ces classes nous permet de réaliser les actions qui sont inhérentes aux documents PlasticML traité. Elle comprend :

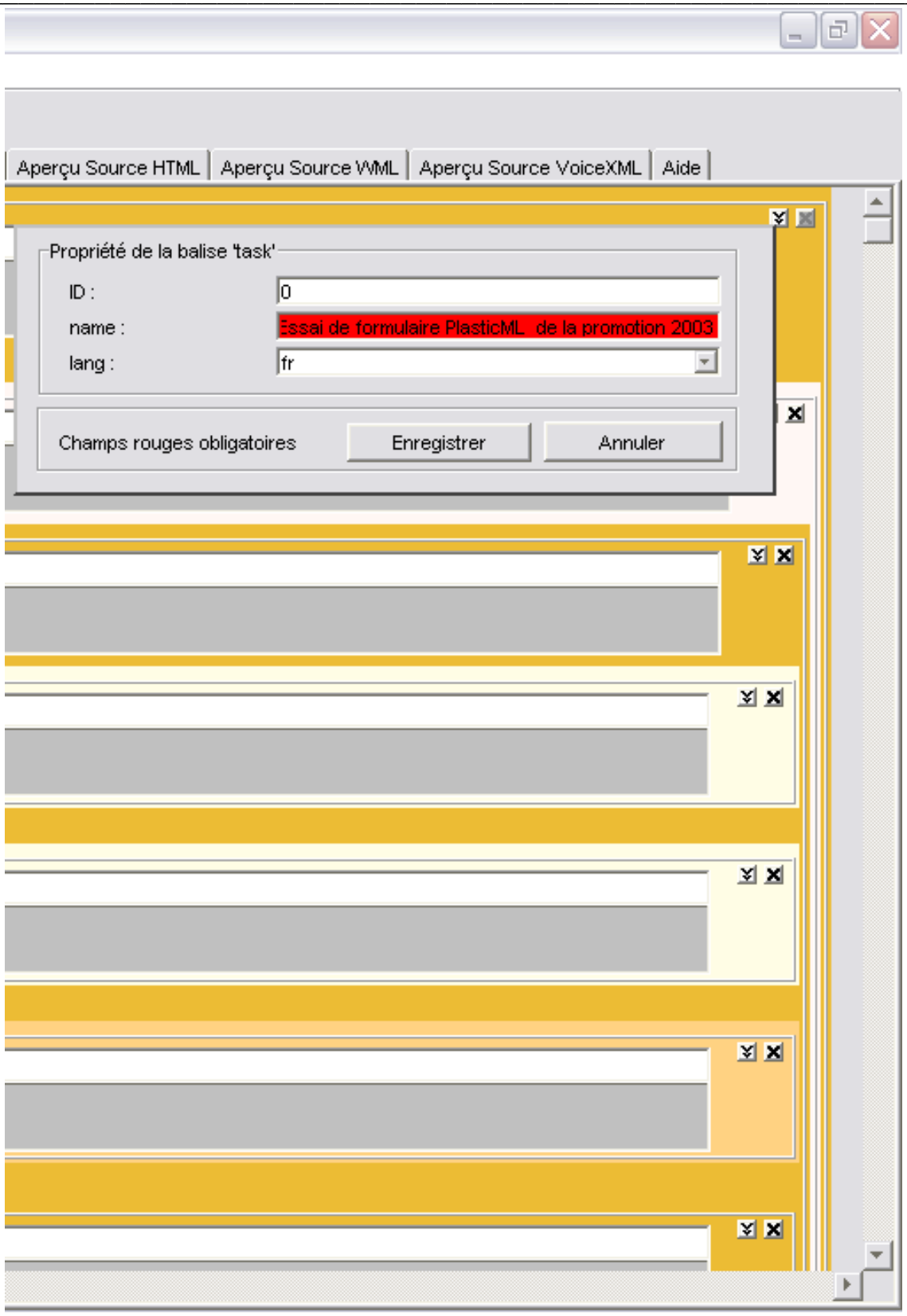
- Un fonction de création d'un nouveau document PlasticML basée sur DOM ,
- Une fonction de chargement d'un document existant ,
- Des fonctions de création d'éléments et d'attributs DOM ,
- Une fonction de vérification de confirmité à la DTD du document créé ,
- Une fonction de convertion vers HTML, WML ou Voice XML

Il est possible d'utiliser cette fonction avec de feuilles XSL de convertion à venir.

La seconde prend en charge les lectures de DTD qui nous permettent non seulement de vérifier la validité de notre document PlasticML mais aussi de générer automatiquement les fonctions de création du ToolKit PlasticML. Nous allons voir cette possibilité en détails dans la partie suivante.

Aperçu de l’ application PlasticML TooKit.




Aperçu Source HTML
Aperçu Source WML
Aperçu Source VoiceXML
Aide
Propriété de la balise 'task'
ID :
0
name :
Essai de formulaire PlasticML de la promotion 2003
lang :
fr
Champs rouges obligatoires
Enregistrer
Annuler

8.1.2 Génération automatique d' une interface à partir d' une DTD.

Une des **forces de notre « Toolkit** » est la prise en charge de **l'évolution** de notre DTD PlasticML. En effet chaque bouton de création des composants est un objet construit dynamiquement et basés sur la DTD PlasticML. Ainsi, si vous décidez de mettre à jour la DTD PlasticML qui supporte les dernières évolutions comme des nouveaux éléments, des nouveaux attributs pour les éléments existants, ceux-ci seront sauf cas exceptionnels pris en compte dans l'interface et **immédiatement disponibles** pour l'édition et l'ajout dans vos composants PlasticML.

Pour réaliser cette opération, le lancement de l'interface se réalise en plusieurs étapes :

- **Ouverture** de la DTD Plastic
- **Parsing de la DTD**
- **Chargement** des boutons

8.1.2.1 Etape 1 : Ouverture de la DTD.

L'ouverture de la DTD se fait de manière simple, par la création d'un objet fichier passé en paramètre à celui-ci, ainsi que d'un objet *FileReader* permettant sa lecture :

Constructeur de l'objet :

```
public DtdTool(String DtdFile) {
   f = new File(DtdFile) ;
 }
```

Le constructeur permet de créer l'objet « File » qui sera utilisé tout au long du « parsing » du document fourni en paramètres

8.1.2.2 Etape 2 : Parsing de la DTD.

Pour réaliser cette opération nous avons **effectué des recherches** pour connaître l'éxistance d'un éventuel **« parser » de DTD**. Nous devons reconnaître que cela ne fut pas une mince affaire. En effet, ceux-ci ne sont pas chose courante. Néanmoins nous nous sommes appuyés sur le travail de Mark WUTKA, un programmeur anglais , qui propose sur son site un parser de DTD libre.

Nous avons donc à partir de ce moment dû appréhender au mieux les possibilités de notre « Parser » qui en est à la version 1.2. Ces **possibilités**, se sont révelées en fait **nombreuses** et répondant parfaitement à nos besoins. A ce point nous sommes capables

de lire **tous les éléments d'un fichier DTD**, leurs attributs possibles, leurs attributs enfants possibles, les champs obligatoires….

Nous avons donc réalisé plusieurs fonctions capables de réaliser les opérations qui nous faisaient défaut pour obtenir le contenu de notre DTD :

Ces fonctions sont :

- *public Vector getTags()* : Cette fonction nous permet de récupérer la liste des éléments de la DTD dans un vecteur

- *public Vector getAttributes(String ParamTagName)* : Cette fonction retourne la liste des noms d'attributs d'un élément donné en paramètre

- *public DTDAttribute getAttribute(String ParamTagName, String ParamTagNameAttribute)* : Cette fonction permet de récupérer un objet Attribut dont le nom est fourni en paramètre dans un objet élément également fourni en paramètre

- *public Vector getItems(DTDAttribute ParamAttr)* : Cette fonction nous permet de récupérer les items d'un attribut dans le cas de création de choix multiples délimités (zone de liste).

- *public Vector getContentTag(String ParamTagName)* : Cette fonction nous permet de récupérer les éléments « Enfant » autorisés à être insérés dans un élément « Parent ».

Pour illustrer le fonctionnement de notre parser dans le cadre de notre Toolkit et de la génération automatique de son interface, nous allons maintenant vous présenter la fonction *getTag* qui , nous vous le rappelons , permet de récupérer l'ensemble des éléments d'une DTD.

```
public Vector getTags(){
   Vector TagList = new Vector();

   try{
         TempFileReader = new FileReader(TempFileReader);
         MyDtdParser = new DTDParser(TempFileReader);
         MyDtd=MyDtdParser.parse();
         e = MyDtd.elements.elements();

        while (e.hasMoreElements()) {
          DTDElement elem = (DTDElement) e.nextElement();
          TagList.addElement(elem.name);
        }
     }catch(IOException ex){
   }
   return TagList;
 }
```

Vous pouvez observer que la lecture du fichier se fait par l'intermédiaire d'un objet *FileReader* et d'un objet *DTDParser* inclus dans la librairie WUZKA. Ensuite la récupération des éléments de la DTD se fait par une lecture élément par élément de la collection préalablement créée . Puis nous retournons le vecteur consitué des noms d'éléments.

8.1.2.3 Etape 3 : Création des boutons.

Grâce au vecteur retourné par la fonction « *getTag* » précédemment illustrée, il nous est maintenant **possible de créer des boutons** de manière automatique dans notre interface permettant la création des divers éléments de la DTD.

Nos boutons gérant les composants PlasticML sont composés de deux classes distinctes. La première gérant l' IHM de notre bouton , et une seconde classe qui se charge de contrôler l'ensemble des événements qu'il peut générer.

8.1.3 Création d'un nouveau document..

La création d'un nouveau document se fait de manière simple en cliquant sur le bouton nouveau situé dans la barre d'outils ou dans le menu fichier. Le fait de cliquer sur ce bouton fera appel à la « *factory* » de document qui retournera un document DOM vierge.

Cette action remet également l'ensemble des variables à zéro ainsi qu' à un nettoyage de l'affichage des anciennes données qui pourraient éventuellement y être encore affichées.

Néanmoins une possibilité indispensable pour tout éditeur s'avère être la modification de documents existants. Nous allons maintenant détailler cette partie.

8.1.4 Chargement d'un document existant.

De loin cela est la fonctionnalité la **plus complexe à réaliser** , la possibilité de recharger un document vous permet de modifier un document PlasticML anciennement créé. Pour réaliser l'affichage de son contenu nous procédons à un parcours complet du document initialisant l'ensemble des variables de gestion qui sont utilisées dans le processus de création manuel et cela automatiquement. Ainsi les éléments sont **dynamiquement créés en Wysiswyg**, leurs attributs sont affichés, les nœuds auxquels ils correspondent sont chargés en mémoire pour une utilisation ultérieure.

Pour bien comprendre ce processus de chargement nous allons détailler dans les parties suivantes, les différentes phases de la création d'un composant.

8.1.5 Création des composants PlasticML.

Après avoir réalisé les boutons de création il nous est maintenant possible de créer des composants PlasticML en **Wysiwyg** dans l'application.

Cette tâche n'est pas la plus simple car nous avons dû tenir compte des **contraintes d'emboîtements de composants** dans Java. Néanmoins nous sommes parvenus à un résultat **tout à fait probant**.

Les composants PlasticML sont donc composés dans notre interface de plusieurs classes comme nous avons déjà pu le signaler auparavant :

- La classe d'interface *ComponentIhm*
- La classe de gestion des événement *ComponentEvent*

Nous avons également ajouté une classe qui permet via une interface graphique l'accès aux attributs du composant. Cette classe est nommée *ComponentProperty*.

Après un clic sur un bouton le procédé de création de composant est lancé. Celui-ci se déroule en plusieurs étapes :

- La création du composant graphique ,
- La saisie des attributs du composant ,
- L’ajout à l’arbre DOM ,
- L’ajout à l’interface graphique générale.

8.1.5.1 Etape 1 : La création du composant graphique

Pour créer un composant grahique, nous faisons simplement appel à la classe *ComponentIhm*. Nous définisons le type de composant à créer via l'identifiant du bouton qui a lancé l'action de création. Cette action ainsi réalisée, nous pouvons lancer une recherche sur la constitution du composant à créer. Notre classe *DtdTool* précédemment utilisée pour la création de l'interface du ToolKit entre à nouveau en action ici. En effet elle nous permet de **récupérer l'ensemble des attributs** qui doivent constituer les composants à créer. Il nous est donc possible de demander à l'utilisateur de renseigner ces attributs.

Pour cela nous utilisons les fonctions « *getAttributes* » «*getAttribute* » et « *getItems* » précedemment présentées. Elles permettent de constituer la liste des attributs et ainsi de construire dynamiquement les champs à renseigner pour le composant et pour le panneau d'attibuts de ce composant.

8.1.5.2 Etape 2 : La saisie des attributs du composant

A chaque début d'une phase de création de composant, nous proposons à l'utilisateur de **renseigner les champs** correspondants à ses attributs. Cette saisie se fait par un **panneau de propriété** qui est créé par la classe « *ComponentProperty* ». Au sein de celle-ci l'utilisateur peut renseigner les différents attributs qui concerneront le composant.

Notez bien que ce panneau de propriété est également construit dynamiquement en fonction des attributs et de leur nombre qui sont indiqués dans la DTD.

L'utilisateur dispose après avoir renseigné les champs de deux alternatives. La première est tout simplement l'annulation de l'action d'ajout et un retour à l'état antérieur de l'application. La seconde est l'enregistrement. C'est dans le cadre de cet enregistrement que se réalise l'étape 3

8.1.5.3 Etape 3 : L'ajout à l'arbre DOM

Lors de cette étape le composant a été validé par l'utilsateur. Le programme se charge donc de terminer son ajout à l'interface. Mais auparavant il va réaliser l'ajout de celui-ci en tant que composant DOM à l'objet DOM qui gère le document PlasticML. C'est à cette étape que se réalise l'action d'ajout en PlasticML et c'est seulement après celle-ci que nous réalisons l'ajout à l'interface.

8.1.5.4 Etape 4 : L'ajout à l'interface graphique générale

Ce passage est **l'étape ultime** de la procédure de création de composant. En effet nous réalisons ici l'ajout effectif du composant à l'interface. Celui-ci sera maintenant visible pour l'utilisateur qui pourra bien-sûr par la suite **le modifier via son panneau de propriété** intégré. Pour assurer le suivi dans le programme et procéder à son retrait éventuel, l'objet est également stocké dans un vecteur local au composant parent qui l' accueille et

dans un vecteur global à l'interface, ceci pour la gestion de l'affichage et du raffraichissement lors d'un retrait de composant.

C'est ainsi que se déroule la procédure de création de composant à l'écran via **l' éditeur Wysiwig**. Notez bien que dans ce cadre , le logiciel propose dans ce cas un mise à jour de l'éditeur en mode texte dès que celui-ci doit être utilisé.

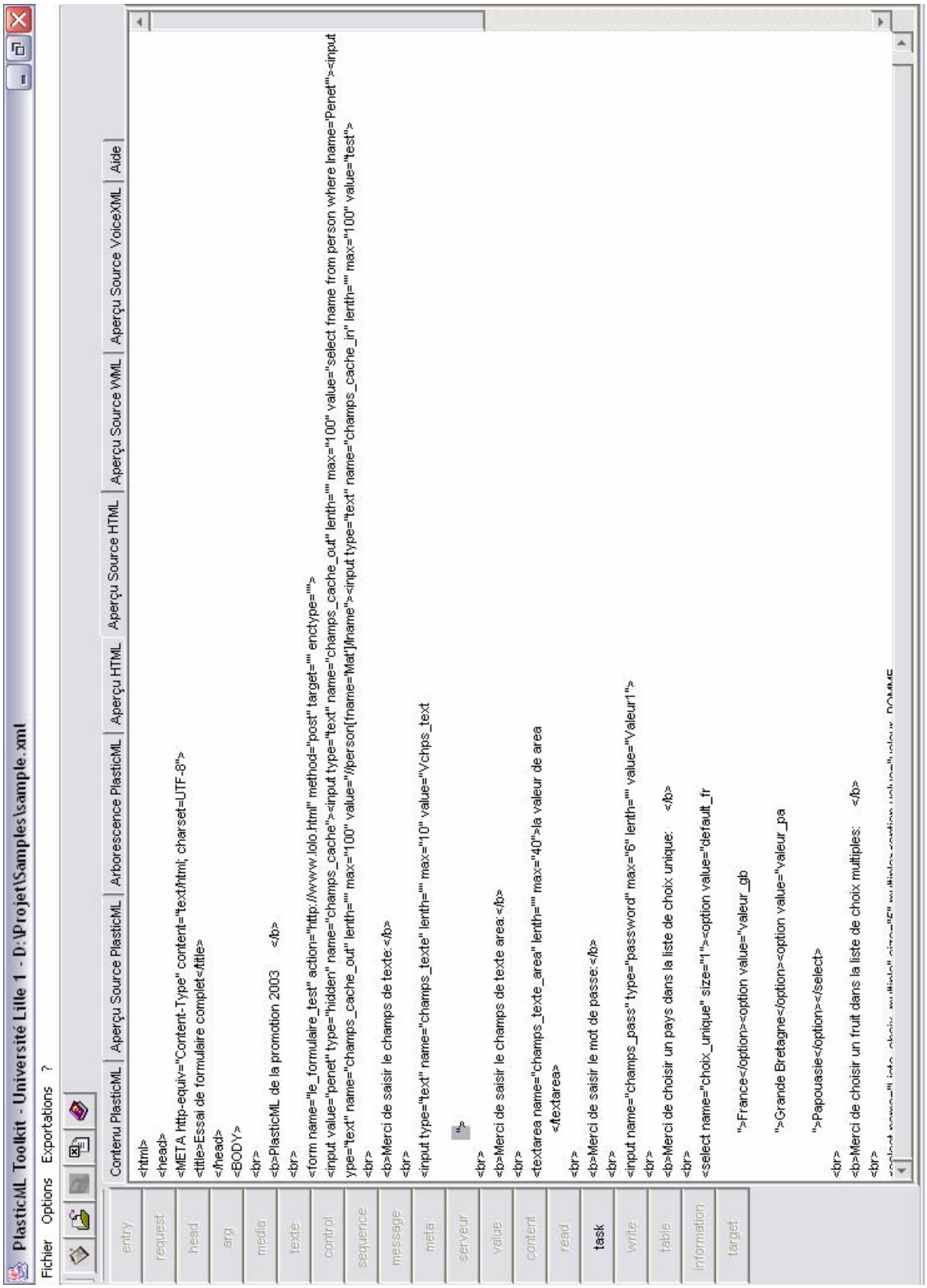
8.1.6 Les possibilités d'exportations de documents

Le but principal de PlasticML est de réaliser **des documents** qui soient **exploitables** sur plusieurs périphériques différents. Il était donc naturel de réaliser un module d'exportation de document vers le format voulu. Nous disposons donc à l' heure actuelle de trois modules d'export distincts.

Le premier nous permet de générer à la volée **des documents HTML**, le second des **documents WAP** et le troisième des **documents VoiceXML**. Cette exportation serialise dans une fenêtre de visualisation ainsi que dans un fichier .

Les caractérstiques prises en compte dans l'exportation sont uniquement celles qui sont intégrées dans la version actuelle de PlasticML. Ainsi nous ne pouvons pas actuellement gérer des structures de tableau pour les documents HTML ou WAP….

Aperçu de la fonction d' exportation de code HTML dans PlasticML ToolKit.



8.2 L' Application PlasticML Servlet.

8.2.1 Principe de fonctionnement.

L' application **est basée sur les servlets de java**. Le servlet est générique dans sa mission qui consiste à renvoyer la sortie désirée .

Le processus se déroule en 14 phases (voir figure 7.3):

1. Réception de la requête , (le nom du fichier plasticML est dans la requête)
2. Analyse des entêtes http ,
3. Traitement « DOM » des fichiers PlasticML et db.xml pour rechercher les valeurs des nœuds et des attributs (in, put, write read) ,
4. Isolation des valeurs dans des objets ,
5. Evaluation en fonction paramètres de la requête rentrante et des descriptions de pilotes dans le fichier db.xml et du type des nœuds ,
6. Traitements différencié des valeurs ,
7. **write** pour écriture dans une source externe ,
8. **out** pour lecture des données dans une source externe ,
9. **write** pour écriture dans une source externe ,
10. **in** pour redéposition dans le fichier plasticML ,
11. **read** pour lecture,
12. Remplacement des valeurs selectionnées dans le fichier PlasticML ,
13. Transformation avec la feuille xslt choisie ,
14. Renvoi sur l' utilisateur.

8.2.2 Les données extérieures vers des SGBD multiples.

Nous avons choisi pour tester le processus trois systèmes de bases de données pour montrer l' aspect ouvert du process.

8.2.2.1 Xindice.

La base de données Xindice est dévelopée sous le projet Apache : Xindice exploite le standard XML:DB (qui n'est pas encore un candidat du W3C) pour gérer des ensembles de documents XML qui peuvent aisément être interrogés à l'aide de XPath. Ce choix est **le prolongement naturel d' un développement basé sur XML** : il apporte une valeur ajoutée dans le projet ainsi qu' une logique « tout XML ».

La difficulté liée à ce type de source est le retour des informations qui reviennent sous la forme d' un document XML dont il faut extraire les valeurs ; cela suppose de **connaître en amont la structure des fichiers** stockés dans la base de données et la structure des fichiers de retour pour effectuer une **extraction DOM** des valeurs de nœuds désirées.

Exemple : une requête de type :

//person[fname='John']
renvoie
<fname xmlns:src="http://xml.apache.org/xindice/Query"
src:col="/db/addressbook" src:key="parts">
Mat
</fname>

Nous connaissons la structure du message retourné et nous en extrayons la valeur « Mat » pour procéder ensuite au remplacement dans le fichier PlasticML.

8.2.2.2 ODBC.

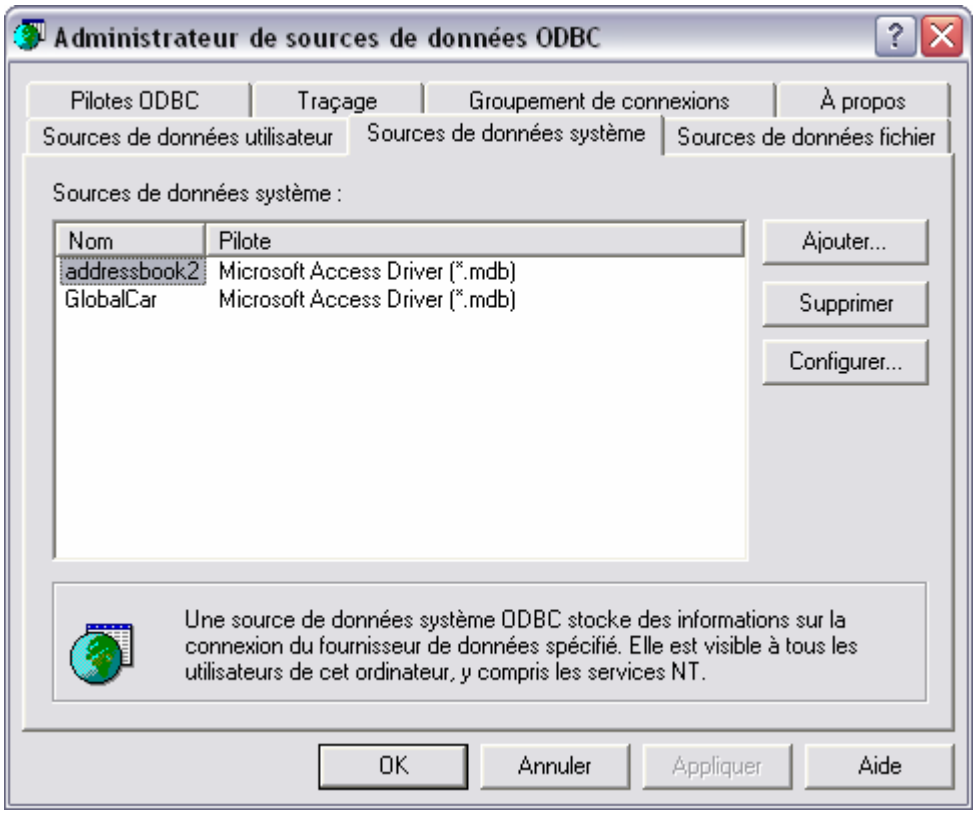
Nous avons également ajouté la possibilité d'accéder à des bases de données ODBC (Open DataBase Connectivity) pour donner la possibilité d'utiliser PlasticML avec d'autres source de données plus conventionnelle basées sur SQL. De plus ODBC est un standard qui est utilisé par un très grand nombre d'utilisateurs.

Nous avons donc réussi à prouver que notre langage couplé à notre Servlet permet une interaction avec des sources de données extérieures.

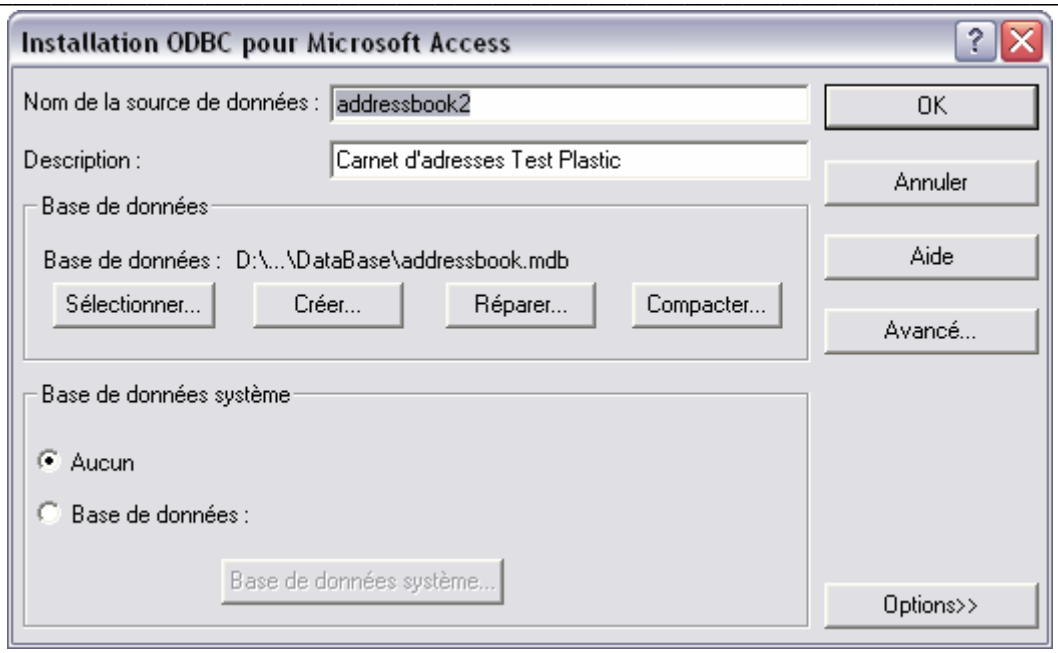
Pour pouvoir utiliser ODBC il est nécessaire d'avoir sur votre machine des serveurs de bases de données ainsi que les pilotes ODBC correspondants.

Nous avons procédé à des test sous les bases de données Access et Mysql.

Avant de pouvoir utiliser PlasticML avec des données externes issues d'une base de données ODBC vous devez créer sur votre machine serveur un pointeur ODBC vers la base correspondante . Pour cela il vous faut lancer le gestionnaire de sources de données ODBC de votre machine. Voici un aperçu de celui-ci sous Microsoft Windows XP :



Vous pouvez observer dans la fenetre ci-dessus que nous avons créé pointeur de base de donnée nommé « addressbook2 ». Celui-ci possède les propriété suivantes :



Ce pointeur utilise le pilote ODBC Microsoft Acces et pointe sur une base de données Access physique installée sur le disque de l'ordinateur.

Un fois ces opérations réalisées, vous pouvez créer vos documents PlasticMLavec pour les sources de données « addressbook2 » et effectuer des requetes SQL sur cette base pour en extraire des données.

8.2.3 Mécanismes détaillés ou lecture et écriture dans les sources externes

L' application va en premier traiter les contenus de l' élément <serveur>
Ces contenus représentent les opérations à effectuer sur les sources externes avant le remplacement des valeurs.

8.2.3.1 Lecture :

Elément < value type="out"> :

Exemple de code XML :	<content id="bienvenue2" db="message" type="**out**"> *contenu[id='bienvenue']* </content>
Traitement source :	1. Lire le fichier db.xml et trouver la ref de la source « message » 2. Executer la requête : *contenu[id='bienvenue']* 3. Stocker le resultat de la requête.
Traitement PlasticML :	Redéposer les valeurs dans le fichier PlasticML

Elément <content type =" out "> :

Exemple de code XML :	<entry name="champs_cache_out" type="hidden"> <value type="**out**" db="addressbook" id="v2"> *//person[fname='John']* </value> </entry>
Traitement source :	1. Lire le fichier db.xml et trouver la reférence de la source « addressbook » 2. Executer la requête : *//person[fname='John']* 3. Stocker le resultat de la requête.

Traitement PlasticML :	Redéposer les valeurs dans le fichier PlasticML.

8.2.3.2 Lecture.

L' élément « read » n' a pas encore été implémenté pour le traitement, notre impératif est dans un premier temps de montrer qu' une connection est possible : cependant nous avons refléchi à un scénario de traitement posssible.

Elément <read> :

Exemple de code XML :	***<read>*** < read id="w1" db="addressbook" > <arg>login</arg> <arg>pass</arg> <request>// login *[login ='John']* <request> <error>erreur.xml</error> <null>erreur.xml</null> <target idref="T1" ></target> <target idref="T2" ></target> ***</ read >***
Traitement source :	1. Lire le fichier db.xml et trouver la ref « addressbook » 2. Prendre les valeurs des paramètres « login » et « pass » 3. Lire le fichier db.xml et trouver la ref « addressbook » 4. Effectuer une requête sur la source 5. En cas de retour null rediriger sur la page « erreur.xml »
Traitement PlasticML :	Redéposer les valeurs dans les éléments dont les attributs « ID » sont "T1" et "T2"

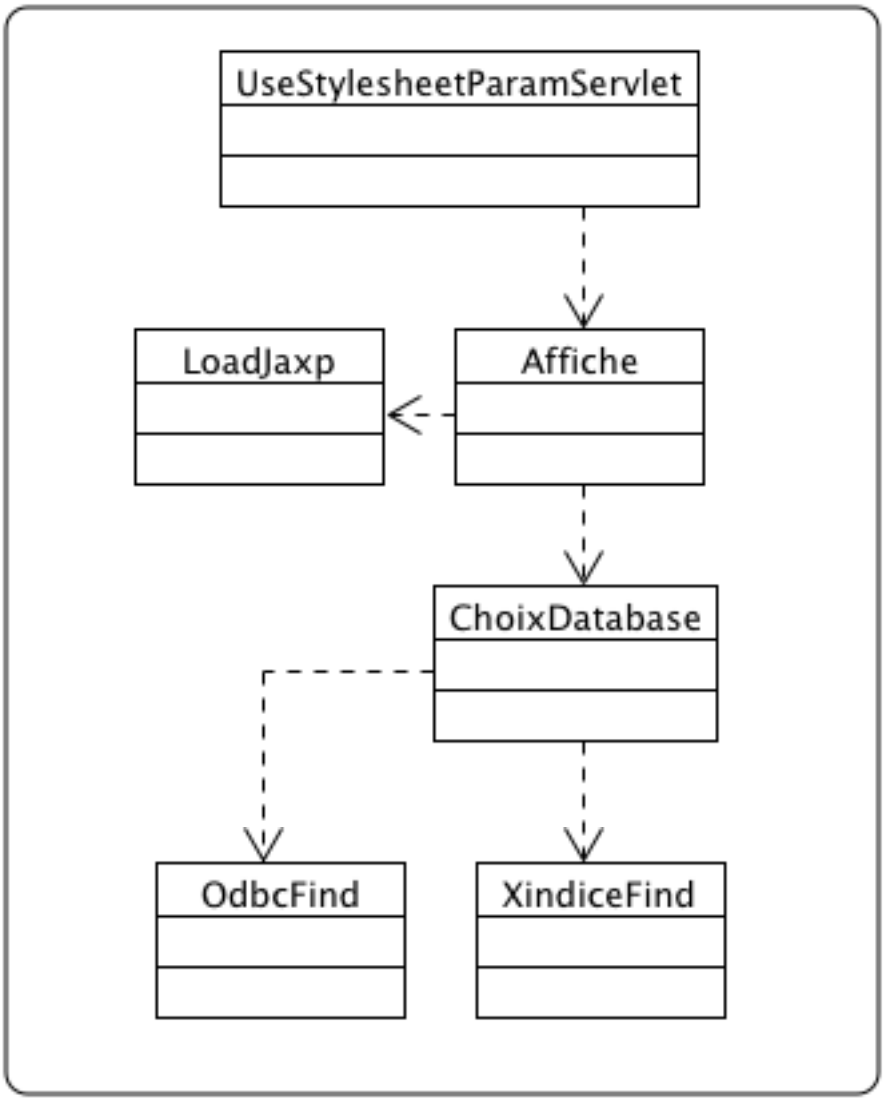
8.2.3.3 Écriture :

Elément <serveur><write> :

Exemple de code XML :	***<write*** id="w1" db="addressbook" > Insert into person (nom, mail) values(« ***», »***») <arg>mail</arg><arg>nom</arg> <table> person</table> ***</write>***
Traitement :	1. Lire le fichier db.xml et trouver la ref de la source « addressbook » 2. Prendre les valeurs des paramètres « mail » et « nom » 3. Ecrire dans la source externe.

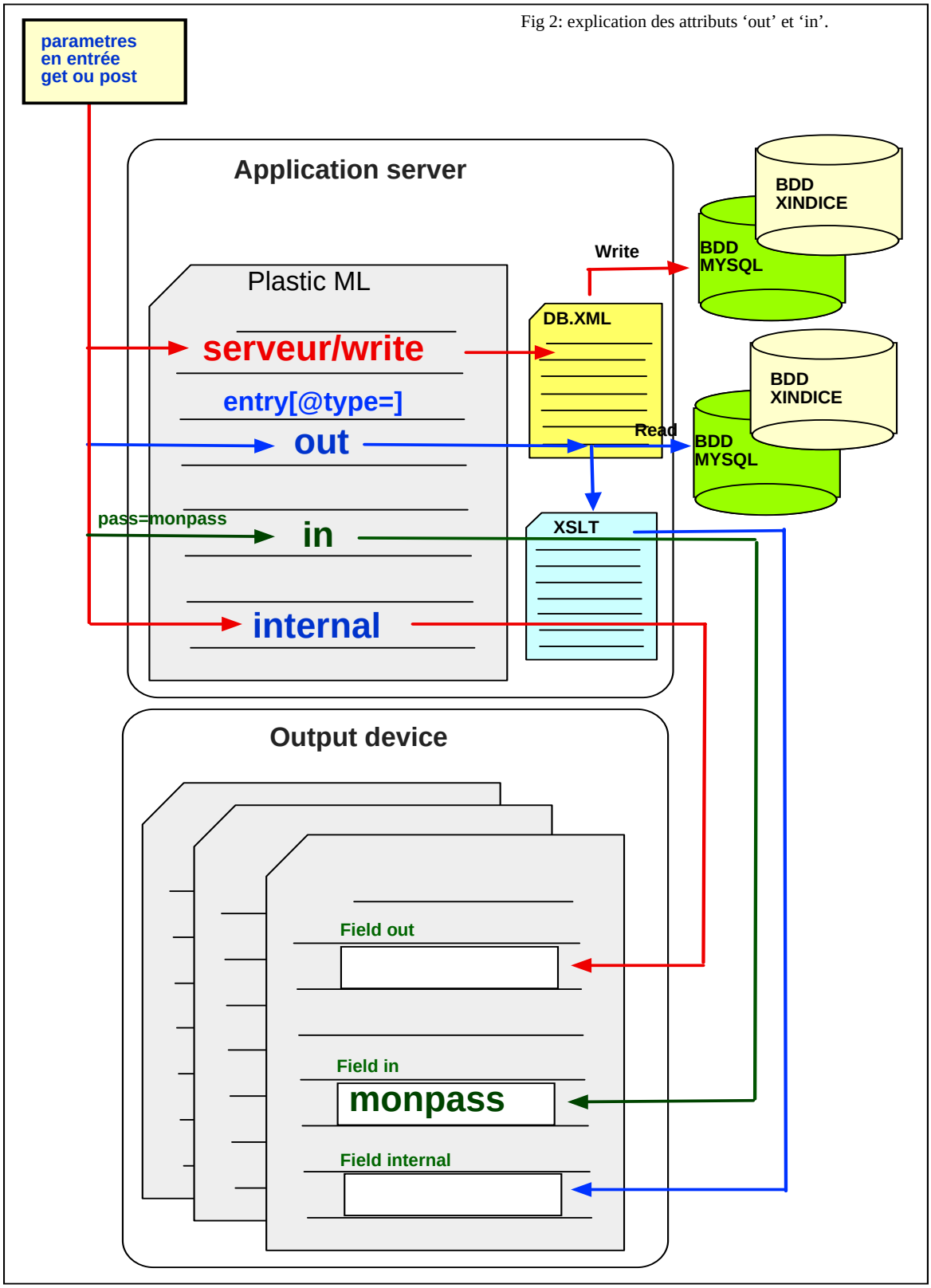
8.2.3.4 Diagramme de classes.

- Classe *UseStylesheetParamServlet* : réception de la requête ,
- Classe *Affiche* : traitement « DOM » des fichier PlasticML et db.xml ,
- Classe *LoadJaxp* : crée les documents DOM ,
- Classe *ChoixDatabase* : controleur des sources externes ,
- Classes *OdbcFind* et *XindiceFind* : execution et récupération des données.



La figure 2 montre le fonctionnement des attributs ‘in’ ou ‘out’ pour les éléments <content> et <value>.

Fig 2: explication des attributs ‘out’ et ‘in’.



9 Conclusions.

9.1 PlasticML et la mise en page.

Le temps imparti pour réaliser le projet ne nous permet pas de traiter cet aspect de PlasticML. Cependant, nous pensons qu' une solution envisageable est d ' isoler les éléments <content> dans un autre fichier « XML » **avec un « Namespace » différent** et d' appliquer une feuille « XSLT » appropriée pour traiter la mise en page.

9.2 La transmission du projet.

Nous avons documenté notre recherche le mieux possible : de cette façon ,**le relai est établi** . Nous epérons qu ' **une prochaine équipe** améliorera notre travail. Nous sommes à disposition pour **répondre** à toutes interrogations ou les questions concernant celui-ci.

9.3 Développements futurs…

Les axes de recherches sont nombreux dans le monde autour du thème de la plasticité des « IHM ». Cependant , une piste de recherche vers un langage qui marie les **interactions de la voix et de l' écriture** (son et écran) nous paraît intéressante et prometteuse en termes économiques.

9.4 Les regrêts.

Nous avons en fin de parcours découvert « COCOON » qui nous a interpellé , et que nous n' avons pas pu approfondir : ce n' est que partie remise !

9.5 Les succès.

Des moments forts ont rythmé notre parcours : notamment l' instant où nous avons lié plasticML à **trois bases de données en simultané** et retrouvé les données externes formatées sur le document de sortie. Nous pensons avoir été à la hauteur de la mission confiée et nous savons que des erreurs ont été commises car nous ne sommes que des élèves : **de la diversité naît la richesse !**

10 Sources de documentation.

http://ictt.insa-lyon.fr/fr/DEA/dea2001_2.htm
Titre : **Etude des différents protocoles XML** supportant la plasticité des interfaces homme-machine. Application aux interfaces de télé-enseignement.
Lieu du stage : Bat Leonard de Vinci INSA de LYON
Résumé :
De par la multiplication des périphériques d'affichage (écran, portable, téléphone, PDA...), des contextes d'utilisation (travail, maison, déplacement) et des profils des utilisateurs, est né le besoin de concevoir des interfaces homme-machine adaptable. On parle alors de plasticité des interfaces...
---

http://www.isys.ucl.ac.be/etudes/memoires/jv/liste/JV-MP-00-01.htm
Sujet de mémoire JV-MP-00-01
Professeur: Jean Vanderdonckt
vanderdonckt@qant.ucl.ac.be
Titre: Production d'une passerelle inter-**environnementale XIML <-> Qtk**
Programmes : INGE2, IAG2M, GETI3DS, INFO2
Objectif: Le consortium international XIML propose un langage d'expression d'interfaces homme-machine basé sur la technologie XML. QTk est un outil avancé de création d'interfaces homme-machine...
Organisation: XIML consortium, Mozart Consortium representant local : Peter Van Roy - FSA/INGI/INFO
---

http://www.isys.ucl.ac.be/bchi/publications/index.htm
quelques publications
---

http://edmi.imag.fr/Themes_recherche_ED/usage_perception.html
Sujet 3 : Modèles, mécanismes et **outils pour Interfaces Homme-Machine plastiques**
Thème : Interaction Homme-Machine, Espace perceptif et objets communicants, Informatique diffuse (ubiquitous computing)
Responsables : Joëlle Coutaz et Gaëlle Calvary {Joelle.Coutaz, Gaelle.Calvary}@imag.fr
Sujet et Motivations
La thèse proposée s'inscrit dans le domaine de l'Interaction Homme-Machine (IHM) et répond à un nouveau requis de l'informatique diffuse : l'adaptation des Interfaces Homme-Machine...
---

http://iihm.imag.fr/thevenin/papiers/RD-IHM'99/IHM'99-Plasticite.html/
La plasticité en interaction Homme-Machine (David Thevenin)
Laboratoire CLIPS-IMAG, Equipe IIHM

BP 53 38041 Grenoble Cedex 9 France
david.thevenin@imag.fr
framwork pour la publication d'interfaces plastiques (Gaëlle Calvary, Joëlle Coutaz, and David Thevenin)
ehci01-calvary.pdf
La plasticité en interaction Homme-Machine (David Thevenin)
plastic.pdf
Adaptation en Interaction Homme-Machine :(David Thevenin)
le cas de la Plasticité
THESE2001_Thevenin.pdf
---

http://iihm.imag.fr/publs/2001
Équipe Ingénierie de **l'Interaction Homme-Machine**
L'équipe Ingénierie de l'Interaction Homme-Machine (IIHM) fait partie du laboratoire ....
Equipe IIHM, Laboratoire CLIPS-IMAG
BP 53, 38041 Grenoble Cedex 9, France
Tél: +33 (0)476 514 854
Fax: +33 (0)476 446 675
Email: iihm@imag.fr
---

http://arnaud.buisine.free.fr/these.html
Mourad ABED
Titre : Conception et **évaluation des systèmes homme-machine** : spécification formelle et semi-formelle, méthodes, modèles et outils
Contact : Christophe.Kolski@univ-valenciennes.fr [Site Web LAMIH]
Titre : Vers une démarche industrielle pour le développement d'Interfaces Homme-Machine
Contact : http://arnaud.buisine.free.fr/
---

http://lis.univ-tlse1.fr/ALF/
Groupe de travail Architecture Langages et Fomalismes
---

http://www-ufrima.imag.fr/PLACARD/RICM3/IHM-TC/documents/cours%20Joelle/
Joelle Coutaz cours
---

http://rangiroa.essi.fr/rainbow/overview.html
L'équipe Rainbow propose un modèle et des outils pour la construction et l'exécution des applications réparties nécessitant d'être **adaptées à leur environnent**.
doc papier: Rainbow.pdf
---

http://www.afihm.org/theses/

David Navarre
Titre : Contribution à **l'ingénierie en Interaction Homme-Machine** : Une technique de description formelle et un environnement pour une modélisation et une exploitation synergiques des tâches et du système
Descriptif : annonce, jury, résumé
Lien : http://lihs.univ-tlse1.fr/navarre
Contact : navarre@univ-tlse1.fr, palanque@univ-tlse1.fr
Mourad ABED
Titre : Conception et évaluation des systèmes homme-machine : spécification formelle et semi-formelle, méthodes, modèles et outils
Contact : Christophe.Kolski@univ-valenciennes.fr [Site Web LAMIH]
Titre : Vers une démarche industrielle pour le développement d'Interfaces Homme-Machine
Lien : http://arnaud.buisine.free.fr/these.html
Contact : http://arnaud.buisine.free.fr/
---

http://iihm.imag.fr/publs/2001
Équipe Ingénierie de l'Interaction Homme-Machine
L'équipe Ingénierie de l'Interaction Homme-Machine (IIHM) fait partie du laboratoire ....
Equipe IIHM, Laboratoire CLIPS-IMAG
BP 53, 38041 Grenoble Cedex 9, France
Tél: +33 (0)476 514 854
Fax: +33 (0)476 446 675
Email: iihm@imag.fr
---

http://lis.univ-tlse1.fr/ALF/
Groupe de travail Architecture Langages et Fomalismes
---

http://www.lisi.ensma.fr/ftp/pub/documents/thesis/
THÈSE Pour l'obtention du grade de DOCTEUR de l'UNIVERSITÉ DE POITIERS
Guillaume Patry
École Nationale Supérieure de Mécanique et d'Aérotechnique ...
... la complexité des **modèles des applications de conception**, il est impossible de déléguer complètement à la couche de présentation du modèle ARCH [1] la ...
doc papier: 1999-ihm-patry.pdf, 1999-thesis-patry.pdf
---

11 Annexes

11.1 DTD de PlasticML 1.1

```
<?xml version="1.0" encoding="UTF-8"?>
<!ELEMENT task (serveur,sequence*,content*)>

<!ELEMENT serveur (write*)*>
<!ELEMENT write (arg+)>
<!ELEMENT arg ( #PCDATA)>
   <!ATTLIST write
id ID #REQUIRED
db CDATA #REQUIRED
>
<!ELEMENT texte ( #PCDATA)>
<!ELEMENT media ( #PCDATA)>
   <!--..(control*,media*,texte*)*..-->
<!ELEMENT content   (#PCDATA | media|texte)*>

<!ATTLIST content
id ID #REQUIRED
db CDATA #IMPLIED
type (internal | out |in)  'internal'
>
<!--une tache peut etre composee de plusieurs ecrans contenant de l' affi-
chage(information) ....-->
<!ELEMENT sequence (head , control* , information* , message*)>

<!ELEMENT head (meta* ,  control*)>
   <!ELEMENT meta ( #PCDATA)>

<!-- information peut etre partout -->
<!ELEMENT information ( #PCDATA)>
<!-- delocalisation des info ds un autre ficher cela ne marche pas -->
<!ATTLIST information
 idref CDATA #REQUIRED
>
<!--Un message correspond au form-->
<!ELEMENT message (information*,entry*)*>
<!ELEMENT entry (information*,value*,control*)*>
<!ELEMENT value  (#PCDATA | information)* >

<!ATTLIST texte
```

```
lang   CDATA #REQUIRED
link CDATA #IMPLIED
>
   <!ATTLIST media
src CDATA #REQUIRED
link CDATA #IMPLIED
format CDATA #IMPLIED
lang   CDATA #IMPLIED
>
<!NOTATION jpeg SYSTEM "image/jpeg">
<!NOTATION png SYSTEM "image/png">
<!NOTATION srgs SYSTEM "application/srgs+xml">

<!ELEMENT control (#PCDATA | information | media)* >

   <!ATTLIST control
type (grammar|filled|help|nomatch|noinput|script) #REQUIRED
>

<!ATTLIST task
name CDATA #REQUIRED
ID CDATA #IMPLIED
lang   (CDATA|fr) 'fr'
>
<!ATTLIST sequence
name CDATA #REQUIRED
>
<!ATTLIST message
name CDATA #REQUIRED
method (post|get|put) 'post'
action CDATA #REQUIRED
enctype CDATA #IMPLIED
target CDATA #IMPLIED
class  CDATA #IMPLIED
id  CDATA #IMPLIED
>

<!ATTLIST entry
type (choice|reset|submit|secret|file|text|hidden) #IMPLIED
name CDATA #REQUIRED
lenth CDATA #IMPLIED
max CDATA #IMPLIED
format  CDATA #IMPLIED
sort (checkbox|radio|choice|multiple|area) #IMPLIED
required (true | false )  'true'
>
```

<!--type: file pour un fichier sur le disque-->

<!ATTLIST value
id ID #REQUIRED
type (internal | out | in) 'internal'
default (yes|no) 'no'
db CDATA #IMPLIED
>

11.2 Feuille xslt vers HTML.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<!—récupération des paramètres de langue du moteur de servlet -->
<xsl:param name="param1" select="'fr'"/>
   <xsl:variable name="lang">
      <xsl:value-of select="/task/@lang"/>
   </xsl:variable>
   <xsl:output method="html"/>
   <!-- template racine -->
   <xsl:template match="/">
      <html>
         <xsl:apply-templates/>
      </html>
   </xsl:template>


   <!-- template task -->
   <xsl:template match="task">
         <xsl:apply-templates select="sequence"/>
   </xsl:template>

<xsl:template match="sequence">
<xsl:apply-templates select="head"/>
<BODY>

<xsl:apply-templates select="information"/>
<xsl:apply-templates select="message"/>
</BODY>
</xsl:template>

   <!-- template head  -->
<xsl:template match="head">
      <head>
         <xsl:apply-templates select="meta"/>
         <xsl:apply-templates select="control"/>
         <title>
            <xsl:value-of select="../@name"/>
         </title>
      </head>
</xsl:template>

<xsl:template match="meta">
```

```
   <!-- scripts pas deploye pour l instant
   <meta>
 <xsl:attribute name="name">
<xsl:value-of select="@name"/>
 </xsl:attribute>
 </meta>
 -->
</xsl:template>

<xsl:template match="control">
<xsl:choose>
   <xsl:when test="@type='script'">
      <!-- scripts pas deploye pour l instant -->
   </xsl:when>
</xsl:choose>
</xsl:template>



   <!-- les contenus d' information -->
   <xsl:template match="information">
      <xsl:variable name="cible">
         <xsl:value-of select="@idref"/>
      </xsl:variable>
      <xsl:for-each select="id($cible)/texte[@lang=$param1]">
      <br/>
      <b>
         <xsl:value-of select="id($cible)/texte[@lang=$param1]"/>
      </b>
      <br/>
      </xsl:for-each>
   </xsl:template>

   <!-- template message -->
   <xsl:template match="message">
      <xsl:apply-templates select="information"/>
      <form>
         <!-- traitement des attributs de entry -->
         <xsl:attribute name="name">
            <xsl:value-of select="@name"/>
         </xsl:attribute>
         <xsl:attribute name="action">
            <xsl:value-of select="@action"/>
         </xsl:attribute>
         <xsl:attribute name="method">
            <xsl:value-of select="@method"/>
```

```
        </xsl:attribute>
        <xsl:attribute name="target">
          <xsl:value-of select="@target"/>
        </xsl:attribute>
        <xsl:attribute name="enctype">
          <xsl:value-of select="@enctype"/>
        </xsl:attribute>
        <xsl:apply-templates select="entry"/>
      </form>
  </xsl:template>
  <!-- positionner sur template entry -->
  <xsl:template match="entry">
    <xsl:apply-templates select="information"/>
    <xsl:for-each select=".">
      <xsl:choose>
        <!-- si le  type est hidden  -->
        <xsl:when test="@type='hidden' or  @type='file'">
          <input>
            <xsl:attribute name="value"><xsl:value-of select="./value"/></xsl:attribute>
            <xsl:attribute name="type">
              <xsl:value-of select="@type"/>
            </xsl:attribute>
            <xsl:attribute name="name">
              <xsl:value-of select="@name"/>
            </xsl:attribute>
          </input>
        </xsl:when>
        <!-- si le  type est texte area  -->
        <xsl:when test=" @type='text'  or @type='submit'  or @type='reset' ">
          <xsl:choose>
            <xsl:when test=" @sort='area' ">
              <textarea>
                <xsl:attribute name="name">
                  <xsl:value-of select="@name"/>
                </xsl:attribute>
                <xsl:attribute name="lenth">
                  <xsl:value-of select="@lenth"/>
                </xsl:attribute>
                <xsl:attribute name="max">
                  <xsl:value-of select="@max"/>
                </xsl:attribute>
                <xsl:value-of select="./value"/>
              </textarea>
            </xsl:when>
            <xsl:otherwise>
              <!-- si le  type est texte  -->
```

```
                <input>
                   <xsl:attribute name="type">
                      <xsl:value-of select="@type"/>
                   </xsl:attribute>
                   <xsl:attribute name="name">
                      <xsl:value-of select="@name"/>
                   </xsl:attribute>

                   <xsl:choose>
                      <xsl:when test="@type='text'">
                         <xsl:attribute name="lenth">
                      <xsl:value-of select="@lenth"/>
                   </xsl:attribute>
                   <xsl:attribute name="max">
                      <xsl:value-of select="@max"/>
                   </xsl:attribute>
                   <xsl:attribute name="value">
                      <xsl:value-of select="./value"/>
                   </xsl:attribute>
                      </xsl:when>
                      <xsl:otherwise>
                <!--  submit ou reset la valeur du nom du bouton doit etre ds la langue-->
                         <xsl:attribute name="value">
                         <xsl:variable name="cible">
                            <xsl:value-of select="./value/information/@idref"/>
                         </xsl:variable>
                         <xsl:value-of select="id($cible)/texte[@lang=$param1]"/>
                         </xsl:attribute>
                      </xsl:otherwise>
                   </xsl:choose>
                </input>
             </xsl:otherwise>
          </xsl:choose>
       </xsl:when>
       <!-- si le  type est secret  -->
       <xsl:when test=" @type='secret'">
          <input>
             <xsl:attribute name="name">
                <xsl:value-of select="@name"/>
             </xsl:attribute>
             <xsl:attribute name="type">password</xsl:attribute>
             <xsl:attribute name="max">
                <xsl:value-of select="@max"/>
             </xsl:attribute>
             <xsl:attribute name="lenth">
                <xsl:value-of select="@lenth"/>
```

```
                </xsl:attribute>
                <xsl:attribute name="value"><xsl:value-of select="./value"/></xsl:attribute>
             </input>
          </xsl:when>
          <!-- si le  sort est choice    -->
          <xsl:when test=" @type='choice' ">
             <!--  lolo    -->
             <xsl:choose>
                <xsl:when test=" @sort='checkbox'  or @sort='radio' ">
                   <xsl:for-each select="./value">
                      <xsl:variable name="cible">
                         <xsl:value-of select="./information/@idref"/>

                      </xsl:variable>
                      <xsl:value-of select="id($cible)/texte[@lang=$param1]"/>
                      <input>
                         <xsl:attribute name="type">
                            <xsl:value-of select="../@sort"/>
                         </xsl:attribute>
                         <xsl:attribute name="name">
                            <xsl:value-of select="../@name"/>
                         </xsl:attribute>
                         <xsl:attribute name="value"><xsl:value-of se-
lect="."/></xsl:attribute>
                      </input>
                   </xsl:for-each>
                </xsl:when>
                <!--CHOIX MULTIPLES -->
                <xsl:otherwise>
                   <select>
                      <xsl:attribute name="name">
                         <xsl:value-of select="@name"/>
                      </xsl:attribute>
                      <xsl:attribute name="size">
                         <xsl:value-of select="@max"/>
                      </xsl:attribute>
                      <xsl:choose>
                         <xsl:when test=" @sort='multiple' ">
                            <xsl:attribute name="multiple">
                               <xsl:value-of select="@sort"/>
                            </xsl:attribute>
                         </xsl:when>
                      </xsl:choose>
                      <xsl:for-each select="./value">
                         <option>
    <xsl:attribute name="value"> <xsl:value-of select="."/></xsl:attribute>
```

```
                        <xsl:variable name="cible">
                            <xsl:value-of select="./information/@idref"/>
                        </xsl:variable>
                        <xsl:value-of select="id($cible)/texte[@lang=$param1]"/>
                    </option>
                </xsl:for-each>
            </select>
        </xsl:otherwise>
    </xsl:choose>
</xsl:when>
<!-- si le  sort est MULTIchoice  -->
<xsl:when test=" @sort='check' ">
 <!--   <xsl:value-of select="@Description"/> -->
    <xsl:attribute name="name">
        <xsl:value-of select="@name"/>
    </xsl:attribute>
</xsl:when>
</xsl:choose>
</xsl:for-each>
</xsl:template>
<!-- positionner sur template value -->
<xsl:template match="value">
    <xsl:value-of select="."/>
</xsl:template>
</xsl:stylesheet>
```

11.3 Feuille xslt vers WML.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
   <xsl:output doctype-public="-//WAPFORUM/DTD WML 1.1//EN"
      doctype-system="http://www.wapforum.org/DTD/wml_1.1.xml" indent="yes" me-
thod="xml"/>
   <!-- variables pour le calcul du nbre de cartes ds le deck -->
   <xsl:variable name="Entry">0</xsl:variable>
   <xsl:variable name="card">
      <xsl:number value="count(/task/sequence/message/Entry)"/>
   </xsl:variable>
   <xsl:variable name="neucour" select="2"/>
   <!-- template racine -->
   <xsl:template match="/">
      <wml>
         <xsl:apply-templates/>
      </wml>
   </xsl:template>
   <!-- on mettra ici les comportements par defaut du deck type <template> -->
   <xsl:template match="Head"/>
   <!-- template head -->
   <!-- template tache a realiser correspond a une carte du deck a couper en subtask par
la suite -->
   <xsl:template match="task">
      <card>
         <xsl:attribute name="ID">
            <xsl:value-of select="@ID+($card+1)"/>
         </xsl:attribute>
         <xsl:attribute name="title">
            <xsl:value-of select="@name"/>
         </xsl:attribute>
         <xsl:attribute name="id">
            <xsl:value-of select="@ID"/>
         </xsl:attribute>
         <xsl:apply-templates select="sequence"/>
      </card>
   </xsl:template>
   <xsl:template match="sequence">
      <xsl:apply-templates select="information"/>
      <xsl:apply-templates select="message"/>
   </xsl:template>
   <!-- les contenus d' information -->
   <xsl:template match="information">
```

```
<xsl:value-of select="@xlink"/>
<xsl:variable name="cible">
<xsl:value-of select="@idref"/>
</xsl:variable>
<xsl:value-of select="id($cible)"/>
</xsl:template>
<!-- fin_____les contenus d' information -->
<!-- template message -->
<xsl:template match="message">
<xsl:apply-templates select="information"/>
<do>
<xsl:attribute name="type">
<xsl:value-of select="@type"/>
</xsl:attribute>
<!-- valeur de l'element submit -->
<xsl:attribute name="label">
<xsl:value-of select="@name"/>
</xsl:attribute>
<go>
<!-- calcul de la carte suivant -->
<xsl:choose>
<!-- Quand le dernier champs n' est pas atteind -->
<xsl:when test="( ( $neucour )&lt;(($card)-1) )">
<xsl:attribute name="href">#<xsl:number value="($neucour)+1"/>
</xsl:attribute>
</xsl:when>
<xsl:otherwise>
<!--on est au rang nbrechamps mois1 -->
<xsl:attribute name="href">
<xsl:value-of select="@Action"/>
</xsl:attribute>
</xsl:otherwise>
</xsl:choose>
<xsl:attribute name="method">
<xsl:value-of select="@type"/>
</xsl:attribute>
<xsl:attribute name="sendreferer">
<xsl:value-of select="@sendreferer"/>
</xsl:attribute>
<xsl:attribute name="class">
<xsl:value-of select="@class"/>
</xsl:attribute>
<xsl:attribute name="id">
<xsl:value-of select="@id"/>
</xsl:attribute>
</go>
```

```
        <xsl:choose>
          <!-- Quand le dernier champs n' est pas atteind -->
          <xsl:when test="($neucour)&lt;(4)"> message-neucourant:<xsl:number va-
lue="($neucour)"/>
            <xsl:variable name="neucour" select="($neucour)+1"/>
            <xsl:apply-templates select="/task/sequence/message/Entry"/>
          </xsl:when>
        </xsl:choose>
      </do>
   </xsl:template>
   <xsl:template name="Lolo">
      <xsl:variable name="neucour" select="($neucour)+1"/>
<xsl:number value="($neucour)"/>
<xsl:apply-templates select="/task/sequence/message/Entry"/>
   </xsl:template>
   <!-- positionner sur template Entry  Entry[position()=$neucour]-->
   <xsl:template match="Entry">

      <xsl:choose>
        <xsl:when test="( (position()=$neucour) )and( ($neucour)&lt;(($card)) )"> ok
            (cartes:<xsl:number value="($card)"/>--- position:<xsl:number
          value="(position())"/>--- neucourant:<xsl:number value="($neucour)"/>---)
<xsl:choose>
            <xsl:when test=" position()=($card) ">
                <xsl:variable name="pos" select="position()"/>
            </xsl:when>
            <xsl:when test=" @type='submit' or  @type='file'"/>
            <xsl:when test=" @type='hidden' ">
              <postfield>
                <xsl:attribute name="position">
                  <xsl:value-of select="position()"/>
                </xsl:attribute>
                <xsl:attribute name="neucour">
                  <xsl:value-of select="$neucour"/>
                </xsl:attribute>
                <xsl:attribute name="card">
                  <xsl:value-of select="$card"/>
                </xsl:attribute>
                <xsl:attribute name="name">
                  <xsl:value-of select="@name"/>
                </xsl:attribute>
                <xsl:attribute name="value">
                  <xsl:value-of select="./value"/>
                </xsl:attribute>
              </postfield>
            </xsl:when>
```

```
<!-- _____si le  sort est choice  -->
<!-- si le  type est choice  -->
<xsl:when test=" @type='choice' ">
   <xsl:apply-templates select="information"/>
   <select>
      <xsl:attribute name="name">
         <xsl:value-of select="@name"/>
      </xsl:attribute>
      <xsl:attribute name="value">
         <xsl:value-of select="@name"/>
      </xsl:attribute>
      <xsl:attribute name="title">
         <xsl:value-of select="@title"/>
      </xsl:attribute>
      <xsl:attribute name="iname">
         <xsl:value-of select="@iname"/>
      </xsl:attribute>
      <xsl:attribute name="ivalue">
         <xsl:if test="@Default ='yes'">1</xsl:if>
      </xsl:attribute>
      <xsl:attribute name="multiple">
         <xsl:choose>
            <xsl:when test=" @sort='multiple' or'checkbox' ">true</xsl:when>
            <xsl:otherwise>false</xsl:otherwise>
         </xsl:choose>
      </xsl:attribute>
      <xsl:attribute name="xml:lang">
         <xsl:value-of select="@xml:lang"/>
      </xsl:attribute>
      <xsl:attribute name="id">
         <xsl:value-of select="@id"/>
      </xsl:attribute>
      <xsl:attribute name="class">
         <xsl:value-of select="@class"/>
      </xsl:attribute>
      <xsl:for-each select="./value">
         <option>
            <xsl:choose>
               <xsl:when test=" ../@sort='checkbox'  or ../@sort='radio' ">
                  <xsl:attribute name="value">
                     <xsl:value-of select="."/>
                  </xsl:attribute>
                  <xsl:value-of select="./@Description"/>
               </xsl:when>
               <xsl:otherwise>
                  <xsl:attribute name="value">
```

```
                        <xsl:value-of select="./@value"/>
                    </xsl:attribute>
                    <xsl:value-of select="."/>
                </xsl:otherwise>
            </xsl:choose>

        </option>
    </xsl:for-each>
</select>
</xsl:when>
<!-- si le  type est texte ou password  -->
<xsl:when test=" @type='text' or @type='secret' ">
<xsl:apply-templates select="information"/>
<!-- ___champs texte  -->
<input>
    <xsl:choose>
        <!-- si le  type  password  -->
        <xsl:when test=" @type='secret' ">
            <xsl:attribute name="type">password</xsl:attribute>
        </xsl:when>
        <xsl:otherwise>
            <xsl:attribute name="type">text</xsl:attribute>
        </xsl:otherwise>
    </xsl:choose>
    <xsl:attribute name="value">
        <xsl:value-of select="./value"/>
    </xsl:attribute>
    <xsl:attribute name="name">
        <xsl:value-of select="@name"/>
    </xsl:attribute>
    <xsl:attribute name="format">
        <xsl:value-of select="@format"/>
    </xsl:attribute>
    <xsl:attribute name="emptyok">
        <xsl:value-of select="@emptyok"/>
    </xsl:attribute>
    <xsl:attribute name="size">
        <xsl:value-of select="@Lenth"/>
    </xsl:attribute>
    <xsl:attribute name="maxlength">
        <xsl:value-of select="@Max"/>
    </xsl:attribute>
    <xsl:attribute name="title">
        <xsl:value-of select="@title"/>
    </xsl:attribute>
    <xsl:attribute name="accesskey">
```

```
                <xsl:value-of select="@accesskey"/>
            </xsl:attribute>
            <xsl:attribute name="xml:lang">
                <xsl:value-of select="@xml:lang"/>
            </xsl:attribute>
            <xsl:attribute name="id">
                <xsl:value-of select="@id"/>
            </xsl:attribute>
            <xsl:attribute name="maclass">
                <xsl:value-of select="@maclass"/>
            </xsl:attribute>
        </input>
    </xsl:when>
    <!--reset -->

    <xsl:when test="@type='reset' ">
        <xsl:value-of select="./value"/>
        <do>
            <xsl:attribute name="type">options</xsl:attribute>
            <xsl:attribute name="label">
                <xsl:value-of select="@name"/>
            </xsl:attribute>
            <refresh>
                <xsl:attribute name="class">
                    <xsl:value-of select="../@class"/>
                </xsl:attribute>
                <xsl:attribute name="id">
                    <xsl:value-of select="../@id"/>
                </xsl:attribute>
                <xsl:attribute name="lolo">
                    <xsl:variable name="card" select="($card)-1"/>
                </xsl:attribute>
                <xsl:for-each select=".">
                    <setvar>
                        <xsl:attribute name="name">
                            <xsl:value-of select="@name"/>
                        </xsl:attribute>
                        <xsl:attribute name="value">
                            <xsl:value-of select="../@id"/>
                        </xsl:attribute>
                        <xsl:attribute name="id">
                            <xsl:value-of select="../@id"/>
                        </xsl:attribute>
                        <xsl:attribute name="class">
                            <xsl:value-of select="../@id"/>
                        </xsl:attribute>
```

```
                        </setvar>
                    </xsl:for-each>
                </refresh>
            </do>
        </xsl:when>

        </xsl:choose>
        <xsl:variable name="neucour" select="($neucour)+1"/> fin
            entry---neucourantincremente:<xsl:number value="($neucour)"/>
        <xsl:call-template name="Lolo"/>
        <!--<xsl:apply-templates select="/task/sequence/message"/> -->
    </xsl:when>
  </xsl:choose>
 </xsl:template>
 <!-- fin_____________________ltemplate message -->
</xsl:stylesheet>
```

11.4 Feuille xslt vers VoiceXML.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
   <xsl:output indent="yes" method="xml"/>
   <xsl:variable name="temp">0</xsl:variable>
   <xsl:variable name="lang">
      <xsl:value-of select="/task/@lang"/>
   </xsl:variable>
   <!-- racine template -->
   <xsl:template match="/">
      <vxml>
         <xsl:attribute name="base">
            <xsl:value-of select="'.'"/>
         </xsl:attribute>
         <xsl:attribute name="version">
            <xsl:value-of select="'1.0'"/>
         </xsl:attribute>
         <xsl:attribute name="lang">
            <xsl:value-of select="$lang"/>
         </xsl:attribute>
      </vxml>
      <var expr="0" name="temp"/>
      <var expr="0" name="champs"/>
      <xsl:apply-templates/>
   </xsl:template>
   <!-- template head pour le futur -->
   <xsl:template match="head"/>
   <!-- template task -->
   <xsl:template match="task">
      <form>
         <block>
            <xsl:value-of select="@name"/>
            <goto>
               <xsl:attribute name="next">#<xsl:value-of se-
lect="./sequence/message/@name"/>
               </xsl:attribute>
            </goto>
         </block>
      </form>
      <xsl:apply-templates select="sequence"/>
   </xsl:template>

   <xsl:template match="sequence">
      <xsl:apply-templates select="message"/>
```

```
    </xsl:template>
    <!-- les contenus d' information -->
    <xsl:template name="information">
        <xsl:variable name="cible">
            <xsl:value-of select="./information/@idref"/>
        </xsl:variable>
        <xsl:apply-templates select="/task/content[@id=$cible]"/>
    </xsl:template>
    <xsl:template match="content">
        <xsl:apply-templates select="texte[@lang=$lang]"/>
    </xsl:template>
    <xsl:template match="texte">
        <xsl:value-of select="."/>
    </xsl:template>
    <!-- template message -->
    <xsl:template match="message">
        <form>
            <xsl:attribute name="id">
                <xsl:value-of select="@name"/>
            </xsl:attribute>
            <!-- appel au traitement des champs -->
<xsl:apply-templates select="entry[not(@type='submit' or @type='file'  or
@type='reset')]"/>
            <block>
                <submit>
                    <xsl:attribute name="next">
                        <xsl:value-of select="@action"/>
                    </xsl:attribute>
                    <!-- construction de la namelist a envoyer ici -->
                    <xsl:attribute name="namelist">
                        <xsl:for-each select="entry">
                            <xsl:if test="@type != 'reset' and @type != 'submit' ">
                                <xsl:value-of select="@name"/>
                                <xsl:value-of select="concat(' ', ')"/>
                            </xsl:if>
                        </xsl:for-each>
                    </xsl:attribute>
                </submit>
            </block>
        </form>
        <!-- _____voir ici pssibilite de reset-->
        <!-- _____voir ici pssibilite de quitter-->
    </xsl:template>
    <!-- positionner sur template entry -->
    <xsl:template match="entry">
```

<xsl:comment>_____</xsl:comment>

```
<xsl:choose>
   <xsl:when test="@type='hidden' ">
      <xsl:comment>champs cache</xsl:comment>
      <var>
         <xsl:attribute name="expr">
            <xsl:value-of disable-output-escaping="yes" select="./value"/>
         </xsl:attribute>
         <xsl:attribute name="name">
            <xsl:value-of select="@name"/>
         </xsl:attribute>
      </var>
   </xsl:when>
   <!-- champs pas cache-->
   <xsl:otherwise>
      <field>
         <xsl:attribute name="name">
            <xsl:value-of select="@name"/>
         </xsl:attribute>
         <xsl:if test="@format!="">
            <xsl:attribute name="type">
               <xsl:value-of select="@format"/>
            </xsl:attribute>
         </xsl:if>
         <prompt>
            <xsl:call-template name="information"/>
            <!--   -->
            <xsl:if test="@type='choice'">
               <xsl:for-each select="./value[@default='yes']">
                  <xsl:value-of select="position()"/>
                  <xsl:value-of select="'  '"/>
                  <xsl:call-template name="information"/>
                  <xsl:value-of select="'  '"/>
               </xsl:for-each>
               <xsl:for-each select="./value[not(@default='yes')]">
                  <xsl:value-of select="position()+1"/>
                  <xsl:value-of select="'  '"/>
                  <xsl:call-template name="information"/>
                  <xsl:value-of select="'  '"/>
               </xsl:for-each>
            </xsl:if>
         </prompt>
```

```
                <xsl:choose>
                    <!--_____choix multiple une selection -->
                    <xsl:when test="@type='choice'">
                        <!-- positionner sur entry en prenant dabord les choix par defaut -->
                        <xsl:apply-templates select="value[@default='yes']"/>
                        <xsl:apply-templates select="value[not(@default='yes')]"/>
                    </xsl:when>
                    <!--_____secret_____ -->
                    <xsl:when test="@type='secret'"/>
                </xsl:choose>
                <!-- positionner sur control avant de fermer le champs -->
                <xsl:choose>

                    <xsl:when test="(not(count(child::control)) and @sort='multiple') or
(not(count(child::control[@type=filled]))and @sort='multiple')">
                        <filled>
                            <xsl:comment>___<xsl:value-of select="@sort"/>____</xsl:comment>

<goto><xsl:attribute name="nextitem">new_choice <xsl:call-template
name="goto"><xsl:with-param name="position"><xsl:value-of se-
lect="position()"/></xsl:with-param></xsl:call-template></xsl:attribute></goto>




                            <!-- a completer par la suite pour les conditions -->
                        </filled>
                    </xsl:when>
<xsl:otherwise>
                        <xsl:comment> ____<xsl:value-of select="@sort"/>____</xsl:comment>
                        <xsl:apply-templates select="control">
                            <xsl:with-param name="rappel">
                                <xsl:for-each select="./value[@default='yes']">
                                    <xsl:value-of select="'  "/>
                                    <xsl:call-template name="information"/>
                                    <xsl:value-of select="'  "/>
                                    <xsl:value-of select="'tapez  "/>
                                    <xsl:value-of select="position()"/>
                                    <xsl:value-of select="'  "/>
                                </xsl:for-each>
                                <xsl:for-each select="./value[not(@default='yes')]">
                                    <xsl:value-of select="'  "/>
                                    <xsl:call-template name="information"/>
                                    <xsl:value-of select="'  "/>
                                    <xsl:value-of select="'tapez  "/>
```

```
                        <xsl:value-of select="position()+1"/>
                        <xsl:value-of select="'  '"/>
                    </xsl:for-each>
                </xsl:with-param>
                <xsl:with-param name="position">
                    <xsl:value-of select="position()"/>
                </xsl:with-param>
            </xsl:apply-templates>
        </xsl:otherwise>
    </xsl:choose>
    <!--_____________________________choix multiple plusieurs selections -->
    </field>
  </xsl:otherwise>
 </xsl:choose>
 <xsl:if test="@sort='multiple'">
  <xsl:call-template name="multiple"/>
 </xsl:if>
</xsl:template>
<!-- template value -->
<xsl:template match="value">
 <xsl:for-each select="."/>
 <!-- compte les noeuds defaut==yes-->
 <xsl:variable name="increment">
  <xsl:number value="(count(../value[@default='yes']))"/>
 </xsl:variable>
 <option>
  <xsl:choose>
   <xsl:when test="@default='yes' ">
    <xsl:attribute name="dtmf">
     <xsl:number value="position()"/>
    </xsl:attribute>
   </xsl:when>
   <xsl:otherwise>
    <xsl:attribute name="dtmf">
     <xsl:number value="position()+$increment"/>
    </xsl:attribute>
   </xsl:otherwise>
  </xsl:choose>
  <xsl:attribute name="value">
   <xsl:value-of select="."/>
  </xsl:attribute>
  <xsl:call-template name="information"/>
 </option>
</xsl:template>

<!-- template control -->
```

```
<xsl:template match="control">
  <xsl:param name="rappel"/>
  <xsl:param name="position"/>
  <xsl:choose>
    <!-- si le type n' est pas vide  -->
    <xsl:when test="(@type='nomatch' or @type='noinput')">
      <catch>
        <xsl:attribute name="event">
          <xsl:value-of select="@type"/>
        </xsl:attribute>
        <xsl:call-template name="information"/>
      </catch>
    </xsl:when>
    <xsl:when test="(@type='help')">
      <help>
        <xsl:call-template name="information"/>
        <xsl:if test="../@type='choice'">
          <xsl:value-of select="$rappel"/>
        </xsl:if>
      </help>
    </xsl:when>
    <xsl:when test="@type='filled'">
      <filled>
        <!-- a completer par la suite pour les conditions -->
        <xsl:if test="parent::entry[@sort='multiple']">
<goto><xsl:attribute name="nextitem">new_choice <xsl:call-template
name="goto"><xsl:with-param name="position"><xsl:value-of se-
lect="$position"/></xsl:with-param></xsl:call-template></xsl:attribute></goto>
        </xsl:if>
      </filled>
    </xsl:when>
    <xsl:when test="@type='grammar'">
      <xsl:if test="not(.='')">
        <gramar>
          <xsl:value-of select="."/>
        </gramar>
      </xsl:if>
      <xsl:apply-templates select="media[@lang=$lang]"/>
    </xsl:when>
  </xsl:choose>
</xsl:template>
<!-- template gramar -->
<xsl:template match="media">
  <gramar>
    <xsl:attribute name="type">
      <xsl:value-of select="@format"/>
```

```
        </xsl:attribute>
        <xsl:attribute name="src">
          <xsl:value-of select="@src"/>
        </xsl:attribute>
      </gramar>
   </xsl:template>
<!—Websphere n' acccepte pas de chiffre ds le nom des champs…remplacement par des lettres -->
    <xsl:template name="goto">
<xsl:param name="position"/>
    <xsl:choose>
 <xsl:when test="$position=1"><xsl:variable name="popo">_A</xsl:variable><xsl:value-of select="$popo"/></xsl:when>
<xsl:when test="$position=2"><xsl:variable name="popo">_B</xsl:variable><xsl:value-of select="$popo"/></xsl:when>
<xsl:when test="$position=3"><xsl:variable name="popo">_C</xsl:variable><xsl:value-of select="$popo"/></xsl:when>
<xsl:when test="$position=4"><xsl:variable name="popo">_D</xsl:variable><xsl:value-of select="$popo"/></xsl:when>
<xsl:when test="$position=5"><xsl:variable name="popo">_E</xsl:variable><xsl:value-of select="$popo"/></xsl:when>
<xsl:when test="$position=6"><xsl:variable name="popo">_F</xsl:variable><xsl:value-of select="$popo"/></xsl:when>
<xsl:when test="$position=7"><xsl:variable name="popo">_G</xsl:variable><xsl:value-of select="$popo"/></xsl:when>
<xsl:when test="$position=8"><xsl:variable name="popo">_H</xsl:variable><xsl:value-of select="$popo"/></xsl:when>
<xsl:when test="$position=9"><xsl:variable name="popo">_I</xsl:variable><xsl:value-of select="$popo"/></xsl:when>
<xsl:when test="$position=10"><xsl:variable name="popo">_J</xsl:variable><xsl:value-of select="$popo"/></xsl:when>
</xsl:choose>
       </xsl:template>
   <!--code de multiple-->
   <xsl:template name="multiple">
      <field>
<xsl:variable name="goto"><xsl:call-template name="goto">
<xsl:with-param name="position">
<xsl:value-of select="position()"/>
</xsl:with-param>
</xsl:call-template>
</xsl:variable>
        <xsl:attribute name="name">new_choice<xsl:value-of select="$goto"/></xsl:attribute>
        <prompt> (new_choice<xsl:value-of select="position()"/>) desirez vous ajouter un nouveau
```

```
            choix dans cette liste</prompt>
        <option dtmf="1" value="1">oui</option>
        <option dtmf="2" value="0">non </option>
        <filled>
          <if>
            <xsl:attribute name="cond">new_choice<xsl:value-of se-
lect="$goto"/>==1</xsl:attribute>
            <if cond="temp==0">
              <assign>
                <xsl:attribute name="expr">
                  <xsl:value-of select="@name"/>
                </xsl:attribute>
                <xsl:attribute name="name">
                  <xsl:value-of select="'temp'"/>
                </xsl:attribute>
              </assign>
              <else/>
              <assign>
                <xsl:attribute name="expr">temp +','+ <xsl:value-of select="@name"/>
                </xsl:attribute>
                <xsl:attribute name="name">
                  <xsl:value-of select="'temp'"/>
                </xsl:attribute>
              </assign>
            </if>
            <!-- Je vide le champs ce qui entrainera un retour sur choix_multi_multi pour
deuxiemme select-->
            <clear>
              <xsl:attribute name="namelist">
                <xsl:value-of select="@name"/>
              </xsl:attribute>
            </clear>
            <clear>
              <xsl:attribute name="namelist">new_choice<xsl:value-of select="$goto"/>
              </xsl:attribute>
            </clear>
            <prompt> la valeur de temp est <value expr="temp"/>
            </prompt>
            <else/>
            <if cond="temp==0">
              <else/>
              <assign>
<!—concatenation des valeurs-->
                <xsl:attribute name="expr">temp +','+new_choice<xsl:value-of se-
lect="$goto"/>
                </xsl:attribute>
```

```
                    <xsl:attribute name="name">new_choice<xsl:value-of select="$goto"/>
                    </xsl:attribute>
                  </assign>
                </if>
                <assign expr="0" name="temp"/>
              </if>
            </filled>
          </field>
        </xsl:template>
</xsl:stylesheet>
```

12 Notes.