République Algérienne Démocratique et Populaire.
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique.

Université Saad Dahlab, Blida
USDB.

Faculté des sciences.
Département informatique.

Mémoire pour l'obtention
d'un diplôme d'ingénieur d'état en informatique.
Option : Nom de l'option

Sujet :

Système d'échange de formulaires normalisés et signés

Présenté par : Mahiout Mehdi
Medjdoub Zoubir

Promoteur : Mr Ferfira
Encadreur : Mm Hessas

Organisme d'accueil : CERIST.

Soutenue le: 27/06/2005, devant le jury composé de :

Mr Hadj Yahya	**Président**
MlleAoussat	**Examinateur**
MmOukid	**Examinateur**

Dédicace

Ce mémoire est dédié :

A Nassima

A ma famille.

A mon binôme Zoubir.

A mes amis Abderezek, Chakib, Rachid, Nabil et Nassim.

Aussi a Mme Bensouna une personne a qui je dois beaucoup.

M.Mehdi

Dédicace

Ce mémoire est dédié :

A mes très chers parents pour leurs soutien durant toute ma carrière.

A mes deux grands parents.

A mes sœurs, Lina et Nawel.

A mon binôme Mehdi.

A son frère Mounir.

Ainsi qu'a toute sa famille.

A mes amis Nabil, Fatah, Samir, Chakib, Bachir et Sylvi.

Et a toute ma famille en particulier Djamel.

M.Zoubir Toufik

Résumé

L'échange d'information occupe une place très importante dans la vie courante. que ça soit dans le domaine administratif ou commercial.
Nous nous intéressant dans notre cas au problème posé principalement lors de la circulation des données entre le citoyen et son administration mais aussi aux échanges inter administrations.
Ce flux est composé en majorité de formulaires, ils représentent une partie importante de cet échange et restent un outil essentiel pour réaliser des applications interactives.
Cependant les outils d'édition de formulaires existant actuellement se rapprochent plus d'outils de programmation (langages de scripts) que de véritables outils d'édition.

- Dans le but de simplifier la tache au auteur de formulaires, notre travaille consiste a développer un système d'édition de formulaires, qui permet de créer facilement des formuliares.
Les formulaires crée sont sauvegarder sous forme (*.xforms ou *.xml avec *.xsl) qui seront visualiser par la suite par un module spécialisé ou par un navigateur web (*au future*).

- Ainsi pour garantir un climat de confiance lors du déroulement de ces transactions. Nous incluons notamment un support d'authentification numérique grâce a la norme XMLDSIG appuyée par son extension XAdES , en offrant a l'utilisateur la possibilité de signé une partie du formulaires (*ou tous le formulaire*),d'une manière simple et rapide et une validation draconiennes coté serveur .

Remerciements

Nous remercions d'abord ALLAH le tout puissant de nous avoir donné la force, la patience et la volonté pour achever ce travail.

Nous remercions nos parent respectifs pour leur patience leurs encouragement et leur soutien précieu tout au long de nos études.

Nous exprimons ici notre vive reconnaissance au personnel du CERIST, Nous pensons particulièrement à :

Madame A.ELMAOUHAB, nos encadreurs, Mme Hassess , Mr Kedjour et Mr Ferfera pour nous avoir encadré durant la réalisation de ce travail, et leurs disponibilité et leurs orientations tout au long de notre passage.

Nous tenons à remercier également les enseignants de l'Université de BLIDA. En particulier : Mlle Boustia, Mlle Aousat, , Mr Hadj Yahia, Mr AKKA , pour leurs présences aide et conseils.

Nous remercions les membres de jury pour nous avoir fait l'honneur de juger notre travail.

Nous remercions, de tout cœur, tous ceux qui ont contribué de près ou de loin à la réalisation de ce modeste travail.

Table des matières

Chapitre 2: XForms

Liste des Abréviations et acronymes

WWW	World Wide Web
W3C	World Wide Web Consortium
XML	eXtensible Markup Language
XFA	Xml Forms Architecture
XFDL	Xml Forms Description Language
XForms	Xml Forms
SGML	Standard Generalized Markup Language
HTML	HyperText Markup Language.
XSL	Extensible Stylesheet Language.
XSLT	Extensible Stylesheet Language Tronsformation.
XSL-FO	XSL Formatting Object.
X-Path	Xml Path.
XHTML	Extensible Hypertext Markup Language.
XMLDSIG	Xml Digital Signature.
XAdSIG	Xml Advanced Signature.
MVC	*Modèle, Vue, Contrôleur.*
UIC	User Interface Control.
SSS	Secure Socket Layer.

Table des figures

Introduction

De quoi se compose t'il ? [Int03]

Les systèmes de formulaires se composent habituellement de deux modules

1. Module de conception du formulaire:

   On recourt à ce module pour :

   - Agencer la disposition du formulaire *(Création du modèle)*.
   - Etablir un croquis (tracé) du formulaire à l'écran.
   - Définir chacun des champs du formulaire et leurs attributs respectifs.
   - Permet de signer numériquement soit tous le formulaire ou bien spécifier le champ voulu.

2. Module de préparation :

   À la lecture de la définition du formulaire et de la feuille de style fournie par le module de conception à celui de la préparation, ce dernier permet a l'utilisateur de :

   - Consulter le formulaire a l'écran.
   - Saisir les données dans les champs.
   - Validation des données en temps réel
   - Effectuer des calculs à partir des données saisies par l'utilisateur.

   (*Par exemple, dans le cas d'un champ «date», la fonction de préparation ne permet que la saisie de la date pertinente*)

   - Sauvegarde des informations saisies par l'utilisateur dans une base de donnée.

Comme son nom l'indique, un système d'échange de formulaires doit favoriser l'échange des instances et de l'information sur la définition des données de formulaires produite par le module de conception pour maintenir une structure ouverte. C'est pour cela que les formulaires générés doivent être normalisés. *(N'importe quelle application utilisant la même norme peut visualiser et traiter notre formulaire)*

Avantages de la gestion de formulaires:

L'utilisation des formulaires réside dans deux domaines, administratif et commercial. Dans notre cas, nous nous intéressant au domaine administratif.

L'informatisation de l'ensemble des traitements liés aux formulaires administratifs va permettre de bénéficier des avantages suivants :

- **Traitements plus rapides des données:** Puisque les formulaires sont envoyés à travers le système d'information par leur saisie directement sur l'ordinateur.

- **Gestion améliorée des formulaires:** Elimination des problèmes de gestion de version du formulaire, de stocks de papier à terminer, etc...

- **Grande facilité de saisie:** Puisque les champs de saisie sont clairement identifiés, des aides en ligne peuvent être créées pour assister l'Utilisateur, etc... Il est également possible de faire des contrôles en temps réel, permettant d'éviter les erreurs de saisie.

- **Grande facilité de validation par les supérieurs hiérarchiques :** Puisque d'une part le formulaire arrivera directement sur leur poste, sous une présentation homogène et leur permettant de valider rapidement la demande.

- **Retour d'information immédiat à l'émetteur :** L'émetteur de la demande peut tout de suite être averti dès la validation par les personnes adéquates.

- **Absence de risque de pertes des données :** Elles sont directement stockées dans des bases de données qui s'inscrivent automatiquement dans des procédures de sauvegardes centralisées par les exploitants du système informatique.

La majorité de flux échanger dans notre système ce fait par le biais des formulaires électronique

Formulaires électroniques :

Les formulaires Web représentent une des méthodes, des moyens pour les pages Web d'interagir, échanger des informations, communiquer avec les utilisateurs.
Les premiers formulaires HTML sont apparus sur le Web vers le 1993, et jusqu'à maintenant ils n'ont pas eu de gros changements, mais par contre son importance est de plus en plus augmenté.
Les formulaires permettent aux utilisateurs la gestion des comptes bancaires, l'achat des billets d'avion ou plus simplement ils servent pour envoyer au site un feedback, un commentaire ou une critique.

Introduction générale:

Aujourd'hui plus que jamais, les entreprises investissent dans des solutions logicielles leur permettant de réduire leurs coûts, d'augmenter leur productivité et d'améliorer leur service à la clientèle.

Système d'échange de formulaires: [Int01]

L'apport de la gestion de formulaires en terme "administratif" au sein d'une entreprise est en général grandement apprécié des utilisateurs et un des gisements importants de productivité. Cette gestion est établit par un système d'échange de formulaires.

Que ce qu'un système d'échange de formulaires?

Un système d'échange de formulaire est une très puissante application qui permet aux équipes et entreprise de rassembler efficacement l'informations dont il ont besoin, via des riches formulaires dynamique.

Que fait-il? [Int02]

-Il permet de créer facilement et rapidement des formulaires Web et ne requiert aucune connaissance technique. La création d'un nouveau formulaire se fait en un simple clic. Des champs sont ensuite ajoutés et paramétrés, définissant ainsi la structure du formulaire. Le Gestionnaire de formulaires se charge de créer et de gérer pour vous la base de données. Il peut également générer des statistiques sur les données saisies.

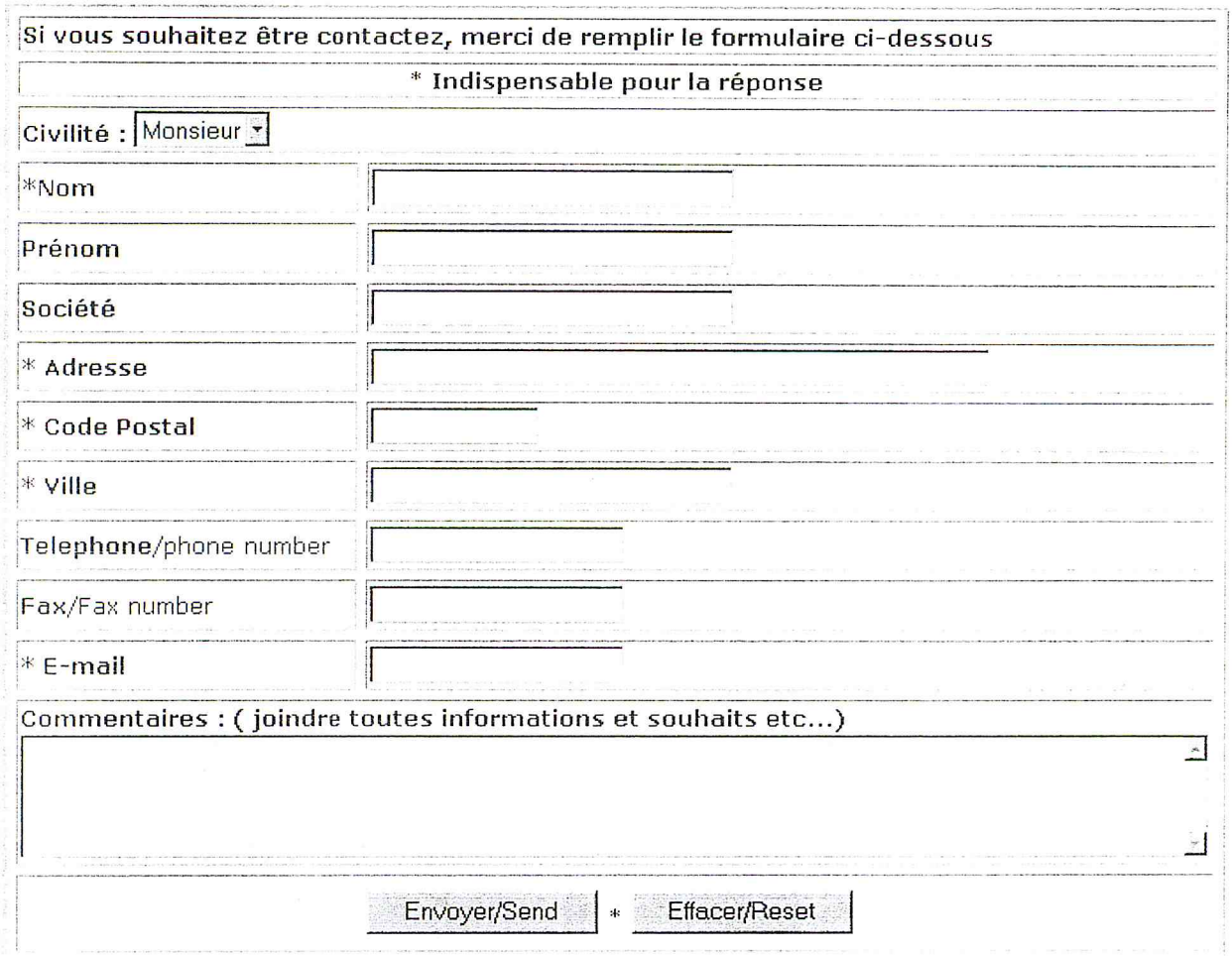
-Il inclut notamment un support d'authentification numérique de formulaires.

-Et se charge aussi d'envoyer les formulaires à travers un réseau.

A qui s'adresse t'il ?

Il s'adresse aux équipes et organisations amenées à collecter et utiliser de l'information pour accomplir leurs tâches. C'est un outil précieux pour tout utilisateur professionnel ou organisation en quête d'une méthode efficace et flexible de collecte d'informations qu'il faudra ensuite pouvoir standardiser, valider et intégrer aux serveurs d'entreprise.

Voici un exemple :


Si vous souhaitez être contactez, merci de remplir le formulaire ci-dessous

* Indispensable pour la réponse

Civilité : Monsieur

*Nom

Prénom

Société

* Adresse

* Code Postal

* Ville

Telephone/phone number

Fax/Fax number

* E-mail

Commentaires : (joindre toutes informations et souhaits etc...)

Envoyer/Send * Effacer/Reset

Figure 1 : *Exemple d'un formulaire électronique*

En plus d'avoir une représentation informatique des formulaires et d'en permettre donc une saisie guidée sur ordinateur, une gestion de formulaires électroniques va permettre également d'en assurer le suivi à travers une automatisation des processus de validation.

Dans le cas par exemple d'une demande de congés, une fois le formulaire rempli par l'émetteur de la demande, il doit être validé par le supérieur hiérarchique. Dans le cadre d'une gestion informatisée, il est alors tout à fait possible, dès la fin de la saisie du formulaire, de le communiquer immédiatement par le biais de la messagerie au supérieur hiérarchique, qui recevra ainsi dans sa boite aux lettres l'ensemble des demandes de ses subordonnés devant être validées. Et de même, lors de la validation *(positive ou négative)*, l'émetteur initial de la

demande pourra immédiatement être prévenu de la réponse de son supérieur hiérarchique. L'utilisation des formulaires électroniques va donc permettre de canaliser une information qui est en général plutôt dispersée, qui représente un nombre de manipulations manuelles importantes, qui n'est pas à proprement parler un travail particulièrement gratifiant mais plutôt perçu par l'ensemble des salariés d'une entreprise comme une perte de temps, bien qu'il soit nécessaire à la bonne marche de cette même entreprise.

Cependant il existe plusieurs normes utilisées pour la création des formulaires électroniques, en vue de permettre l'échange de données des formulaires électroniques entre les applications de divers fournisseurs.

Définissons quelques normes existantes:

1) **SGML**: *(Standard Generalized Markup Language) (ISO 8879)* [Int04]
Approuvée par l'ISO (Organisation internationale de normalisation) en 1986, SGML est un langage de codage de données dont l'objectif est de permettre, dans un échange entre systèmes informatiques, de transférer, en même temps, des données textuelles et leurs structures. Cette norme définit une syntaxe permettant de préciser des types de documents *(DTD)*, et permet de définir et de valider une série d'étiquettes pour un certain genre de documents. Appliquée aux formulaires électroniques, le SGML peut servir à définir les règles et la syntaxe de présentation du formulaire, les modèles et les instances à échanger entre les applications et les utilisateurs.

Quelques applications *(sous-normes)* appartenant a SGML :
* HTML *(Hyper text markup Language) (langage de balisage hypertexte)* - Langage de balisage de document destiné au Web. HTML est une application de SGML qui définit une série d'étiquettes que les navigateurs Web, comme Mosaic et Netscape, peuvent formater. Il comprend également un mécanisme universel de liaison avec les hyperliens, d'un endroit à l'autre dans un même document, à l'aide d'un serveur Web se trouvant quelque part dans le monde.
* CALS *(initiative d'échange électronique militaire).*

* J2008 *(pour les documents techniques dans le domaine de l'automobile).*

2) **XML**: *(eXtended Markup Language) Sous ensemble du SGML,* comprenant les sous normes suivantes *:*

1998, date à laquelle le W3C donnait aux spécifications XML le statut de recommandation. Leur adoption par les principaux acteurs du marché de l'Internet a conduit à la création d'une pléthore de « standards » XML.

Ce succès prouve la capacité d'adaptation de cette spécification mais se révèle être une source de confusion *(parfois entretenue par certains acteurs compte tenu des enjeux de l'internet).*

Il y a trois catégories de produits XML :

· **La grammaire XML:** XML est un langage qui permet de présenter l'information encadrée par des balises. Ces balises peuvent être définies par le concepteur d'une application ou l'architecte d'un système d'information *(c'est là le caractère* ***extensible*** *de ce langage).* Ces balises décrivent les informations qui seront échangées entre les applications et les systèmes d'informations à travers Internet ou les intranets.

· **Les protocoles fondés sur XML:** Les documents XML et les messages structurés XML peuvent être facilement transmis en s'appuyant sur des protocoles standards *(tels que HTTP ou FTP).*

· **Les vocabulaires XML:** La mise en place d'une grammaire commune et de protocoles d'échange communs n'exonère pas les acteurs de la définition d'une représentation commune de l'information.

On entre alors dans le domaine des vocabulaires XML.

C'est dans ce domaine que de nombreuses initiatives visant à s'approprier la représentation de l'information sont apparues, avec notamment l'élaboration de répertoires et de vocabulaires spécifiques à divers secteurs d'activités.

Un vocabulaire XML spécifie donc comment utiliser XML pour une application particulière ou pour un secteur d'activité donné.

Par exemple, le secteur de la chimie a défini CML, qui est une extension des spécifications XML, pour décrire les données du secteur chimique. De nombreux autres exemples existent, en particulier, dans le domaine financier.

-Quelques vocabulaires (spécifications) XML touchant le domaine de Formulaires Electronique: [Int05]

* **XForms** *(XML Forms):* Spécification dont l'élaboration a été entamée par le W3C en 1999 , et est destinée aux futur générations de sites web, Représentant des formulaires destinés à être intégrés dans un langage de description de documents *(XHTML,SVG,...)*

* **XFDL** *(XML Form Description Language:* Conçu pour le e-commerce par Pure Edge. XFDL est une syntaxe du UFDL *(Universal Form Description Language)* exprimée en XML et utilisant des données de XForms. Un Formulaire XFDL pourra supporter la signature numérique et être conserver intégralement pour fin de preuve légale de transaction. Il est actuellement a l'état de note auprès W3C depuis le 16 octobre1998.

* **XFA** *(XML Form Architecture):* Proposé à la normalisation W3C par JetForm Corp *(Firme basée a Ottawa)*. A la différence des formulaires (X)HTML, la spécification XFA prévoit un rendu visuel spécifique, et comporte tous les éléments syntaxiques associés à une telle proposition : sélection de polices *(comme <FONT Typeface="..." />)*, définition de bordures, de tracés, ect. Il est actuellement a l'état de note auprès W3C depuis le 14 juin 1999.

Quelle norme choisir ? [Int06]

Comparaison des différents standards :

XML Vs SGML:

Concernant le domaine documentaire : le succès de XML permet d'envisager une généralisation de son usage dans le domaine documentaire.

La question de la migration est d'ordre stratégique et économique.

D'un point de vue plus stratégique, beaucoup se posent la question de passer de SGML à XML, en raison des organismes de normalisation qui s'en occupent. Pour SGML, c'est l'ISO qui assure la normalisation, avec ses processus de maturation d'une norme et ses représentations nationales assurant la qualité des normes ; en ce qui concerne la normalisation de XML, c'est le travail du W3C, un Consortium qui n'est représentatif que de lui-même et qui, comme son nom l'indique, ne s'intéresse qu'au point de vue du Web. Cet argument devrait rapidement être balayé par la volonté commune des deux organismes de promouvoir XML au sein des normes documentaires de l'ISO.

Économique, parce que l'offre de service et les outils disponibles suivant les recommandations XML du W3C sont plus importantes que pour SGML. Fatalement, les coûts de développement et de maintenance d'une application seront moins élevés en XML qu'en SGML. En outre, la compatibilité XML-SGML s'estompe avec la notion de schéma et de *namespaces* qui ne sont pas dans la norme Iso. Il semble donc préférable de privilégier le développement en XML dans ce domaine.

De plus les DTD du XML sont nettement moins lourdes que celles du SGML, qui mélangeaient deux type d'information : la notion de modèle de donnée et le format de codage des documents conformes à ce modèle.

Un document SGML doit être valide pour être lu *(respectant une grammaire précise, qui est conforme à la DTD)*. Tandis-que XML ne requière pas cette formalité, le document doit être simplement bien formé *(respectant la syntaxe XML)*.

D'un point de vue technique, ce format peut sans problème être remplacé par XML : tout ce qui est nécessaire à SGML existe à peu près dans XML.

D'un point de vue pratique, les utilisateurs de SGML y gagneront en diversité d'outils utilisables et apprécieront à son juste avantage la prise en compte par XML d'Unicode XML.

Résultat:

L'introduction de XML dans l'entreprise semble inéluctable, que ce soit en périphérie des systèmes d'information, pour des échanges en interne ou avec des partenaires.

La séparation entre le contenu, sa structure et la présentation d'un document XML en fait le format d'échange universel de données intra et interentreprises.

Outre les applications dans le domaine de l'échange, la structuration de l'information et la gestion des méta-données induite par XML offrent des perspectives importantes dans le domaine de la gestion de la connaissance. Par ailleurs elle est de nature à simplifier la gestion de la conservation des documents électroniques.

Donc il serait préférable de développer notre application avec une spécification XML.

Mais cependant:

Quelle spécification choisir ?

Notre choix se penchera vers le XForms, tous simplement parce que XForms à été reconnu comme norme par le W3C, alors que XFDL et XFA sont toujours en cour validation. En plus à la différence de XFA, XForms sépare la présentation de la structure par une organisation en trois couches: données, logique, et présentation, et permet pour la présentation l'utilisation de technologie déjà existantes comme XSL et CSS.

Caractéristiques de XForms :

-Séparation entre les contrôles du formulaire et les données qu'ils collectent.

-Séparation contenu/présentation, contrôles de haut niveau : multi plates-formes, multi modalité.

-Typage fort des données permettant un contrôle de saisie local.

-Calcul et validation des données côté client *(expressions XPath)*.

-Données soumises sous la forme d'une instance XML.

-Nombreux événements et *handlers* prédéfinis : pas besoin de scripts.

- Indépendance des Terminaux.

La Signatures numériques : [Int03]

Il est essentiel que l'information contenue dans un formulaire électronique parvienne intégralement à son destinataire et qu'elle ne soit pas altérée. En d'autres termes, il faut être en mesure de vérifier que les données échangées que reçoit le destinataire correspondent exactement à celles transmises par l'expéditeur.

Les cinq caractéristiques ci-après définissent les exigences essentielles à la sécurité du réseau :

- confidentialité : permet de s'assurer que les données ne sont ni divulguées à quiconque n'est pas autorisées à les recevoir, ni transmises à des ordinateurs non autorisés;
- Contrôle d'accès : Permet de s'assurer que l'accès aux données sera réservé à ceux qui sont autorisés à les consulter ou à les modifier;
- Intégrité : permet de s'assurer que les données n'ont pas été altérées depuis leur création ou leur transmission;
- Authentification de l'origine des données : Permet de certifier la source des données;
- non-répudiation : permet de s'assurer que personne ne peut nier être parti d'une opération électronique à laquelle il a participé.

Le Module XML qui s'occupe des signatures numériques est la recommandation XMLDSIG appuyée par son extension XAdES de l'IETF/W3C.

Plan du Mémoire:

Ce mémoire est organisé en quatre chapitres:

Chapitre 1: Dans ce chapitre nous donnerons un aperçu sur les langages de balisage XML, XSLT et XHTML.

Chapitre 2: Ce chapitre sera dédié à l'étude de la norme de création de formulaires, XForms.

Chapitre 3: Ce chapitre sera consacré à l'étude de la signature électronique, il est divisé en trois parties:Dans la première partie nous donnerons des définitions générales concernant la signature digitale *(clés, certificats,...)*.Dans la deuxième partie nous donnerons la syntaxe et la sémantique de la signature XML (XMLDsig).Dans la troisième partie nous aborderons une extension de XMLDsig : XAdES qui résous le problème de répudiation.

Chapitre 4: Nous aborderons dans ce chapitre la démarche de développement de notre système.

Chapitre 1 :

XML XSLT et XHTML

I. XML :

I.1. Introduction à XML: [XML01]

Le Word Wide Web a considérablement transformé notre vie quotidienne, et comme sans doute beaucoup de gens le savent le langage du Web est le HTML (*Hyper Text Markup Language*).

Un langage de marquage est une technique nous permettant d'indiquer la manière dont on souhaite présenter un contenue, un tel langage repose sur des balises, délimitées par les caractères < et > précisant certains critères.

Mais le HTML a un inconvénient, il a été conçu pour décrire les documents Web, de manière que les navigateurs sachent comment les afficher. Lorsque nous concevons un document en HTML nous gênerons du même coup sa mise en forme sans ce soucier d'autre chose, toutes les balise d'une page HTML sont donc relatives à sa présentation finale et rien d'autre, rien ne permet a un logiciel de connaître la sémantique (*sens*) du texte.

Le XML apporte une solution à ce problème, le contenu est nettement distingué de sa mise en forme.

Contrairement a document HTML un document XML se doit d'être «Bien-formé» pour pouvoir être interprété (*respecte la syntaxe XML*), cela signifie notamment qu'une balise ouverte doit être fermé et que certaines imbrications ne sont pas admissibles

Une ligne comme la suivante n'est pas correcte, même en HTML, mais le navigateur n'y trouve rien à redire :

```
<b>L'université <i> de</b>Blida</i>
```

Il en irait bien autrement avec du XML.

I.2. Definition :

XML (*eXtensible Markup Language*) ou Langage de balisage extensible, est un nouveau langage normalisé de balisage numérique, permettant de décrire la structure hiérarchique d'un document, c'est solution incontournable pour la représentation et l'échange de documents sur Internet.

Ainsi, XML est devenu le langage universel d'échange de données informatiques, qu'il s'agisse de les stocker, de les échanger, de les traiter ou de les afficher.

I.3. Avantages du XML :

- Indépendance de données :Avec le XML, les données ne dépendent plus d'une application spécifique pour leur création, leur affichage ou leur édition.
  Il est au données ce que Java représente pour les applications. Il permet que les données soit utilisables par n'importe qu'elle application.
- Une seule source de donnes et plusieurs présentations possibles : Le formatage des données au moyen d'un langage de balisage permet aux applications informatiques de traiter et de présenter ces données de différentes manières, ceci grâce a la séparation du contenu de sa mise en forme.
- Un support pour les transactions commerciales sur Internet : Pour que les transactions commerciales puissent être réalisées sur Internet, on a besoin d'échanger des données en temps réel, Le XML offre le support dont a besoin ce nouveau model économique.
- Amélioration des recherches de données : Puisque le XML prend en compte le type de données et non simplement le contenu, il est possible d'améliorer considérablement notre capacité a trouver des informations pertinentes. A l'heure actuelle la recherche de document repose sur les correspondances de mots clés dans les contenus. Un monteur de recherche intelligent qui prend en compte les dialectes XML peut s'intéresser tant au contenu qu'aux Meta-données.
- Possibilité d'accéder aux données de n'importe quel périphérique d'affichage : Tout comme les données d'un document XML peuvent être affichées de différentes manières, il est possible de reformater automatiquement un document XML selon le périphérique d'affichage.
- Simplification du développement des applications : Les application n'auront plus besoins d'importer ou d'exporter des centaines de formats de données propriétaires.

I.4. Les Règles d'écriture :

- Un document XML est sensible à la casse.
- Comme en HTML, les espaces, tabulations, retours et sauts à la ligne sont considérés comme des séparateurs et sont générés par le Parseur (*l'Analyseur*) sous forme d'espace simple.
- Un nom (*Token*) commence par une lettre et peut contenir : Lettre, Chiffre, (point), (tiret), '_' (*tiret souligné*), ':' (*deux points*). (*Tous les mots commençant par XML ou xml sont réservés*).

I.5. Les Composants du XML :

Comme nous l'avons expliqué plus haut l'XML est une formalisation de règles pour baliser des documents au moyen de Meta-données qui apportent des informations supplémentaires (*la fonction des données*) à l'utilisateur. Toutes les balises commencent soit par une esperluette (&), soit par le signe inférieur a (<).

Il Existe six types de balisage en XML :

a) **Les éléments** : Les éléments sont les composants les plus habituels des langages des balisages, c'est un composant logique d'un document, il est composé de balises d'ouverture et de fermeture en encadrant un contenu, d'autres éléments ou les deux à la fois.

Les balises d'un éléments sont délimitées par le signe inférieur a (<) et supérieur a (>).

Par exemple :

```
<université> Saad dahleb </université>
```

b) **Les attributs** : un élément peut avoir des attributs spécifiés par la paire nom/valeur placé après le nom de la balise d'ouverture , en reprenant l'exemple précédant :

```
<université  ville= "Blida"> Saad dahleb </université>
```

c) **Les commentaires** : Un commentaire est une description qui sera ignorée par le processeur XML, il est compris entre <!-- et -->.

d) **Les instructions de traitements** : Les instructions de traitements permettent de passer des informations a l'application de traitement.

e) Les références a des entités : Les références a des entités sont utilisées pour insérer des caractères réservés ou des abréviations dans le balisages, ce sont comme une boite pourvue d'étiquette . Elles sont déclarées dans la définition du type de document (*Nous y reviendrons dans « Structure de document XML »*).

e.1) Entité générale :

Une entité utilisée seulement dans le contenue du document elle se déclare comme suite :

```
<ENTITY nom entite "valeur">
```

e.2) Entité analysée : Le contenu de la boite est du texte pur.

e.3) Entité externe :

Une Entité obtenue a partir d'une source externe, cette source externe est spécifié par un URL. Elle est déclarée de la manière suivante :

```
<ENTITY nom SYSTEM "url">   (identifiée par le mot clé SYSTEM)
```

f) Les sections CDATA : [XML02]

Une section CDATA est utilisée dans un document afin que le contenu ne soit pas traité comme balisage. Une section CDATA commence par la chaîne `<![CDATA[` et se termine par la chaîne `]]>` .

Exemple :

```
<![CDATA[ <TITLE> un exemple XML </TITLE> ]]>
```

`<TITLE>` ne sera pas traitée comme étant une balise mais comme une simple chaîne de caractères.

I.6. Structure d'un document XML :

Il existe deux sorts de documents XML, le document bien formé et le document valide.

- Un document bien formé est un document adhérent a la syntaxe du XML.
- Un document valide est un document bien formé qui respecte une grammaire définie dans une DTD (*que nous aborderons plus loin dans ce chapitre*).

Un document XML se compose de :

- Un prologue (*facultatif*).
- Un arbre d'éléments qui constitue le contenu du document.
- Des commentaires et des instructions de traitement.

I.6.1 Le prologue :

Le prologue des documents XML joue trois rôles importants :

a) Préciser qu'il s'agit d'un document XML (*déclaration XML*).

b) Identifier le jeu de caractères utilisé.

c) Identifier la grammaire utilisée (*DTD*).

Comme il a été signalé plus haut ces trois éléments sont facultatifs, mais il est généralement préférable d'inclure les déclarations représentant les deux premiers informations (*a et b*).

* **La déclaration XML :**

- La déclaration du document XML doit être nécessairement au début du document, voici la forme minimale de cette déclaration :

```
<?xml version= "1.0" ?>   avec la version du langage utilisée.
```

* **Identification du jeu de caractères :**

- L'option de la déclaration de l'encodage est utilisée pour spécifier le codage des caractères utilisés.

- Par défaut, il s'agit d'Unicode encodé au format UTF-8, exemple de cette déclaration :

```
<?xml version= « 1.0 » encoding= "ISO-8859-1"?>
```

- La déclaration standalone est rarement utilisée, mais elle précise si une application de traitement peut s'épargner le chargement de la définition du type de document (*DTD*), pour traiter le document.

* **La définition du type de document :**

Les éléments et les attributs donnent des indications sur la structure ou la fonction du contenu.

La DTD (Document Type Desfontaines) Déclare tous les éléments légaux d'un document, les attributs légaux que peuvent avoir ces éléments ainsi que la hiérarchie. Pour qu'un document soit valide celui-ci doit indiquer la DTD a laquelle il se rattache.

Toutes les déclarations DTD commence par la chaîne littérale `<!DOCTYPE`, le mot qui suit est le nom de la racine identifiant l'élément qui contient tous les autre dans le document.

La DTD peut être référencée de la manière suivante :

```
<!DOCTYPE racine PUBLIC "identificateur Public" "url de  la DTD">
```

I.6.2 L'arbre d'éléments :

Les éléments sont les objets les plus importants d'un document XML, les documents XML sont des hiérarchies strictes d'éléments, cela veut dire qu'il existe un et un seul élément supérieur qui contient tous les autres et qu'on appel racine.

Exemple:

```
< ?xml version='1.0' ?>
<LIVRES>
<LIVRE>
          <TITLE>Croc Blanc</TITLE>
          <AUTEUR>Jack London</AUTEUR>
    <PUBLISHER>EDIFRANCE</PUBLISHER>
</LIVRE>
<LIVRE>
           <TITLE>Roméo et Juliette</TITLE>
     <AUTEUR>Willam Scheckspear</AUTEUR>
           <PUBLISHER>Eyrolles</PUBLISHER>
</LIVRE>
</LIVRES>
```

I.6.3Les commentaires et des instructions de traitement :

(*Partie déjà expliqués en* **I.5.c** *et* **I.5.d**)

Remarque importante :

En exécutent un document XML dans un browser supportant XML tel qu'il a été montré Précédemment, le fichier sera affiché tel quel.

Pour générer l'interface souhaitée, le document doit être accompagné d'une feuille de style (XSL).

II.XSLT :

II.1. Introduction XSLT (eXtended Stylesheet Language Transformation):

Il serait incommode d'expliquer ce que c'est qu'une XSLT sans développer XSL, car l'XSLT en est une extension.

Comme vue précédemment XML sépare contenu et présentation. Nous attaquant cette fois-ci la partie présentation. Un document XML contient de l'information pure sans se soucier de la mise en forme, ceci dit comment organiser ces informations ?

Et bien en associant tous simplement a notre document une feuille de style enregistré en: ".xsl".

II.2. XSL (eXtended Stylesheet Language): XSL est une famille de spécifications comprenant : [XS01]

- XSLT (*XSL Transformation*) pour le parcours et la transformation d'un document source XML vers un autre document XML.
- X-Path pour la navigation dans l'arbre XML source,
- XSL-FO (*XSL Formatting objects*) pour décrire la présentation d'un document (*ne sera pas développée*).

Au départ, il n'y avait pas cette séparation, et c'est pourquoi le terme générique XSL désigne aussi bien la phase de transformation que celle de formatage.

-La feuille XSL est donc la feuille de style associée au document XML. Elle contient les règles de transformation, de présentation et de mise en forme des données.
-Le XSL ne sert pas uniquement de langage de feuille de style, il est aussi un très puissant manipulateur d'éléments. Il permet de transformer un document XML source en un autre, lui permettant ainsi des modifications de structure.
-Le fonctionnement du XSL est basé sur les manipulations de modèles (*templates*). Un modèle contient le texte (*éléments, attributs, texte...*) de remplacement d'un élément donné.

Voici un exemple pour associer une feuille de style XSL a un document XML:

```
<?xml version="1.0" encoding="utf-8" ?>
<?xml-stylesheet type="text/xsl" href="catalog.xsl"?>
<catalogue>

<PRODUIT NOM="T-shirt" TAILLE="XL" COULEUR="BLANC" />
<PRODUIT NOM="Chemise" TAILLE="38" COULEUR="BLEUE" />
<PRODUIT NOM="T-shirt" TAILLE="LL" COULEUR="ROUGE" />
<PRODUIT NOM="Chemise" TAILLE="40" COULEUR="JAUNE" />
 </catalogue>
```

Cette méthode d'association s'appelle la méthode directe, elle consiste à lier le document XML à la feuille de style XSL par l'intermédiaire de la "*processing instruction*" suivante :

```
<?xml-stylesheet type="text/xsl" href="noteFeuille.xsl"?>
```

Il existe d'autres méthodes qu'on peut utiliser mais qu'on ne développera pas:

Citons quelques exemples:

- Transformation explicite par code : dans un fichier ou un script exécutable par le serveur.
- Complément sur le DOM : manipulation avec un script dans une page HTML.

II.3.Structure d'un document XSL: [XS03]

Une feuille de style XSL est un document XML qui commence par l'en-tête XML ayant pour élément racine : <xsl:stylesheet> ... </xsl:stylesheet>

La structure de base d'un document XSL comprend un *prologue* ainsi qu'un élément `<xsl:stylesheet` pouvant contenir quelques attributs, notamment une déclaration d'espace de noms ainsi que le numéro de version.

II.3.1 Le prologue: Contient la déclaration XML indiquant la version et le type d'encodage utilisé :

```
<?xml version="1.0" encoding="ISO-8859-1"?>
```

II.3.2 L'appel a l'element <xsl:stylesheet> : Il Se fait par l'instruction suivante:

```
<xsl:stylesheet id="id" version="number" xmlns:pre="URI">
```

id: (*facultatif*) est l'identifiant unique de la feuille de style.

version: est le numéro de version de la feuille de style XSL/XSLT.

pre: indique le préfixe qui sera utilisé dans la feuille de style pour faire référence à l'URI.

L'URI est très important, au fait on doit l'insérer même si il n y a rien au bout, car c'est elle qui signale au parseur avec quel type de document XSL en travaille.

Exemples:

```
<xsl:stylesheet version="1.0" xmlns:xsl="uri:xsl">
```

... permet d'avoir accès uniquement à des fonctions de base.

```
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
```

... permet d'avoir accès à des fonctions évoluées d'XSLT.

II.3.3 Les fondamentaux:

1) **<xsl:output>:** Cet élément, à placer comme premier enfant de `<xsl:stylesheet>`, permet de spécifier des options concernant l'arbre de sortie. L'utilisation de cet élément est de la forme :

```
<xsl:output method="xml"|"html"|"text" version="nmtoken" encoding="chaine"
standalone="yes"|"no" doctype-public="chaine" doctype-system="chaine"
media-type="chaine"/>
```

Il y a certains attributs qui n'ont pas été développés ou alors très brièvement, comme par exemple l'attribut `cdata-section-elements` qui indique les éléments dont le contenu doit être traité lors de la transformation via une section `CDATA`.

- **`method`** identifie la méthode de transformation. Dans le cas où elle serait égale à `text`, aucune mise en forme n'est effectuée.
- **`version`** identifie la version de la méthode de sortie (*xml 1.0, html 4.01...*).
- **`encoding`** indique la version du jeu de caractères à utiliser pour la sortie.
- **`standalone`** indique au processeur XSLT s'il doit créer un arbre de sortie avec ou sans déclaration de type de document.
- **`doctype-public`** indique l'identifiant public utilisé par la `DTD` associée à la transformation.
- **`doctype-system`** indique l'identifiant system utilisé par la `DTD` associée à la transformation.
- **`media-type`** indique le type MIME des données résultantes de la transformation.

2) **<xsl:template>**: Cet élément définit un model a appliquer a un nœud et a un contexte spécifique.

L'utilisation de cet élément et de la forme suivante:

```
<xsl:template name="nommodele" match="expression_match" mode="modemodele">
```

- **name**: correspond au nom associé au modèle.
- **Match**: indique quel jeu de nœuds sera affecté par le modèle. Cette expression peut comprendre un test d'existence d'attribut, le caractère | indiquant que le modèle s'applique à un élément ou à un autre, ainsi que tout élément permettant de définir un jeu d'attributs.
- **mode** : permet à un élément d'avoir plusieurs modèles, chacun générant une sortie différente.

Voici quelques valeurs que peut prendre "match":

/	racine du document
repertoire	tous les elements <repertoire>
*	tous les nœuds de type elements
repertoir fichier	tous les elements <repertoirs> ou <fichier>
racine/fichier	tous les elements <fichier> avec <racine> comme parent
repertoire//fichier	tous les elements <fichier> avec <racine> comme ancetre
@nom	l'attribut nom du nœud (element) courant
fichier/@nom	l'attribut nom de l'element fichier
fichier[@nom='Zizou']	les element <fichier> dont l'attribut nom vaut 'Zizou'
comment()	les commentaires
texte()	les nœuds texte
node()	les nœuds autres que racine

3) **<xsl:apply-templates>** : Cet élément permet de descendre dans l'arbre, et préciser au processeur de traiter les éléments fils du nœud courant avec lesquels le traitement continue.

4) **<xsl:value-of>:** Cet élément permet d'insérer la valeur d'un nœud dans la transformation. Ce nœud est évalué en fonction d'une expression. Cette expression peut correspondre a un élément, a un attribut ou a tout autre nœud contenant une valeur. L'utilisation de cet élément est de la forme:

```
<xsl:value-of select="expression"/>
```

L'attribut `disable-output-escaping` ***a été omis***).

La valeur de **`select`** est évaluée, et c'est cette valeur qui sera insérée dans la transformation.

II.3.4 Ajout d'éléments et d'attributs:

1) **<xsl:element>** : Cet élément insert un nouvel élément dans la transformation. Le nom de l'élément est spécifié par l'attribut `name`. L'utilisation de cet élément est de la forme:

```
<xsl:element name="nom_element" use-attribute-sets="jeu_attributs">
```

Exemple:

```
<xsl:element name="livre"><xsl:value-of select="titre"/></xsl:element>
```

...permet de créer dans le fichier `HTML` un élément "livre", renfermant le contenu de l'élément `"titre"` du document `XML`.

2) **<xsl:attribut>** : Cet élément définit un attribut et l'ajoute à l'élément résultat de la transformation. L'utilisation de cet élément est de la forme :

```
<xsl:attribut name="nom">valeur</xsl:attribut>
```

Exemple:

```
<image><xsl:attribut name="src">maphoto.gif</xsl:attribut></image>
```

... permet d'ajouter à l'élément `image` l'attribut `src` et de lui affecter la valeur `maphoto.gif`, ce qui a pour effet de produire en sortie l'élément suivant : `<image src="test.gif"></image>`

II.3.5. Gestion des boucles:

1) **<xsl:for-each>** : Cet élément crée une boucle dans laquelle est appliquée un certains nombres des transformations. Son utilisation est de la forme:

```
<xsl:for-each select="jeudenoeuds">
```

2) **<xsl:sort>** : Cet élément permet d'effectuer une tri sur un jeu de nœuds. Il doit être placé soit dans un élément <xsl:for-each> soit dans un élément <xsl:apply-templates>.C'est un élément vide qui peut être appelé plusieurs fois pour effectuer un tri multicritères. Chaque appel à cet élément provoque un tri sur un champ spécifique, dans un ordre prédéfini. L'utilisation de cet élément est de la forme :

```
<xsl:sort select="noeud" data-type="text"|"number"|elt
order="ascending"|"descending" lang=nmtoken case-order="upper-
first"|"lower-first"/>
```

- **select** permet de spécifier un nœud comme clé de tri.
- **data-type** correspond au type des données à trier. Dans le cas où le type est `number`, les données sont converties puis triés.
- **order** correspond à l'ordre de tri. Cet attribut vaut `ascending` ou `descending`.
- **lang** spécifie quel jeu de caractères utiliser pour le tri ; par défaut, il est déterminé en fonction des paramètres système.
- **case-order** indique si le tri a lieu sur les majuscules ou minuscules en premier.

Exemple:

```
<xsl:for-each select="livre">
  <xsl:sort select="auteur" order="descending"/>
  <b><xsl:value-of select="auteur"/></b><br/>
  <xsl:value-of select="titre"/><br/>
</xsl:for-each>
```

3) **<xsl:number>** : Cet élément permet d"insérer un nombre formaté pouvant servir de compteur. L'utilisation de cet élément est de la forme :

```
<xsl:number level="single"|"multiple"|"any" count="noeud" from="noeud"
value="expression" format="chaine" lang=nmtoken grouping-separator="car"
grouping-size="nombre"/>
```

- `level` indique quels niveaux doivent être sélectionnés pour le comptage.
- `count` indique quels nœuds doivent être comptés dans les niveaux sélectionnés ; dans le cas où cet attribut n'est pas défini, les nœuds comptés sont ceux ayant le même type que celui du nœud courant.
- `from` identifie le nœud à partir duquel le comptage commence.
- `value` indique l'expression correspondant à la valeur du compteur ; si cet attribut n'est pas défini, le nombre inséré correspond à la position du nœud (`position()`).
- `format` spécifie le format de l'affichage du nombre ; cela peut être un chiffre, un caractère (*a-z, A-Z*) et comprendre un caractère de séparation tel que le point (.), le trait d'union (-) ou autre. Les formats possibles sont "1", "01", "a", "A", "i", "I".
- `lang` spécifie le jeu de caractères à utiliser ; par défaut, il est déterminé en fonction des paramètres du système.
- `grouping-separator` identifie le caractère permettant de définir la séparation entre les centaines et les milliers.
- `grouping-size` spécifie le nombre de caractères formant un groupe de chiffres dans un nombre long ; Le plus souvent la valeur de cet attribut est 3. Ce dernier attribut fonctionne avec `grouping-separator` ; si l'un des deux manque, ils sont ignorés.

Exemple d'utilisation:

```
<xsl:for-each select="livre">
  <xsl:sort select="auteur"/>
  <xsl:number level="any" from="/" format="1. "/>
  <b><xsl:value-of select="auteur"/></b><br/>
  <xsl:value-of select="titre"/><br/>
</xsl:for-each>
```

II.3.6. Condition de teste:

1) **<xsl:if>** : Cet élément permet la fragmentation du modèle dans certaines conditions. Il est possible de tester la présence d'un attribut, d'un élément, de savoir si un élément est bien le fils d'un autre, de tester les valeurs des éléments et attributs. L'utilisation de cet élément est de la forme :

```
<xsl:if test="condition">action</xsl:if>
```

Exemple:

```
<xsl:for-each select="livre">
  <b><xsl:value-of select="auteur"/></b><br/>
  <xsl:value-of select="titre"/><br/>
  <xsl:if test="@langue='francais'">Ce livre est en français</xsl:if>
</xsl:for-each>
```

2) <xsl:choose> : Cet élément permet de définir une liste de choix et d'affecter à chaque choix une transformation différente. Chaque choix est défini par un élément `<xsl:when>` et un traitement par défaut peut être spécifié grâce à l'élément `<xsl:otherwise>`.

Exemple d'utilisation :

```
<xsl:for-each select="livre">
  <b><xsl:value-of select="auteur"/></b><br/>
  <xsl:value-of select="titre"/><br/>
  <xsl:choose>
    <xsl:when test="@langue='francais'">Ce livre est en français</xsl:when>
    <xsl:when test="@langue='anglais'">Ce livre est en anglais</xsl:when>
   <xsl:otherwise>Ce livre est dans une langue non répertorié
```

III.XHTML:

III.1. Introduction à XHTML (*eXtended Hyper Text Markup Language*)**:** Le XHTML servent tout d'abord à faire de la pagination qui est la mise en page d'informations, d'images et de mettre des liens vers d'autres pages WEB.

Le **XHTML** est un langage hybride entre le **XML** et le **HTML**. Il ne s'agit, en fait, que d'une reformulation de HTML pour le rendre compatible avec XML.

Le langage XHTML 1.0 jette les bases d'un langage Web **modulaire** et **extensible** basé sur le **XML.**

Il permet en fait d'adapter progressivement les structures du web à la mise en place imminente des langages sémantiques.

Le XHTML permet l'utilisation d'applications reposant sur le DOM *(Document Object Model)* comme des scripts ou des applets.

Le langage HTML demeure figé et limité, XHTML s'engage à l'épurer et à l'étendre par une série de définitions XML.

III.2. Les Origines : Le langage XHTML est avant tout une version plus élaborée et purifiée du langage HTML, Ce nouveau langage met un terme définitif à l'édition de nouvelles recommandations du HTML.

La première distinction du XHTML se trouve dans la manière de formuler ces documents, En effet, toutes les pages Web se reposant sur l'architecture XHTML, doivent impérativement respecter des règles syntaxiques draconiennes, du fait que ce nouvel outil a été conçu à partir du XML.

La différence principale réside dans la modularisation des définitions de type de document *(DTD)* permettant de déclarer les éléments et les attributs utiles, ainsi que leurs comportements dans un document XHTML.

III.3. Validation : La validation d'un document XHTML est accordée par le processeur XHTML si le document est bien formé (*ne comporte aucune erreur de syntaxes)* et s'il est conforme à la DTD associée, La conformité d'un document XHTML par rapport à une Définition de Type de Document est effective lorsqu'il n'existe, dans ce document, aucune transgression des règles définies par la DTD.

III.4. Structure d'un document XHTML:

Un document XHTML a une structure minimale indispensable :

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict // EN "
" http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
```

Début du document html

< head >	Début de l'entête
<title> Cours XHTML </title>	Informe le navigateur du titre de la page
.....	autres infos
</head>	Fin de l'entête
<body>	Début du texte
.....	corps du document html
</body>	Fin du texte
</html>	Fin du document

III.5 Les principales balises à connaître : Les éléments de base du HTML et du XHTML sont "les balises" : des opérateurs entre des < >. En HTML, beaucoup de balises vont par paire afin de marquer une partie de texte par exemple *en gras* pour **en gras**. Notez le / pour la balise fermante qu'il faudra mettre au bon endroit.

1°) La mise en forme :

- *br* : saut de ligne.
- *hr* : saut de ligne avec un trait entre les lignes.
- *p* : marque un paragraphe.
- *b* : **mettre en gras** .
- *i* : *mettre en italique* .
- *s* : ~~barrer .~~
- *u* :souligner
- *big* : écrire plus gros.
- *small* : écrire plus petit.
- *sup* : mettre en exposant .
- *sub* : mettre en indice.
- *font* : Modifie les attributs de la fonte de caractères en fonction des paramètres.
- *center* : Centre le paragraphe (met un saut de ligne avant et après).
- *span* : Délimite une partir de texte où on peut appliquer les styles souhaités, pour suppléer par exemple aux balises devenues obsolètes en XHTML.
- Abbr : Cette balise est utilisée pour indiquer que ce quelle contient est une abréviation. Cette dernière n'est vraiment utile que si l'on indique son sens grâce à l'attribut title

2°) images et liens :

- ***img*** **:** désigne une image

```
<img  src  = "chemin _ vers _ image" alt = "commentaire"/>.
```

- ***a*** **:** encadre un lien

```
<a  href ="../Math/index.html"> Index  </a>
```

- *href* **:** indique un chemin relatif *(Lien vers une page situé dans le même site).*

Lien vers une page extérieure :

```
<a  href ="http:// geii.univ-lyon1.fr "> Lien  </a>
```

3°) hiérarchisation :

* *ol* : marque le début d'un niveau de liste numérotée.
* *ul* : marque le début d'une liste désordonnée
* *li* : indique une entrée de liste aussi bien pour *ul* que *ol.*
* *dl* : marque le début d'une liste avec terme et définition.
* *dt* : nom d'un terme d'une liste définie avec dl.
* *dd* : définition d'un terme d'une liste définie avec dl.
* *hx* : lettrage d'entête avec x de 1 à 6 (*niveau de titre*).

4°) Tableau :

table : début d'un tableau, composé de lignes avec des cases dedans.

* *tr* : définition d'une ligne qui va contenir des cases. Cette ligne devra être dans un tableau (*entre des balises table*).
* *td* : définition d'une case normale. Cette case devra être dans une ligne (*entre des balises tr*).
* *th* : définition d'une case de titre (centrée et en gras) Cette case devra être dans une ligne (entre des balises *tr*).
* *caption* : Titre du tableau : en caractère normal centré au-dessus du tableau.

Exemple:

```
<table>
<caption>Exemple de tableau</caption>
<tr><th>A</th><th>B</th><th>C</th><th>D</th></tr>
<tr><td>E</td><td>F</td><td>G</td></tr>
<tr><td>H</td><td>I</td><th>J</th></tr>
</table>
```

Exemple de tableau			
A	B	C	D
E	F	G	
H	I	J	

5°) Les formulaires : interfaces de dialogues

*) form : délimite un formulaire. Il pourra contenir autant d'entrées qu'on le souhaite, toutes nommées différemment afin de les différencier. On peut définir une *action* qui sera effectuée lorsqu'on actionnera input

```
submit : <form Method ="get"   Action ="sortie.html">.
```

Dans ce cas, tous les paramètres seront automatiquement transmis.

* *label* : sert à ce que lors d'un clique sur un texte, ça fasse comme si on cliquait sur l'input auquel il correspond

```
<label for="un"> et </label>
```

*) *input* : défini une entrée de formulaire. Son type pourra être :

- radio : défini un ensemble de bouton pour chaque nom *id = "nom"* pour lesquels un seul pourra être sélectionné simultanément.

Exemple :

```
<input name='un'  id='un'  value='un zero' />
```

- checkbox : défini une case à cocher

Exemple :

✔ `<input type='checkbox' name='trois' value ='trois_zero'/>`

- `text` : permet de rentrer un texte qui sera visible

Exemple :

ici `<input type='text' name='quatre' value='ici'/>`

On peut rajouter l'extension *readonly = "readonly"* à toute entrée input afin de la rendre non éditable.

- `password` : permet de rentrer un texte qui sera caché, comme l'entrée d'un mot de passe, les caractères sont remplacés par des *.

Exemple :

`<input type='password' name='cinq' value='pas là'/>`

- `button` : affiche un bouton

Exemple:

`<input type='button' name='six' value='bouton'/>`

- `file` : permet de repérer un fichier dans l'arborescence de l'ordinateur

Exemple :

`<input type='file' name='sept' value = 'fichier'/>`

- `submit` : quand un utilisateur clique sur un bouton "submit", le questionnaire est soumis avec l'action spécifiée par la balise *form* . J'ai dit dans la balise *form* que cette page doit elle-même s'appeler tout a l'heure.

Exemple :

```
<input type ="submit" name="huit"  value = "Oui"/>
```

- `reset` : remet tous les boutons aux valeurs initiales

Exemple :

```
<input  type = "reset" name=" neuf "   value = "Non"/>
```

- `select` : un menu déroulant

Exemple :

0

```
< select  name = 'dix' >
< option value = "0" > 0 < / option >
< option value = "1" > 1 < / option >
< / select >
```

- `textarea` : défini une zone de texte

Exemple :

C'est la première ligne
Voici la seconde

Et la dernière

```
<textarea name='onze' rows = '4'  cols="55" >
C'est la première ligne
Voici la seconde

Et la dernière
</textarea>
```

IV.Conclusion:

Dans ce chapitre nous avons étudié les langages de balisages XML, XSLT et XHTML, nous concluons que:

- Les documents XHTML sont conformes à XML. Ainsi, ils sont directement lisibles, éditables, et valables avec les outils XML standards.

- Les documents XHTML peuvent être écrits pour fonctionner aussi au mieux qu'ils ne le faisaient précédemment dans les agents utilisateurs compatibles HTML 4 ainsi que dans les nouveaux agents utilisateurs compatibles XHTML 1.0.

- Les documents XHTML peuvent utiliser des applications *(soit des scripts et des applets)* qui repose sur le Modèle Objet de Document HTML autant que sur le Modèle Objet de Document XML.

- Par conséquent le langage XHTML va nous servir de document hôte au langage qui sera entamé au prochain chapitre qui est le XForms .

Chapitre 2 :

XForms

I.Preface: [XF01]

W3C XForms-(XML powered Web Forms) est une révision des formulaires HTML. Les formulaires en ligne sont critique au commerce électronique sur Internet, et les formulaires HTML qui existe depuis 1993 commence à être dépassé. L'arrivée de XML a bouleversée le monde du service web en offrant des moyens de connexion de diverses technologies d'informations.

XForms permet au concepteur web de se concentrer sur les aspects principaux de l'application et de compter sur la plate-forme fondamentale de XForms pour les services suivants :

- Produire les interfaces utilisateur appropriées pour le dispositif de connexion.
- Fournir la rétroaction interactive a l'utilisateur par l'intermédiaire de la validation automatisée côté client
- Validation automatique des entrées utilisateur sur le serveur.
- Rassembler les entrées utilisateur sur le serveur dans une structure appropriée à l'application principale.

II. Architecture: L'architecture globale de XForms a été divisée en plusieurs composants. Un dispositif principal de cette décomposition MVC est une séparation claire du modèle de sa présentation finale.

Ces composants sont les suivants :

a) Model (Modèle) :

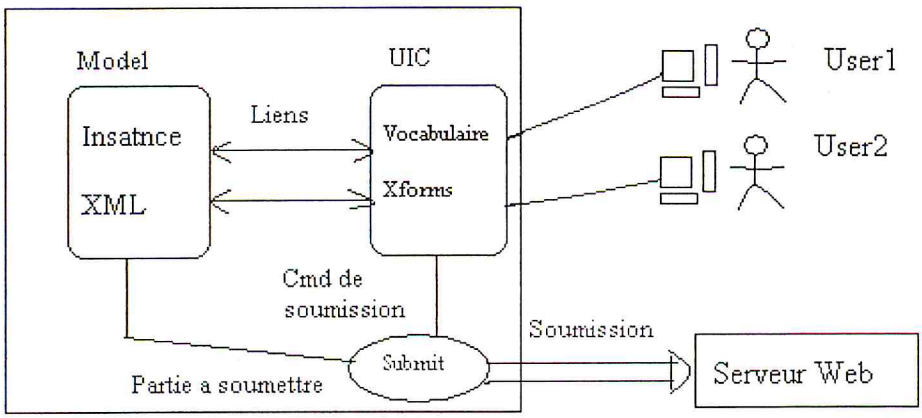
Le modèle de données incorpore une instance XML qui reçoit l'entrée d'utilisateur, les contraintes employées pour valider cette entrée, et les metadata nécessaire sur la manière avec laquelle les entrées utilisateur seront communiquées au web serveur.

b) UI (User Interface) :

XForms définit un vocabulaire d'interface utilisateur qui se compose des commandes abstraites et des constructions d'agrégation employées pour créer des interfaces utilisateur riches. Ce vocabulaire permet de fournir des applications web XForms à différents dispositifs et modalités.

c) Submit (Soumettre) :

Ceci permet à l'auteur d'application web d'indiquer où, comment, et quelles parties de données à soumettre au serveur web. Il permet également au concepteur d'application d'indiquer quelles actions à prendre lors de la réception d'une réponse du serveur.



L'architecture globale de XForms

Figure 2.0 : *Architecture XForms.*

III.Vue sur XForms :

Un formulaire XForms sera crée a l'intérieur d'un document XHTML. Le modèle de XForms qui est délimité par les balises <model>…</model> est placé dans l'élément de XHTML <head>.

Les commandes d'interface utilisateur de XForms créent l'interaction utilisateur et apparaissent dans l'élément de XHTML <body>.

III.1 Modèle De XForms :

Tous les aspects non présentables des applications web sont encapsulés dans le modèle de données de XForms.

Puisque nous employons maintenant XML, nous ne devons plus nous limiter à un modèle plat de données se composant d'un ensemble de paires nom-valeur non-typées. Au lieu de cela, nous encapsulons l'information rassemblée de l'utilisateur dans un document structuré de XML. Ceci s'appelle l'instance XML.

Elément <instance> déclare le XML Template qui reçoit les entrées utilisateur et les valeurs par défaut.

```
<model id="p1" xmlns="http://www.w3.org/2002/xforms">
  <instance>
    <personne xmlns="">
       <nom>
         <prenom/>
         <nomdefamille/>
       </nom>
       <age/>
       <email/>
       <adresse>
         <rue/>
         <ville/>
         <codepostale/>
       </adresse>
       <datenaissence>
         <jour/>
         <mois/>
         <annee/>
       </datenaissence>
       <ssn>000-000-000</ssn>
       <sexe>m</sexe>
    </personne>
  </instance>
</model>
```

Noter que des données élémentaires composées telles que adresse sont maintenant modelées pour refléter la structure des données, à la différence de l'utilisation d'une paire plate de nom-valeur. Ceci permet à l'application la saisie obligatoire de données intermédiaires.

III.2 Interface utilisateur De XForms:

Elle rassemble les entrées utilisateur, une telle interface utilisateur est créée par une structure de balisage spéciale appelée commandes d'interface utilisateur *(UIC)*.
A cause de la séparation du contenu de la présentation, le balisage de l'interface utilisateur XForms apparaît n'importe où dans l'élément de XHTML <body>, avec le reste du balisage du document. Contrairement aux formulaires HTML, les commandes d'interface utilisateur doivent apparaître dans l'élément <form>.

Voici un tableau récapitulatif des commandes d'interface utilisateur de XForms et leurs correspondances HTML qui serons détaillées après:

Formulaires HTML	XForms
<input type="text">	<input>
<input type="password">	<secret>
<input type="textarea">	<textarea>
<input type="checkbox">	<input> brachement verrs xsd:boolean
<input type="radio">	<select1>
<select>	<select>
<input type="submit">	<submit>
<input type="reset">	<trigger> avec un handler <reset>
<input type="file">	<upload>
<input type="image">	<trigger> avec label image
<input type="boutton">	<trigger>

En plus de ces éléments, l'élément <output> a été rajouté pour l'affichage du texte.

Et l'élément <range> pour la sélection parmi une rangé de données.

Chaque commande d'interface contient les attributs et éléments suivants:

Balisage	Description
model	Identifie le model contenant l'instance branchée
ref	Identifie un seul nœud de l'instance branchée
nodeset	Identifie un jeu de nœuds de l'instance branchée
bind	Identifie un branchement abrégé.
appearance	Identifie la présentation, valeurs:full, minimal, compact
class	Identifie la classe de style
<label>	Label des commandes d'interface utilisateur
<help>	Texte d'aide des commandes d'interface utilisateur
<hint>	Conseil des commandes d'interface utilisateur
<alert>	Message délivre en cas de données non valides
acceskey	Touches de raccourci clavier
navindex	Position dans la séquence de la navigation
inputmode	Facilite la saisie du texte sur de petits dispositifs
eventing	Attribut de XML Events pour déclencher des évènements
<action>	Traiteur déclaratif d'action définis par XForms

III.2.1 Les commandes d'interface utilisateur (UIC):

Dans cette section nous allons détailler chaque commande de UI XForms.

Lors de la collecte des données, ces dernières sont stockées dans un arbre XML pour être utilisées plus tard.

a)- Collecter le texte saisi:

C'est la tache la plus commune de l'utilisation des formulaires électroniques, les commandes XForms correspondantes sont <input>, <textarea>, <secret>.

* **<input>** : Champ de texte pour introduire des valeurs.
* **<textarea>** : Champ de texte multi-lignes.
* **<secret>** : pour l'introduction de valeurs destinées a ne pas être visualisées *(Ex: mot de passe).*

b)-La sélection parmi un ensemble de valeurs :

Un tel choix se manifeste souvent dans une interface visuelle sous forme de listes déroulantes ou de boutons radio, nous retrouvons ce type d'interface dans de petits dispositifs tels que les caméras numériques ou les téléphones cellulaires par exemples, dans un menu guidant progressivement l'utilisateur à un choix spécifique.

-**<select>**: Crée les commandes de choix qui renvoient une liste de valeurs.

-**<select1>**: Pour créer les commandes de choix qui renvoient une valeur atomique *(0,1).*

c)-La sélection parmi une rangé de valeurs :

XForms définit une commande générique **<rang>** qui peut être employée pour sélectionner une valeur d'un ensemble de valeurs bien- ordonnées.

L'élément <rang> renvoie une valeur unique de l'ensemble de valeurs disponibles

En plus des attributs acceptés par les autres commandes, il dispose de 3 autres attributs:

start	Spécifie la première valeur de l'ensemble
end	Spécifie la dernière valeur de l'ensemble
step	Spécifie le pas utilisé en se déplaçant d'une valeur à l'autre

d)- Chargement de données:

Le chargement de données employant les formulaires en ligne a été présenté pour la première fois dans le HTML par l'intermédiaire du dispositif maintenant familier: chargement de fichier.

Dans les formulaires HTML, ce dispositif a été ajouté à la commande <input> en ajoutant un attribut supplémentaire qui permet à la commande <input> d'être représenté comme étant une boite de dialogue permettant de sélectionner le fichier souhaité.
Aujourd'hui, dans les sites web, ce dispositif est employé dans l'ajout des pièces jointes aux e-mails.
XForms définit l'élément **<upload>** qui permet une fonctionnalité similaire.

La commande <upload> de XForms va encore plus loin. Lorsque Les formulaires HTML en été conçues, les Browser tournés sur des machines de bureaux. Désormais l'accès au web est disponible sur différents dispositifs, allant des machines de bureaux aux téléphones cellulaires. En conséquence, la capacité de chargement de données a une applicabilité bien plus large que le simple chargement de fichiers. Par exemple, un téléphone cellulaire équipé d'un appareil photo numérique pourrait employer la commande <upload> pour créer les interfaces utilisateur qui permettent à l'utilisateur de transmettre les photos prises.

La commande <upload> de XForms a été conçue pour couvrir tout ce genre d'utilisation.
En plus des éléments et attributs en commun avec les autres commandes, <upload> dispose en plus de 2 sous éléments <filename> et <mediatype>.

Voici leur Fonctions:

filename	Spécifie le nom de fichier par défaut sur lequel <uopload> va pointer
mediatype	Spécifie le filtre de fichiers utilisés *(extension de fichiers à visualiser)*

e)- Déclenchement d'actions:

Les sections précédentes ont décrit les diverses commandes d'interface utilisateur de XForms pour collecter des données de l'utilisateur. La collecte de données à part, des commandes d'interface utilisateur sont employées pour déclencher des actions spécifiques. Dans les interfaces visuelles traditionnelles, de telles commandes se manifestent en boutons. Des boutons dans une interface visuelle sont branchés vers des actions à exécuter sur l'activation du bouton. Une action utilisateur spécifique par exemple appuie sur un bouton s'appelle un événement *(évoqués dans une section ultérieure)*, causes l'action à déclencher.

XForms définit la commande abstraite `<trigger>` qui fournit un cadre générique pour des événements et des actions reliées à l'utilisateur.

La commande `<trigger>`, avec le reste du balisage de l'interface utilisateur abstraite, n'est pas spécifique à une représentation donnée, Par exemple un bouton elle peut avoir différentes interfaces visuelles.

XML et XForms définissent un ensemble d'événements pour déclanchent une action spécifique, ces événements seront détaillés plus tard dans un tableau.

A la différence des autres commandes XForms, <trigger> n'opère pas directement sur les données d'instance stockées dans le modèle, cela rend les couples d'attributs (model,ref) ou (model,nodeset) facultatifs. Cependant ces derniers peuvent être utilisés pour activer ou désactiver <trigger> *(en utilisant l'attribut "relevant", cette section sera évoquée plus tard)*.

<trigger> définit les attributs et élément suivants:

Considérant l'espace de nom suivant: `xmlns:ev="http://www.w3.org/2001/xml-events"`

ev:event	Indique le type d'événement qui déclenche l'action encapsulée. Le type d'événement indiqué peut être de n'importe quel des types d'événement standard *DOM2* ; il peut également être de n'importe quel des types d'événement spéciaux définis par les spécifications de *XForms*, par exemple, xforms-reset.
ev:handler	Indique l'action à déclencher lors de la réception de l'événement
<action>	Pour encapsuler plus actions à déclancher. Dans la plupart des cas les actions à déclancher sont indiquées comme sous éléments de <trigger> plutôt qu'a être indiquées indirectement par l'attribut *handler*.

f)- <u>Soumission de données:</u>

Les commandes d'interface utilisateur décrites jusqu'ici peuvent rassembler des données d'utilisateur et permettre le déclenchement des actions en réponse aux événements d'interaction d'utilisateur et sont suffisantes pour mettre en application un grand nombre d'interaction en ligne.

Les concepteur de XForms ont sentir que le processus de soumission de données collectées vers les serveurs web été suffisamment important pour mériter sa propre commande d'interface utilisateur. Pour cela XForms définit la commande **<submit>**.

La commande <submit> peut être implémentée en utilisant la commande <trigger> qui invoque l'action <send>. Ceci dit <submit> rend cette tache beaucoup plus facile à réaliser.

- **Anatomie de <submit>:**

Comme <trigger>, <submit> n'affecte pas directement le modèle de données, donc les attributs de branchement ne sont pas exigés. Cependant comme dans le cas de <trigger>, ces attribut sont utilisés pour activer ou désactiver la commande.

Avant de soumettre une donnée il faut d'abord savoir:

- Ce qu'on doit soumettre, grâce aux attributs de branchement.
- Où doit on soumettre, via l'attribut "**action**".
- Comment soumettre, via l'attribut "**method**".

Ces trois détails sont encapsulés dans l'élément **<submission>** dans le modèle de données.

Il faut aussi signaler un autre attribut de l'élément <submission>:

replace, qui permet d'interpréter la réponse du serveur d'une certaine manière.

Voici un tableau qui met en évidence les trois valeurs que peut prendre l'attribut replace:

all	La réponse du serveur met à jour toute la page
instance	La réponse du serveur met à jour que l'arbre du modèle de données
none	La réponse du serveur est interprétée comme un acquittement

Tableau récapitulatif des valeurs que peut prendre l'attribut "method":

Method	Description
poste	Envoi XML utilisant le HTTP POST
put	Envoi XML utilisant le HTTP PUT
get	HTTP GET
multipart-post	Envoi XML utilisant le HTTP POST
form-data-post	Envoi des données utilisant multipart/form-data
urlencoded-post	Envoi des données url-encoded

III.3 Création d'interface utilisateur complexe:

Des interfaces utilisateurs complexes peuvent être créées en agrégeant les commandes d'interface utilisateur décrites précédemment. Nous évoquerons dans cette section les constructions d'agrégation de *XForms*. Comme les commandes d'interface utilisateur, des constructions d'agrégation sont conçues pour donner une mise en forme logique ou physique (si l'on peut dire) de l'interface utilisateur.

Le but est de saisir des informations suffisantes sur la conversation homme-machine fondamentale et de fournir de ce fait une expérience utilisateur satisfaisante sur une variété de modalités et de dispositifs.

Par exemple une grande interface utilisateur qui demande affichage à sa taille, peut être découpée en plusieurs affichages.

a. Agrégation utilisant <group>:

L'agrégation <group> peut être utilisée pour grouper logiquement des commandes d'interface utilisateur.

Lors de l'évocation du modèle de données, nous avons vu que les données été groupées dans des structures.

Par exemple:

```
<Adresse>
    <cite/>
    <vile/>
</Adresse>
```

Imaginons maintenant qu'on peut faire la même chose avec les commandes d'interface utilisateur.

Donnant un exemple pour mieux comprendre:

```
<group xmlns="http://www.w3.org/2002/xforms" ref="/adresse" accesskey="a">
        <label>Adresse e-mail:</label>
        <input ref="cite">
              <label>Cité</label>
        </input>
        <input ref="ville">
              <label>ville</label>
        </input>
</group>
```

Au lieu de l'attribut "ref" dans chaque commande <input>, l'ensemble des <input> à été groupé dans un même groupe, et du coup tout les paramètres de <group> affecteront les <input>. L'utilisation de <group> nous a épargné du code.

b. Interaction dynamique avec <switch>:

Un dispositif principal des formulaires électroniques est leur capacité de réagir aux entrées utilisateur et d'employer des mises à jour dynamiques pour faciliter l'accomplissement rapide de tâche. Ainsi, tandis que les formulaires papier sont statiques, les formulaires électroniques aident typiquement l'utilisateur dans la navigation à travers diverses étapes d'une interaction complexe convenablement en indiquant ou en cachant des sections du formulaire.

Voici La forme générale de <switch>:

```
<switch>
        <case id="...">...</case>
        <case id="...">...</case>
    ¦
    ¦
        <case id="...">...</case>
</swith>
```

- Chaque <case> comporte des commandes d'interface.
- un seul <case> est actif. Par défaut c'est le premier <case> qui est activé.
- L'élément <toogle> permet de naviguer à travers les <case> grâce à son attribut "case".

c. Répétition de structures avec <repeat>:

La construction <repeat> de XForms définit une construction déclarative qui peut être employée pour réitérer les collections finies de comme noeuds dans le modèle de XForms, par exemple, les articles dans un caddie.

Le but de la construction <repeat> est de générer un nœud spécifique dans notre arbre de données (nœud déjà existant).

Des opérations peuvent être liées a repeat tel que setindex, insert et delete qui nous permettent de faire défiler, ajouter et supprimer des noeuds de la collection.

Noter que toutes ces opérations dépendent de la notion d'un noeud courant bien défini dans la collection.

XForms appelle ceci l'index de <repeat>.

Ainsi, la construction <repeat> encapsule les information suivantes :

Id	Identifiant de la construction <repeat>
Collection	La collection de noeuds étant réitérés, établit par nodeset
Index	Pointeur sur le nœud courant de la collection
UI Template	L'interface utilisateur instancier par <repeat>
Controls	Contrôles permettent d'ajouter, supprimer, défiler les noeuds
Presentation	Indique la partie de la collection à afficher

-Attributs <repeat> optionnelles :

startindex	Spécifie le 1e membre de la collection présenté à l'utilisateur
number	Spécifie le nombre max des éléments à afficher

-Éléments enfant de <repeat> :

Le corps de la construction <repeat> est l'interface utilisateur destinée à être répété, Ceci peut employer tout le vocabulaire d'interface utilisateur de XForms, en plus du balisage défini par le langage du document l'hôte. Ainsi, en utilisant XForms dans XHTML, le corps de la construction <repeat> pourrait employer le balisage de XHTML en plus de celui défini par XForms.

III.4 Les propriétés du modèle XForms:

Cette partie définit des propriétés pouvant être liées aux nœuds de donnée d'instance avec l'élément `bind`.

III.4.1. La propriété `relevant`:

Cette propriété prend une expression XPath, évaluée durant l'interaction utilisateur pour retourner une valeur booléenne.

Lorsque un nœud a sa propriété relevant mise à false, le contrôle interface utilisateur lié à ce nœud devient inaccessible à l'utilisateur.

III.4.2. La propriété `required`:

Un formulaire peut *imposer* certaines valeurs, et cette obligation peut être dynamique.

Définit si une valeur est obligatoire avant que les données d'instance ne soient soumises.

* **Avantage** : L'obligation de remplir certains champs peut être définit par l'intermédiaire d'un schéma, dans ce cas les valeurs de ces champs seront toujours exigées.

La propriété `required` sert à rendre des champs obligatoires selon les entrées de l'utilisateur.

III.4.3. La propriété `readonly`:

La propriété `readonly` nous permet de rendre le changement de certaines valeurs possibles seulement si certaines conditions sont vérifiées.

III.4.4. La propriété `constraint`:

La propriété `constraint` peut être employée pour déclarer des contraintes de validité destinées à être évaluée durant l'interaction d'utilisateur.

Celles-ci servent à raffiner les contraintes statiques définies par l'intermédiaire du XML Schéma.

III.4.5. La propriété `calculate`:

La valeur de la propriété `calculate` est une expression XPath qui calcule la valeur à stocker dans le noeud auquel la propriété `calculate` s'applique.

III.5 Les fonctions XForms:

La partie précédente montrée comment des expressions XPath peuvent être employées en calculant les valeurs de certains champs de formulaires. En écrivant de telles expressions, il est utile d'avoir un ensemble standard de fonctions de service pour exécuter des calculs communs.

XForms 1.0 définit une poignée de telles fonctions d'extension, et ceux-ci seront décrits dans le reste de cette partie.

III.5.1. Fonctions booléennes:

Cette section définit les diverses fonctions booléennes fournies par la bibliothèque de fonction de XForms.

a.1 Function boolean-from-string: *boolean* boolean-from-string(*chaîne*)

Renvoie la valeur "`true`" si le paramètre demandé *chaîne* de type string est la chaîne "`true`", ou "`1`", ou elle renvoie la valeur "`false`" si le paramètre *chaîne* est la chaîne "`false`", ou "`0`". Elle se révèle utile lors de l'appel d'une valeur du type de donnée `xsd:boolean` du schéma dans une expression XPath. Si la chaîne paramètre ne correspond à aucune de celles mentionnées ci-dessus, selon une comparaison insensible à la casse, alors le traitement s'interrompt avec une exception.

a.2 Function if: *string* if(*booléen*, *chaîne*, *chaîne*)

La fonction if évalue le premier paramètre en tant que valeur booléenne de type boolean : s'il vaut "true", alors il renvoie le deuxième paramètre *chaîne* de type string, sinon il renvoie le troisième paramètre.

III.5.2. Fonctions numériques :

Cette section définit les diverses fonctions numériques fournies par la bibliothèque de fonction XForms

Ces fonctions ont été ajoutées à XForms parce qu'elles étaient absentes dans XPath mais ont été considérées nécessaires pour certaines applications.

Certaines de ces fonctions sont adoptées par le groupe de travail de XPath pour l'inclusion dans XPath 2.0.

b.1 Calcul du Minimum, maximum, et de la moyenne:

b.1.1) number ***avg(*** ensemble-de-nœuds *)*:

La fonction `avg` renvoie la moyenne arithmétique du résultat de la conversion des valeurs de chaîne de chaque nœud de l'argument ensemble de nœuds en un nombre (type number).

b.1.2) number ***min*** *(ensemble*-de-nœuds*)*:

La fonction `min` renvoie la valeur minimum du résultat de la conversion des valeurs de chaîne de chaque nœud de l'argument ensemble de nœuds en un nombre (type number).

b.1.3) *number* **max** (*ensemble-de-nœuds*)

La fonction `max` renvoie la valeur maximum du résultat de la conversion des valeurs de chaîne de chaque nœud de l'argument ensemble de nœuds en un nombre (type number).

b.2 *number* **count-non-empty**(*ensemble-de-nœuds*):

XPath contient la fonction `count` qui renvoie le nombre de noeuds dans un jeu de nœud. XForms ajoute une fonction `count-non-empty` qui renvoie le nombre de nœuds non vides dans l'argument ensemble de nœuds.

Un nœud est considéré non vide s'il est convertible en une chaîne dont la longueur est supérieure à zéro.

b.3 *number* **index**(*chaîne*):La fonction `index` admet comme argument une chaîne qui est l'id de `repeat`, Il renvoie la valeur courante de l'index de répétition pour cette structure de <repeat>. Si l'argument fourni n'identifie aucun élément `repeat`, alors le traitement s'interrompt avec une exception.

III.5.3. Les fonctions de chaîne:

a.) *string* **property**(*chaîne*): La fonction `property` renvoie la propriété XForms nommée par le paramètre *chaîne*.

b.) Les fonctions de date et d'heure : Cette section définit les diverses fonctions date-heure fournies par la bibliothèque de fonction XForms

** string **now**()

La fonction `now` renvoie la date et l'heure courantes du système sous forme d'une valeur de chaîne dans le format canonique `xsd:dateTime` du schéma XML.

Si l'information de fuseau horaire est disponible, alors elle est comprise (normalisée au format UTC), sinon une valeur par défaut de l'implémentation sera utilisée.

Autres fonctions date et heure:

Fonction	Description	exemple
`days-from-date`	`Cette fonction prend une chaine représentant une valeur de type date ou dateTime et renvoie le nombre de jours depuis janvier 1, 1970.`	`days-from-date('1971-01-01')` *resultat:365*
`seconds-from-dateTime`	`Similaire a la précédente sauf qu'elle`	`seconds-from-dateTime(`

	prend un dateTime et retourne le nombre de secondes depuis janvier 1, 1970.	'1970-01-02T00:00:00Z') *resultat:* 86400

c.) Les fonctions d'ensemble de nœuds:

** *node-set* **instance**(*chaîne*): Un modèle XForms peut contenir plusieurs instances : cette fonction permet d'accéder aux données d'instance, dans le même modèle XForms.

III.6 Les actions XForms:

Un but principal en concevant XForms 1.0 était de permettre la fonctionnalité des formulaires en ligne sans recourir à un script étendu par l'intermédiaire des langages comme le Javascript, *(qui est indispensable au HTML).*

Cependant, tout ceci a à un prix ; il devient difficile se déployer l'interaction utilisateur dépendante de scripts sur une variété de dispositifs et environnements *(téléphones mobiles).* En outre, il est difficile de maintenir les pages HTML qui se fondent fortement sur des scripts.

XForms 1.0 essaye de réduire au maximum le besoin de script en créant des traiteurs d'événement XML qui couvrent des cas communs d'utilisation.

III.6.1 Action <setfocus> :

L'action <setfocus> est employée pour déplacer le focus vers une commande UI XForms.

L'élément <setfocus> utilise l'attribut `control` pour identifier le contrôle qui recevra le focus.

La valeur de cet attribut est `un id` d'une commande UI XForms.

III.6.2 Action <setvalue> :

L'action <setvalue> peut être employés pour placer une valeur spécifique dans un élément d'instance.Les attributs de l'élément <setvalue> indiquent la valeur à placer et où devrait-elle être placée.

III.6.3 Action <load> :

L'action <load> est employée pour charger une ressource indiquée, par exemple, un document externe en indiquant son URL.

III.6.4 Action <send> :

Nous avons décrit la commande d'interface utilisateur <submit>.

La commande <submit> réalise la fonctionnalité décrite en appelant implicitement l'action <send>.

III.6.5 Action <reset> :

Cette action lance le traitement de réinitialisation en distribuant un événement `xforms-reset` à l'élément `model` indiqué.

III.6.6 Action <message> :

Cette action encapsule un message à afficher à l'utilisateur.

III.6.7 Action <action> :

L'action `<action>` permet de regrouper plusieurs actions.

Lorsqu'on utilise l'élément `<action>` pour regrouper des actions, il faut faire attention à lister l'événement sur l'élément `action` plutôt que sur les actions qui y sont contenues.

III.6.8 Action <dispatch> :

L'action `<dispatch>` permet à l'auteur de XForms d'expédier *(distribués)* des événements explicites pendant l'interaction utilisateur.

III.6.9 Les actions <insert>, <delete>, et <setindex> :

XForms définit trois traiteurs déclaratifs spéciaux pour l'usage de `<repeat>`, qui seront récapitulé dans le tableau suivant :

Action	Description
<insert>	Permet l'insertion d'un noeud en utilisant <repeat>.
<delete>	Permet la suppression d'un noeud en utilisant <repeat>.
<setindex>	Défiler à travers les nœuds en utilisant <repeat>

III.7 Les événements XForms:

Le traitement XForms se définit en fonction d'événements, de gestionnaires d'événement et de réponses aux événements. Le langage XForms emploie le système d'événements défini dans DOM2 Events *(exposé aux auteurs XML par le module XML Events)*, avec les phase suivantes:

1. Capture de l'événement.
2. L'arrivée de l'événement sur sa cible.
3. Bouillonnement de l'événement *(L'événement remonte au document l'hôte)*.

Dans la suite la section, toute mention d'une commande de formulaire comme élément cible représente l'un des éléments suivants : `input`, `secret`, `textarea`, `output`, `upload`, `trigger`, `range`, `submit`, `select`, `select1` ou `group`.

Ainsi, la description du modèle de traitement XForms réduit à énumérer les aspects suivants:

Quoi	les divers événements XForms-spécifiques qui peuvent être déclenchés.
Quand	Indiquer à quels points dans le modèle de traitement XForms ces événements sont soulevé.
Où	Indiquer où ces événements sont envoyés.
Qui	les divers acteurs responsables a manipuler ces événements.
Comment	Indiquer comment les acteurs manipulent ces événements.

Nous organiserons le reste de cette partie en groupant les événements XForms par rapport a quand et comment ils sont déclenchés .Les événements XForms peuvent être groupés dans les classes suivantes :

Initialisation	Evénement déclenché lors du chargement du document
Interaction	Evénement déclenché par l'interaction utilisateur
Notification	Evénement conçu pour déclencher un comportement spécifique
Erreurs	Déclenchés quand des erreur sont produites

III.7.1 Les événements d'initialisation :

a. L'événement `xforms-model-construct:`

Distribué par le processeur du document conteneur afin d'amorcer l'initialisation du processeur XForms.

L'événement est adressé à l'élément <model>. L'événement est manipulé en chargeant toutes les ressources indiquées, telles que des définitions de schéma et des déclarations d'instances. Si aucune erreur ne résulte, le traitement continue à la prochaine étape.

b. L'événement `xforms-model-construct-done:`

Distribué après l'achèvement du traitement de l'événement `xforms-model-construct.`

Cet événement est augmenté quand tous les modèles sont initialisés et préparent l'interaction utilisateur. En réponse à cet événement, le traitement continue en initialisant l'interface utilisateur qui est liée au modèle. Initialisant les résultats d'interface utilisateur dans toutes les commandes d'interface utilisateur étant liées aux instances de données.

c. L'événement `xforms-ready`:

Distribué dans le cadre du traitement de l'événement `xforms-model-construct-done.`

Cet événement est envoyé à chaque modèle pour indiquer que l'interface utilisateur a été initialisée.

d. L'événement `xforms-model-destruct`:

Distribué par le processeur pour avertir de la mise hors-service imminente du processeur XForms, laquelle peut se produire à partir d'une action de l'utilisateur, ou à partir d'une action XForms load, ou comme conséquence de la soumission du formulaire.

III.7.2 Les événements d'interaction :

a. Les événements `xforms-next` et `xforms-previous`:

Distribués en réponse à une requête de l'utilisateur pour naviguer vers la commande de formulaire suivante, ou la commande précédente.

L'action implicite de ces événements produit une navigation selon l'ordre de navigation par défaut. Par exemple, avec une interface clavier, la touche tabulation pourrait générer un événement `xforms-next`, tandis que les touches majuscule + tabulation pourraient générer un événement `xforms-previous.`

b. L'événement `xforms-focus`:

Distribué en réponse au fait que le focus se place sur une commande de formulaire.

L'action implicite de cet événement produit le placement du focus sur la commande de formulaire cible, si cette commande est capable de recevoir le focus.

c. Les événements `xforms-help` et `xforms-hint:`

Distribués en réponse à une requête de l'utilisateur pour une *assistance ou des indications*. L'action implicite de ces événements produit ce qui suit : si la commande de formulaire comporte les éléments `help`/`hint`, ils servent à construire un message affiché à l'utilisateur. Sinon, les agents utilisateurs peuvent fournir des messages d'assistance ou d'indication par défaut, mais ils ne sont pas oblligés de le faire.

d. L'événement `xforms-refresh`:

Distribué en réponse à une demande de mise à jour de toutes les commandes de formulaire associées à un modèle XForms particulier.
L'action implicite de cet événement est la suivante : l'interface d'utilisateur reflète l'état du modèle, ce qui veut dire que toutes les commandes de formulaire reflètent les données d'instance correspondantes auxquelles elles sont liées :

- sa valeur courante ;
- sa validité ;
- si elle est obligatoire (`required`), en lecture seule (`readonly`) ou pertinente (`relevant`).

e. L'événement `xforms-revalidate`

Distribué en réponse à une demande de revalidation d'un modèle XForms particulier.
L'action implicite de cet événement est la suivante :
La gestion minimale de cet événement doit satisfaire aux conditions suivantes :

1. Tous les nœuds de donnée d'instance dans tous les éléments `instance` de l'élément `model` sont vérifiés par rapport à un éventuel schéma XML défini.
2. Tous les nœuds de donnée d'instance dans tous les éléments `instance` de l'élément `model` sont vérifiés par rapport aux éventuelles propriétés d'élément de modèle liées qui définissent des contraintes sur la valeur, c'est-à-dire `required`, `constraint`
3. Les événements de notification appropriés sont distribués à celles des commandes de formulaire où la propriété d'élément de modèle correspondante est évaluée à une valeur différente de celle du début du traitement de cet événement.

f. L'événement `xforms-recalculate:`

Distribué en réponse à une requête pour le recalcul de tous les calculs associés à un modèle XForms particulier.

g. L'événement `xforms-rebuild`:

Distribué en réponse à une requête pour reconstruire les structures de données internes du suivi des dépendances de calcul dans un model XForms particulier.
L'action implicite de cet événement est la suivante : les structures des données de dépendance sont reconstruites, puis la liste des changements est ajustée pour contenir des références à tous les nœuds d'instance ayant une expression de calcul associée, de sorte qu'un recalcule *complet* soit effectué à la prochaine distribution d'un événement xforms-recalculate au modèle.

h. L'événement `xforms-reset :`

Distribué en réponse à une requête de l'utilisateur pour réinitialiser le modèle.
L'action implicite de cet événement est la suivante :
Les données d'instance sont réinitialisées selon la structure de l'arbre et les valeurs qu'elles avaient immédiatement après avoir traité l'événement `xforms-ready`. Les événements `xforms-rebuild`, `xforms-recalculate`, `xforms-revalidate` et `xforms-refresh` sont ensuite successivement distribués à l'élément `model`.

i. L'événement `xforms-submit:`

Voir la partie réservée a la soumission.

III.7.3 Les événements de notification:

a. L'événement `DOMActivate`:

Distribué en réponse à la *requête d'action implicite* d'une commande de formulaire, par exemple, presser un bouton ou taper la touche entrée.

b. L'événement `xforms-value-changed`:

Distribué en réponse au fait qu'un changement confirmé survienne sur un nœud de donnée d'instance lié à une commande de formulaire, comme lorsque l'utilisateur navigue hors de la commande de formulaire.

c. Les événements `xforms-select` et `xforms-deselect`:

Distribués en réponse à la sélection, ou à la désélection, d'un élément dans un élément `select`, `select1` ou `switch`.

d. Les événements `xforms-scroll-first` et `xforms-scroll-last`:

Distribués en réponse au fait qu'une action **`setindex`** essaye de fixer un index en dehors de l'intervalle défini par un élément **`repeat`**.

e. Les événements `xforms-insert` et `xforms-delete`:

Distribués en réponse au fait qu'un gestionnaire d'événement invoque une action XForms **`insert`**, ou **`delete`**, et ajoute, ou supprime, avec succès un élément de répétition.

f. L'événement `xforms-valid:`

Distribués en réponse au fait qu'un nœud de donnée d'instance devienne valide.
Cet événement est distribué dès lors que la valeur du nœud de donnée d'instance lié change, et en outre, lorsque le nœud de donnée d'instance lié devient indirectement valide, au travers de la propriété d'élément de modèle **`constraint`** évaluée à la valeur "**`true`**".

g. L'événement `xforms-invalid`:

Distribué en réponse au fait qu'un nœud de donnée d'instance devienne invalide.
Cet événement est distribué dès lors que la valeur du nœud de donnée d'instance lié change, et en outre, lorsque le nœud de donnée d'instance lié devient indirectement invalide, au travers de la propriété d'élément de modèle **`constraint`** évaluée à la valeur "**`false`**".

h. L'événement `DOMFocusIn`:

Distribué en réponse au fait qu'une commande de formulaire reçoit le focus.

i. L'événement `DOMFocusOut:`

Distribué en réponse au fait qu'une commande de formulaire perde le focus.

k. L'événement `xforms-readonly`:

Distribué en réponse au fait qu'un nœud de donnée d'instance devienne en lecture seule.

III.7.4 Les événements d'erreurs:

Les indications d'erreur surviennent à la suite de conditions inhabituelles dans le processeur XForms. Certaines d'entre elles sont des erreurs fatales qui interrompent le traitement : elles comportent le suffixe exception. D'autres servent simplement à la notification : elles portent le suffixe error. Pour tous les événements décrits dans cette section, le processeur XForms peut, dans une certaine mesure, effectuer une gestion minimale, par exemple, tenir un journal des messages d'erreur dans un fichier.

a. L'événement `xforms-binding-exception:`

Distribué comme indication d'une expression de liaison illégale, ou d'un attribut `model` qui échoue à pointer vers l'ID d'un élément `model`, ou d'un attribut `bind` qui échoue à pointer vers l'ID d'un élément `bind`, ou d'un attribut `submission` qui échoue à pointer vers l'ID d'un élément `submission`.

b. L'événement `xforms-link-exception`:

Distribué comme indication de l'échec dans la traversée du lien d'un attribut de liaison. Information de contexte : The URI that failed to load (xsd:anyURI).

c. L'événement `xforms-compute-exception`:

Distribué comme indication d'une erreur survenue au cours de l'évaluation XPath.

IV.Conclusion :

Dans cette partie nous avons étudié le langage XForms. Son architecture est bien adapté au modèle MVC.

Cette séparation apporte les avantages suivants :

* Réutilisation:

Les modules d'XForms peuvent être réutilisés indépendamment des informations collectées.

* Indépendance vis à vis des terminaux :

Les contrôles de l'interface utilisateur étant abstraits *(leurs caractéristiques génériques étant seules visibles),* ils peuvent être facilement représentés sur différents terminaux avec des capacités différentes.

* Accessibilité :

Les technologies XForms rendent possible l'utilisation de formulaires à partir d'un assistant personnel de poche, d'un téléphone mobile, d'un lecteur d'écran ou encore d'un ordinateur de bureau conventionnel, et cela sans perte de fonctionnalités pour l'utilisateur final.

Comme conclusion nous pouvons dire que :

- XForms est complet, portable, facile à utiliser *(réduit 70 % des scriptes).*

- XForms est une nouvelle technologie, une technologie en pleine évolution mais elle sera sûrement la technique de l'avenir dans tous les pages Web.

Chapitre 3:

Signature Numérique

I.Introduction: [SIG01]

Internet est en voie de devenir la principale plate-forme en vue du commerce et des communications au niveau mondial. La même ouverture qui a favorisé la croissance fulgurante d'Internet rend difficile la protection des transactions par le biais de celui-ci. Les gouvernements, les entreprises et les particuliers exigent des mécanismes qui non seulement garantissent l'intégrité de l'information transmise par Internet, mais qui offre le même niveau de confiance que les transactions sur papier.

Or les technologies actuelles dans la transmission de données sont insuffisantes pour la sécurisation des transactions commerciales sur le Web. La plupart des mécanismes de sécurité incorporés aux navigateurs, en général suffisant pour les transactions concernant des données "bas niveau", ne fournissent pas le niveau de sécurité et la flexibilité nécessaires pour la protection des transactions commerciales manipulant des données sensibles.

Aujourd'hui, la technologie SSL[1] protège les données lors du transfert, mais qu'en est-il une fois ces données transmises? Il est plus simple de pirater un serveur pour prendre des dizaines, voir des centaines de données non protégées plutôt que d'essayer de surveiller une seule connexion pour intercepter un numéro de carte bleue unique.

I.1. Modèle de base de sécurité: [SIG02]

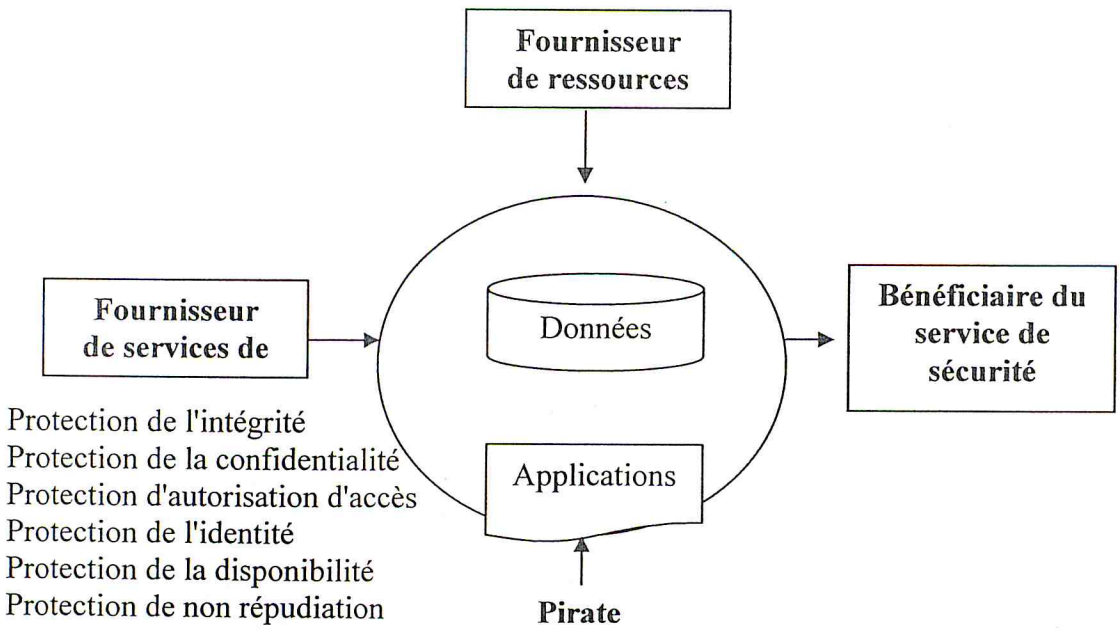


Figure 3.0 : *Exemple d'un modèle de sécurité de base*

* **Authentification**: Cela permet de déterminer de façon irrévocable l'utilisateur qui tente d'accéder à un système grâce à la confirmation de son identité.

* **Intégrité**: Cela signifie que vous savez que le message que vous lisez n'a pas été altéré, volontairement ou involontairement.

* **Non-répudiation** : l'expéditeur ne peut nier avoir envoyé le message et le destinataire ne peut nier l'avoir reçu. Simplement, la non répudiation signifie qu'une information ne peut être rejetée, tout comme avec les signatures manuscrites.

I.2. Cryptographie: [SIG04]

Les données lisibles et compréhensibles sans intervention spécifique sont considérées comme du *texte en clair*. La méthode permettant de dissimuler du texte en clair en masquant son contenu est appelée le *cryptage*. Le cryptage consiste à transformer un texte normal en charabia inintelligible appelé *texte chiffré*. Cette opération permet de s'assurer que seules les personnes auxquelles les informations sont destinées pourront y accéder. Le processus inverse de transformation du texte chiffré vers le texte d'origine est appelé le
décryptage.

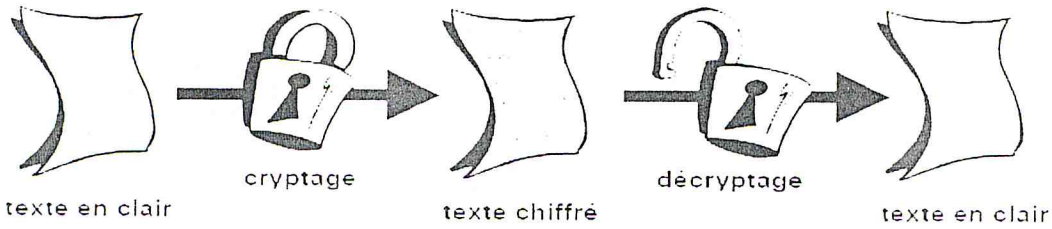


Figure 3.1 : *Principe de la cryptographie*

I.2. 1 Définition:

La *cryptograpie* est la science qui utilise les mathématiques pour le cryptage et le décryptage de données. Elle vous permet ainsi de stocker des informations confidentielles ou de les transmettre sur des réseaux non sécurisés (*tels que l'Internet)*, afin qu'aucune personne autre que le destinataire ne puisse les lire. Alors que la cryptographie consiste à sécuriser les données, la *cryptanalyse* est l'étude des informations cryptées, afin d'en découvrir le secret. La cryptanalyse classique implique une combinaison intéressante de raisonnement analytique, d'application d'outils mathématiques, de recherche de modèle, de

patience, de détermination et de chance. Ces cryptanalystes sont également appelés des *pirates*. La *cryptologie* englobe la cryptographie et la cryptanalyse.

I.2.2 Cryptographie Conventionnelle:

En cryptographie conventionnelle, également appelée cryptage de *clé secrète* ou de *clé symétrique*, une seule clé suffit pour le cryptage et le décryptage. La norme de cryptage de données *(DES)* est un exemple de système de cryptographie conventionnelle largement utilisé par le gouvernement fédéral des Etats-Unis.

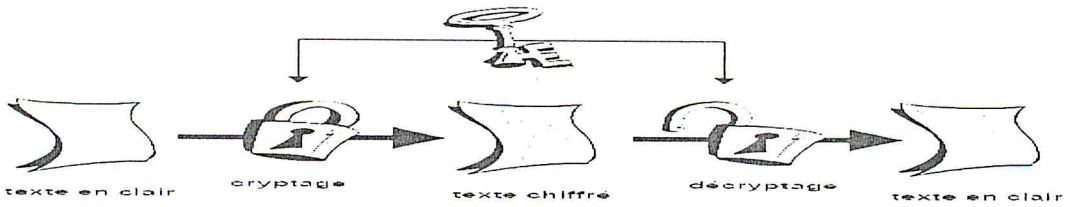


Figure 3.3 : Exemple de *Cryptographie conventionnelle*

Le cryptage conventionnel comporte des avantages. Il est très rapide. Mais, il s'avère particulièrement utile pour les données véhiculées par des *moyens de transmission* sécurisés. Toutefois, il peut entraîner des coûts importants en raison de la difficulté à garantir la confidentialité d'une clé de cryptage lors de la distribution.

Un expéditeur et un destinataire souhaitant communiquer de manière sécurisée à l'aide du cryptage conventionnel doivent convenir d'une clé et ne pas la divulguer. S'ils se trouvent à des emplacements géographiques différents, ils doivent faire confiance à un coursier, ou à tout autre moyen de communication sécurisé pour éviter la divulgation de la clé secrète lors de la transmission. Toute personne interceptant la clé lors d'un transfert peut ensuite lire, modifier et falsifier toutes les informations cryptées ou authentifiées avec cette clé.
La *distribution des clés* reste le problème majeur du cryptage conventionnel. Autrement dit, comment faire parvenir la clé à son destinataire sans qu'aucune personne ne l'intercepte?

I.2.3 Cryptographie à clé publique:

Les problèmes de distribution des clés sont résolus par la *cryptographie de clé publique*. Ce concept a été introduit par Whitfield Diffie et Martin Hellman en 1975. *(Il est maintenant prouvé que les services secrets britanniques avaient fait cette même découverte plusieurs années avant Diffie et Hellman et avaient protégé ce secret militaire).*

La cryptographie de clé publique est un procédé asymétrique utilisant une *paire* de clés pour le cryptage : une *clé publique* qui crypte des données et une *clé privée* ou *secrète* correspondante pour le décryptage. Vous pouvez ainsi publier votre clé publique tout en conservant votre clé privée secrète. Tout utilisateur possédant une copie de votre clé publique peut ensuite crypter des informations que vous êtes le seul à pourvoir lire. Même les personnes que vous ne connaissez pas personnellement peuvent utiliser votre clé publique.

D'un point de vue informatique, il est impossible de deviner la clé privée à partir de la clé publique. Tout utilisateur possédant une clé publique peut crypter des informations, mais est dans l'impossibilité de les décrypter. Seule la personne disposant de la clé privée correspondante peut les décrypter.

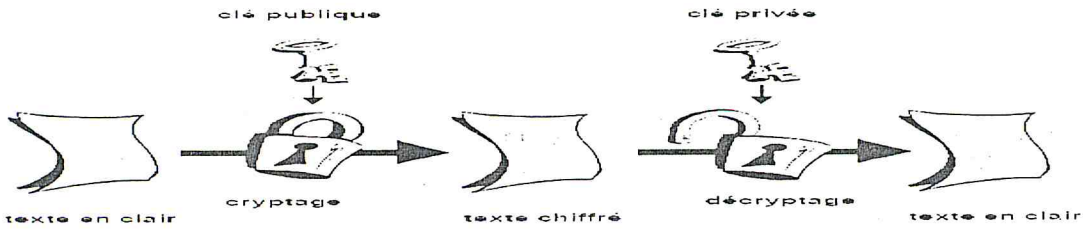


Figure 3.4 : *Exemple de Cryptographie asymétrique*

La cryptographie de clé publique présente un avantage majeur : en effet, elle permet d'échanger des messages de manière sécurisée sans aucun dispositif de sécurité. L'expéditeur et le destinataire n'ont plus besoin de partager des clés secrètes via une voie de transmission sécurisée. Les communications impliquent uniquement l'utilisation de clés publiques et plus aucune clé privée n'est transmise ou partagée

La cryptographie étant un composant fondamentale dans solution de sécurité, certains pensent par erreur qu'une solution sécurisé revient à utiliser uniquement la cryptographie. Ce qui n'est pas le cas.

I.3. Qu'est-ce qu'une signature électronique ? [SIG05]

Une signature numérique est un identificateur électronique comparable à la signature conventionnelle sur papier - elle est unique, vérifiable et seul le signataire peut la donner. La signature numérique peut être utilisée pour les messages chiffrés (cryptés) ou non.Tout comme les signatures manuscrites, les signatures électroniques sont utilisées pour identifier les auteurs/co-signataires d'un e-mail ou d'autres données électroniques. Les signatures

électroniques sont créées et vérifiées grâce aux certificats numériques. A l'heure actuelle, certaines communautés internationales élaborent une législation qui reconnaîtra et donnera une valeur légale aux signatures électroniques, leur donnant la même valeur que les signatures manuscrites.
Une signature peut être générée en utilisant une clé privée sur un bloc de données, afin de générer un bloc de données désigné comme signature. Le générateur de cette signature est appelé signataire. La signature ne peut plus alors être décryptée qu'en utilisant la clé publique di signataire, ce qui garanti donc son identité.

Mais cependant une question ce pose comment faire confiance a une signature numérique. Ce problème est résolu par l'incorporation des **certificats numériques**.

Pour signer des informations, pour opérer des transactions de façon sécurisée, vous devez avoir votre propre et unique certificat numérique. Ce dernier est délivré par une autorité de confiance appelée **autorité de certification**.

I.4. Qu'est ce qu'un certificat numérique? [SIG02]

C'est simplement un bloc de données contenant les informations qui permettent d'identifier un principal. Ces informations comprennent la clé publique, des informations sur le principale, les dates de validité du certificat, des informations sur l'émetteur du certificat et une signature gérée par l'émetteur.

Les certificats deviennent pratiques quand votre clé publique est envoyée aux autres entités. Elle leur permet d'identifier les informations qui ont été cryptées à l'aide de votre clé privée. Les certificats permettent aussi aux autres entités d'envoyé des informations que vous seul pourrez décrypter a avec votre clé privée. Sans un certificat, des pirates pourraient envoyer à une autre entité une clé publique faussement identifiée comme la vôtre, puis signer des messages vers cette entité à l'aide de la clé privée correspondant à la clé publique mal identifiée.

I.5. Autorité de certification:

Un certificat est signé par une autorité de certification tierce. Si elle indique qu'une clé publique associée un certificat vous appartient bien, l'entité destinataire peut être sûre que la clé publique est bien la votre et pas celle d'un pirate.

Naturellement, le même problème existe pour la fourniture initiale d'une clé publique associée à l'autorité de certification à une entité destinataire. Ce problème peut se résoudre en utilisant une autorité de certification commune *(par exemple, VeriSign)* dont la clé publique a été diffusée à un groupe d'utilisateur d'une manière sécurisée. Cette fourniture sûre en une fois de la clé publique d'une autorité de certification, qui sera utilisée pour vérifier les certificats de plusieurs principaux, est certainement plus simple que de tenter de fournir de manière sûre une clé publique pour chaque principale de groupe.

Des outils comme les navigateurs Web sont souvent préconfigurés avec les certificats de nombreuses autorités de certification courants et approuvées.

XML est aujourd'hui la technologie la plus puissante pour la transaction de données commerciales en permettant l'encryptions ainsi que la signature digitale des données, codées en XML. Ainsi, dans un scénario de workflow particulier où une signature digitale implique un certain niveau d'engagement, chaque participant pourra signer la portion du document partagé dont il doit assurer la responsabilité.

En partant de ces constatations le W3C à mis en place un groupe de travail (XML Signature WG) qui a définit plusieurs recommandations permettant de mettre en place la signature de document en XML.

II.XMLDsig :

Cette partie spécifie les règles de traitement et la syntaxe pour créer et représenter des signatures digitales XML.

II.1 Introduction à XMLDsig: [XSIG]

Standardiser par le W3C le 12 février 2002, la signature XML fournie des services d'intégrité, d'authentification du message et du signataire pour tous types de données, qui se trouvent dans le code XML ou bien ailleurs.

Une signature XML peut s'appliquer sur le contenu d'une ou plusieurs ressources. Les signatures enveloppées ou enveloppantes opèrent sur des données dans le même document XML que la signature ; les signatures détachées opèrent sur des données en dehors de l'élément signature.

Plus précisément, cette spécification définit un type d'élément signature XML et une application de signature XML ; une telle application doit spécifier une clé, un algorithme et des conditions de traitement.

*** L'espaces de nommage :**

L'URI de l'espace de nommage XML `[XML-ns]` qui doit être utilisé par les implémentations de cette spécification est

`xmlns="http://www.w3.org/2000/09/xmldsig#"`

Cet espace de nommage est également utilisé comme préfixe pour les identifiants d'algorithme utilisés par cette spécification.

II.2 Présentation de la signature et exemples :

II..2.1 Présentation : Les signatures XML sont représentées par l'élément `Signature` qui a la structure suivante:

```
<Signature ID?>
   <SignedInfo>
      <CanonicalizationMethod/>
      <SignatureMethod/>
      (<Reference URI? >   //une ref de l'objet a signé
           (<Transforms>)?
           <DigestMethod>
           <DigestValue>
      </Reference>)+
   </SignedInfo>
   <SignatureValue>
   <KeyInfo>)?
   (<Object ID?>)*
 </Signature>
```

Avec les caractères de cardinalité :
« ? » Dénote zéro ou une occurrence,
« + » Une ou plusieurs occurrences
« * » Zéro ou plusieurs occurrences

- La balise <Signature> est la balise "mère" de la structure de la signature XML;
- La balise <SignedInfo> contient la liste des ressources à signer, identifiées par une URI;
- L'élément (CanonicalizationMethod) contient une référence sur l'algorithme de canonisation;
- L'élément (SignatureMethod) contient l'algorithme utilisé pour signer les données;
- La balise <Reference> contient une URI qui pointe sur une ressource que l'on veut signer;
- L'élément (Tranforms) contient la liste des modifications que l'on va effectuer sur le contenu des ressources avant d'effectuer la condensation;
- L'élément (DigestMethod) définit l'algorithme de condensation;
- L'élément (DigestValue) contient la valeur de la condensation;
- L'élément (SignatureValue) contient la valeur de la signature;
- L'élément (KeyInfo) spécifie des informations sur la clé à utiliser pour valider la signature.
- L'élément (Object) permet d'insérer un fragment XML dans la signature XML.

Selon l'emplacement de l'objet de données a signé on distinguera trois cas possibles :

1. cas : l'objet a signés se trouvant dans l'élément `<object>` a l'intérieure de l'élément `<signature>` dans ce cas la signature est **enveloppante .**

2. cas : l'objet a signés est l'élément racine de la signature, la signature est **enveloppées.**

3. cas : l'objet a signés est a l'extérieure de l'élément `Signature`, Par conséquent, la signature est **détachée** du contenu qu'elle signe.

II.2.2 Un exemple simple (`Signature`, `SignedInfo`, `Méthodes` et `Références`)

L'exemple suivant est une signature détachée d'un contenu HTML4 dans une spécification XML.

```
[s01]<Signature Id="MyFirstSignature" xmlns="http://www.w3.org/2000/09/xmldsig#">
[s02]   <SignedInfo>
[s03]    <CanonicalizationMethod
                Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315"/>
[s04]    <SignatureMethod
                Algorithm="http://www.w3.org/2000/09/xmldsig#dsa-sha1"/>
[s05]    <Reference URI="http://www.w3.org/TR/2000/REC-xhtml1-20000126/">
[s06]       <Transforms>
[s07]         <Transform
                   Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315"/>
[s08]       </Transforms>
[s09]       <DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
[s10]       <DigestValue>j6lwx3rvEPO0vKtMup4NbeVu8nk=</DigestValue>
[s11]    </Reference>
[s12]   </SignedInfo>
[s13]   <SignatureValue>MC0CFFrVLtRlk=...</SignatureValue>
[s14]   <KeyInfo>
[s15a]     <KeyValue>
[s15b]       <DSAKeyValue>
[s15c]         <P>...</P><Q>...</Q><G>...</G><Y>...</Y>
[s15d]       </DSAKeyValue>
[s15e]     </KeyValue>
[s16]   </KeyInfo>
[s17]</Signature>
```

L'URI en gras contient une référence vers l'objet de donnés a signé.

[s02-12]: L'élément obligatoire **SignedInfo** est véritablement l'information qui sera signée.

[s03]: L'élément **CanonicalizationMethod** donne l'algorithme qui est utilisé pour canoniser l'élément SignedInfo avant qu'il ne soit prétraité en tant que partie de l'opération de signature.

[s04]: L'élément **SignatureMethod** donne l'algorithme qui est utilisé pour convertir l'élément SignedInfo canonisé en un élément **SignatureValue**. C'est la combinaison d'un algorithme de prétraitement et d'un algorithme dépendant d'une clé, et éventuellement d'autres algorithmes comme un algorithme de remplissage, par exemple RSA-SHA1. Les noms des algorithmes sont signés .l'architecture permet également d'utiliser des algorithmes arbitraires spécifiés par l'utilisateur.

[s05-11]: Chaque élément **Reference** inclut la méthode de prétraitement **<DigestMethod>** et la valeur de prétraitement résultante calculées à partir d'un objet de données **<DigestValue>**. L'élément Reference peut également inclure les transformations qui ont produit l'entrée pour l'opération de prétraitement.

[s14-16]: L'élément **KeyInfo** indique la clé qui doit être utilisée pour valider la signature. Les formes d'identification possibles comprennent des certificats, des noms de clé et des algorithmes et renseignements pour l'agrément des clés -- nous n'en définissons que quelques unes.

II.3 Les règles de traitement :

Cette section décrit les opérations à effectuer pour créer et valider une signature XML.

II.3.1 La génération principale :

La génération d'une signature XML se fait en deux étapes importantes :

A- La génération d'une référence :

Pour chaque objet de données en cours de signature :

1- Appliquer l'élément Transforms, comme déterminé par l'application, sur l'objet de données ;

2- Calculer la valeur prétraitée sur l'objet de données resultant. *(DigestValue)*

3- Créer un élément Reference, incluant l'identification de l'objet de données, tout élément de transformation, l'algorithme de prétraitement DigestMethod .

B- La génération de la signature :

1- Créer l'élément **SignedInfo** avec les éléments SignatureMethod, Canonicalization et Reference.

2- Canoniser puis calculer l'élément **SignatureValue** sur l'élément SignedInfo, en fonction

des algorithmes de `SignatureMethod`.

3- Construire l'élément `Signature` qui inclut les éléments `SignedInfo`, le ou les Objets. `KeyInfo` *(si requis)* et `SignatureValue`.

II..3.2 La validation principale :

Les étapes requises de la validation principale incluent :

A. La validation de signature :

- Obtenir les informations de clé à partir de l'élément `KeyInfo` ou d'une source externe ;
- Obtenir la forme canonique de l'élément `SignatureMethod` en utilisant l'élément `CanonicalizationMethod`, puis utiliser le résultat *(et les informations de clé de l'élément `KeyInfo` précédemment obtenues)* pour confirmer la valeur de l'élément `SignatureValue` sur l'élément `SignedInfo`.

Elle se fait en inversant tous simplement les étapes de signatures du document.

De cette manière si une erreur c'est produite dans la signature on ne sera pas obligé d'aller plus loin.

Par exemple si la signature n'est pas valide, pas besoin d'aller vérifier les valeurs des condensées.

B. La validation de référence :

1- Canoniser l'élément `SignedInfo` en fonction de l'élément `CanonicalizationMethod` dans `SignedInfo` ;

2- Pour chaque élément `Reference` dans `SignedInfo` :

a). Obtenir les données à prétraiter. *(Par exemple, l'application de signature peut résoudre l'`URI` et exécuter l'élément `Transforms` fourni par le signataire dans l'élément `Reference`)*;

b). Prétraiter l'objet de données résultant en utilisant l'élément `DigestMethod` spécifié dans son élément `Reference` ;

c). Comparer la valeur condensée générée avec la valeur de `DigestValue` dans l'élément `Reference` de `SignedInfo` ; s'il y a une quelconque divergence, la validation échoue.

À noter que l'élément `SignedInfo` est canonisé lors de la 1e étape. L'application doit s'assurer que l'élément CanonicalizationMethod n'a pas d'effets secondaires indésirables, tels que la réécriture des URI *(l'application de signature et de validation traiteront le même document XML).*

Pour le calcule des condensés *(DigestValue),* les références n'ont pas besoin d'être canoniser, puise qu'il ne seront pas traitées autant que documents XML,mains autant que série d'octets,ce qui évite les différences de traitements des différent analyseurs XML comme SAX,JDOM…

II.4 Syntaxe de la structure principale de la signature :

Cette section fournit la syntaxe et les caractéristiques principales d'une signature.

II.4.1 L'élément `Signature` :

L'élément `Signature` est l'élément racine d'une signature XML, il contient tous les éléments devant être signés, ou plutôt des références vers ces derniers.

Définition de schéma :

```
<element name="Signature" type="ds:SignatureType"/>
<complexType name="SignatureType">
  <sequence>
    <element ref="ds:SignedInfo"/>
    <element ref="ds:SignatureValue"/>
    <element ref="ds:KeyInfo" minOccurs="0"/>
    <element ref="ds:Object" minOccurs="0" maxOccurs="unbounded"/>
  </sequence>
  <attribute name="Id" type="ID" use="optional"/>
</complexType>
```

- L'élément **`SignatureValue`** contient la valeur réelle de la signature digitale ; il est toujours codé en utilisant l'algorithme base64.

- La structure de l'élément **`SignedInfo`** inclut l'algorithme de canonisation, un algorithme de signature, et une ou plusieurs références. L'élément `SignedInfo` peut contenir un attribut optionnel ID qui lui permet d'être référencé par d'autres signatures et objets.

Définition de schéma :

```
<element name="SignedInfo" type="ds:SignedInfoType"/>
<complexType name="SignedInfoType">
  <sequence>
    <element ref="ds:CanonicalizationMethod"/>
    <element ref="ds:SignatureMethod"/>
    <element ref="ds:Reference" maxOccurs="unbounded"/>
  </sequence>
  <attribute name="Id" type="ID" use="optional"/>
</complexType>
```

* L'élément **`CanonicalizationMethod`** est un élément requis qui spécifie l'algorithme de canonisation appliqué à l'élément `SignedInfo` avant de réaliser les calculs de signature. Cet élément utilise la structure générale pour les algorithmes. Les implémentations DOIVENT gérer les algorithmes de canonisation.

* L'élément **`SignatureMethod`** est un élément requis qui spécifie l'algorithme utilisé pour la génération et la validation de la signature. Cet algorithme identifie toutes les fonctions cryptographiques impliquées dans l'opération de signature (par exemple, le hachage, les algorithmes à clé publique, les codes MAC, le remplissage, etc.). Alors qu'il n'y a qu'un seul identifiant, celui-ci peut spécifier un format contenant plusieurs valeurs de signature

distinctes.

* L'élément **Reference** est un élément qui peut apparaître une ou plusieurs fois. Il spécifie un algorithme de prétraitement et une valeur condensées et, facultativement, un identifiant de l'objet en train d'être signé, le type de l'objet et/ou une liste de transformations à appliquer avant le prétraitement. L'identification *(URI)* et les transformations décrivent la manière dont le contenu condensé (*c.-à-d., la valeur d'entrée pour la méthode de prétraitement)* a été créé.

- L'élément **KeyInfo** est un élément facultatif qui permet au(x) destinataire(s) d'obtenir la clé nécessaire à la validation de la signature. L'élément `KeyInfo` peut contenir des clés, des noms, des certificats et d'autres informations concernant la gestion de clés publiques, telles que des données concernant la diffusion en ligne ou l'agrément des clés.
Si l'élément `KeyInfo` est omis, le destinataire est sensé pouvoir identifier la clé en fonction du contexte de l'application.
* Un élément `X509Data` dans `KeyInfo` contient un ou plusieurs identifiants de clés ou certificats X509 (*ou identifiants de certificats ou une liste de révocation)*.
* L'élément `PGPData` dans `KeyInfo` est utilisé pour donner des informations liées aux couples de clés publiques PGP et aux signatures basées sur de telles clés. La valeur de `PGPKeyID` est une séquence binaire codée en base64 contenant un identifiant de clé PGP publique.

* L'élément `SPKIData` dans `KeyInfo` est utilisé pour transmettre des informations liées aux couples de clés publiques SPKI, aux certificats et autres données SPKI. L'élément `SPKISexp` est l'expression codée en base64 d'une S-expression SPKI canonique

* L'élément `MgmtData` dans `KeyInfo` est une valeur de chaîne de caractères utilisée pour transmettre des données concernant la diffusion en ligne ou l'agrément de clés. L'utilisation de cet élément N'EST PAS RECOMMANDÉE.

- L'élément **Object** est un élément facultatif qui peut apparaître une ou plusieurs fois. Lorsqu'il est présent, cet élément peut contenir tout type de données. Cet élément `Object` peut inclure un type MIME facultatif, un ID, et des attributs d'encodage.

II.5 Les identifiants d'algorithme :

Cette section identifie les algorithmes utilisés dans la spécification de la signature digitale XML.

Les condensés des messages :

SHA-1

Identifiant : `http://www.w3.org/2000/09/xmldsig#sha1`
`<http://www.w3.org/2000/09/xmldsig`

Les algorithmes de signature :

Les algorithmes de signature admettent deux paramètres implicites : leur matériel de gestion de clé déterminé à partir de l'élément `KeyInfo` et le flux d'octets de sortie de l'élément `CanonicalizationMethod`.

L'algorithme DSA

Identifiant : `http://www.w3.org/2000/09/xmldsig#dsa-sha1`

L'algorithme PKCS1 (RSA-SHA1)

Identifiant : `http://www.w3.org/2000/09/xmldsig#rsa-sha1`

Les algorithmes de canonisation :

Divers algorithmes de canonisation transcodent d'un encodage non Unicode vers Unicode.
L'identifiant pour le XML canonique REQUIS (sans commentaires) :

`http://www.w3.org/TR/2001/REC-xml-c14n-20010315`

L'identifiant pour le XML canonique avec commentaires :

`http://www.w3.org/TR/2001/REC-xml-c14n-20010315#WithComments`

Incontestablement, la signature de document en XML prendra de plus en plus de place dans le commerce électronique et tout autre domaine où l'on veut avoir confiance dans les documents auxquels on accède. En plus des apports techniques que permet XML dans la signature numérique, on se retrouve avec une technologie normalisée ne venant pas d'une entreprise unique, ce qui à son importance, car la norme a pu être réfléchie en dehors de toutes considérations personnelles et mercantiles.

Le manque d'outil intégrant la signature XML est du à la relative jeunesse de la recommandation. On peut espérer voir d'ici quelques temps les navigateurs Web intégrer des modules permettant de vérifier les signatures XML comme ils peuvent aujourd'hui déjà se connecter sur des sites en HTTPS (HTTP sur une connexion sécurisée par SSL).

Ce pendant cette Norme ne répond pas à tous les besoins d'une signature numérique comme la non-repudiation a long terme, c'est pourquoi une extension à été établit pour

combler les brèches de cette spécification. Cette extension sera abordée dans le reste de ce chapitre.

III.XAdES :

III.1 Introduction à XAdES : *(Note du W3C du 20 février 2003)*

La signature électronique évoluée XML (XAdES) :Cette note prolonge la spécification de la syntaxe et du traitement XML Signature de IETF/W3C [XMLDSIG] dans le domaine de la non répudiation, en définissant des formats XML pour les signatures électroniques évoluées qui restent valides pendant une longue période, et qui incorporent des informations supplémentaires utiles dans les situations d'utilisations courantes. Cela comprend la preuve de leur validité, même si le signataire ou le tiers vérifiant essayent ensuite de nier (répudier) la validité de la signature.

Cette extension rajoute six formes supplémentaires à [XMLDSIG] :

1. XAdES.
2. XAdES-T : la signature électronique évoluée XML avec estampille
3. XAdES-C : la signature électronique évoluée XML avec données de validation complètes.
4. XAdES-X : la signature électronique évoluée XML avec données de validation étendues.
5. XAdES-X-L : la signature électronique évoluée XML avec données de validation étendues incorporées pour le long terme.
6. XAdES-A : la signature électronique évoluée XML avec données de validation pour l'archivage.

Mais seulement les deux premières seront détaillées (i.e XAdES et XAdES-T).

Note:

Les autres spécifications ont certes leur importance, ils permettent d'offrir plus de signature à l'utilisateur comme pour le cas de XAdES-A qui archive la totalité du certificat utilisé dans la signature. Mais dans notre cas nous nous somme arrêté a XAdES-T qui remplie parfaitement les exigences d'une signature numérique évoluée, et surtout la non-répudiation qui n'été pas vérifiée par la spécification XMLDSIG.

III.2. XAdES :

Cette forme fournit une authentification et une protection d'intégrité minimales et qui satisfait aux obligations légales pour les signatures électroniques évoluées. Ne fournit cependant pas la non-répudiation de son existence. Cette forme rajoute des éléments de qualifications à l'element `<object>` de `XMLDSIG`

Cette extension spécifie deux types principaux de propriétés : les propriétés signées et celles non signées. Les premières sont des objets-données supplémentaires qui sont également sécurisés par la signature produite par le signataire sur l'élément `ds:SignedInfo`, ce qui implique que le signataire détient ces objets-données, en calcule un hachage pour tous et génère l'élément `ds:Reference` correspondant. Les propriétés non signées sont des objets-données rajoutés par le signataire, par le vérificateur ou par des tiers après la production de la signature. Ci-dessous on montre une signature électronique évoluée XML :

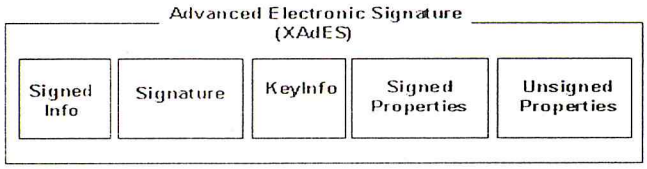


Figure 3.5: *Illustration d'une signature XAdES*

Une signature XAdES se construira à partir de [XMLDSIG] par incorporation d'un élément `ds:Object` qui se comportera comme le récipient pour l'ensemble entier des propriétés de qualification. Certaines d'entre elles seront signées, et d'autres ne le seront pas.

III.3. Aperçu de la syntaxe :

Cette partie introduit la syntaxe concernant l'ajout d'informations de qualification à une signature XML.

III.3.1. L'élément QualifyingProperties : L'élément `QualifyingProperties` se comporte comme un conteneur pour toutes les informations de qualification devant être ajoutées à une signature XML. L'élément a la structure suivante :

```
<xsd:element name="QualifyingProperties type="QualifyingPropertiesType"/>
   <xsd:complexType name="QualifyingPropertiesType">
      <xsd:sequence>
        <xsd:element name="SignedProperties" type="SignedPropertiesType"
                     minOccurs="0"/>

    <xsd:elementname="UnsignedProperties"type="UnsignedPropertiesType"
                   minOccurs="0"/>
      </xsd:sequence>
      <xsd:attribute name="Target" type="xsd:anyURI" use="required"/>
      <xsd:attribute name="Id" type="xsd:ID" use="optional"/>
   </xsd:complexType>
```

Les propriétés de qualification se répartissent en propriétés qui sont cryptographiquement rattachées à *(i.e., signées par)* la signature XML (**SignedProperties**) et en celles qui ne sont pas cryptographiquement rattachées à la signature XML (**UnsignedProperties**).

III.3.1.1 **L'élément SignedProperties** : Identifie un certain nombre de propriétés qui sont collectivement signées par la signature [XMLDSIG].ci-dessous le schéma XML:

```
    <xsd:element name="SignedProperties" type="SignedPropertiesType" />
    <xsd:complexType name="SignedPropertiesType">
      <xsd:sequence>
        <xsd:element name="SignedSignatureProperties"
          type="SignedSignaturePropertiesType"/>
          <xsd:element name="SignedDataObjectProperties"
            type="SignedDataObjectPropertiesType" minOccurs="0"/>
      </xsd:sequence>
      <xsd:attribute name="Id" type="xsd:ID" use="optional"/>
    </xsd:complexType>
```

L'élément `SignedProperties` contient :

* Des propriétés qui qualifient la signature [XMLDSIG] en question ou le signataire. Elles sont incluses en contenu de l'élément **SignedSignatureProperties**.

* Des propriétés qui qualifient certains des objets-donnés signés. Ces propriétés apparaissent en contenu de l'élément **SignedDataObjectProperties**.

a. L'élément SignedSignatureProperties : Cet élément contient des propriétés qui qualifient la signature XML spécifiée par l'attribut `Target` de l'élément conteneur `QualifyingProperties`.

```
<xsd:element name="SignedSignatureProperties"
type="SignedSignaturePropertiesType" />

<xsd:complexType name="SignedSignaturePropertiesType">
  <xsd:sequence>
    <xsd:element name="SigningTime" type="xsd:dateTime"/>
    <xsd:element name="SigningCertificate" type="CertIDListType"/>
    <xsd:element name="SignaturePolicyIdentifer"
     type="SignaturePolicyIdentifierType"/>
    <xsd:element name="SignatureProductionPlace"
type="SignatureProductionPlaceType"
     minOccurs="0"/>
    <xsd:element name="SignerRole" type="SignerRoleType"
minOccurs="0"/>
  </xsd:sequence>
</xsd:complexType>
```

* La propriété **SigningTime** spécifie le moment auquel le signataire a réalisé le processus de signature.

* La propriété **SigningCertificate**: contient des références aux certificats et aux valeurs de condensé calculé sur elles.

* La propriété **SignaturePolicyIdentifier**: identifie La politique de signature qui défini un ensemble de règles pour la création et la validation d'une signature électronique.

* La propriété **SignatureProductionPlace** : Indique l'endroit prétendu dans lequel le signataire se trouvait au moment de la création de la signature.

* La propriété **SignerRole** : Indique la position du signataire dans une société ou une organisation

b. L'élément SignedDataObjectProperties : Contient des propriétés qui qualifient l'objets de données a signés.

III.3.1.2 L'élément UnsignedProperties : L'élément UnsignedProperties contient un certain nombre de propriétés qui ne sont pas signées par la signature [XMLDSIG].

```
<xsd:element name="UnsignedProperties" type="UnsignedPropertiesType" />
  <xsd:complexType name="UnsignedPropertiesType">
    <xsd:sequence>
      <xsd:element name="UnsignedSignatureProperties"
        type="UnsignedSignaturePropertiesType" minOccurs="0"/>
        <xsd:element name="UnsignedDataObjectProperties"
          type="UnsignedDataObjectPropertiesType" minOccurs="0"/>
    </xsd:sequence>
    <xsd:attribute name="Id" type="xsd:ID" use="optional"/>
  </xsd:complexType>
```

- L'élément `UnsignedProperties` contient des:

Propriétés qui qualifient la signature XML ou le signataire **`UnsignedSignatureProperties`**.

Propriétés qui qualifient certains des objets-données signés

`UnsignedDataObjectProperties`.

a. L'élément UnsignedSignatureProperties :

Le contenu de cet élément n'est pas couvert par la signature XML. C'est dans cet élément qu'on se passe la différence entre les six différentes formes vues précédemment, Chacune d'elle va y rajouter un élément précis pour donner plus de précision et de sécurité, la forme XAdES contient que l'élément **`CounterSignature`**.

Le schéma XML de l'élément UnsignedSignatureProperties est montré dans la page suivante :

```
<xsd:element name="UnsignedSignatureProperties"
             type="UnsignedSignaturePropertiesType"/>
   <xsd:complexType name="UnsignedSignaturePropertiesType">
     <xsd:sequence>
       <xsd:element name="CounterSignature" type="CounterSignatureType"
                    minOccurs="0" maxOccurs="unbounded"/>
       <xsd:element name="SignatureTimeStamp" type="TimeStampType"
         minOccurs="0" maxOccurs="unbounded"/>
       <xsd:element name="CompleteCertificateRefs"
         type="CompleteCertificateRefsType" minOccurs="0"/>
       <xsd:element name="CompleteRevocationRefs"
         type="CompleteRevocationRefsType" minOccurs="0"/>
       <xsd:choice>
         <xsd:element name="SigAndRefsTimeStamp" type="TimeStampType"
           minOccurs="0" maxOccurs="unbounded"/>
         <xsd:element name="RefsOnlyTimeStamp" type="TimeStampType"
           minOccurs="0" maxOccurs="unbounded"/>
       </xsd:choice>
       <xsd:element name="CertificateValues" type="CertificateValuesType"
         minOccurs="0"/>
       <xsd:element name="RevocationValues" type="RevocationValuesType"
         minOccurs="0"/>
       <xsd:element name="ArchiveTimeStamp" type="TimeStampType"
         minOccurs="0" maxOccurs="unbounded"/>
     </xsd:sequence>
 </xsd:complexType>
```

* L'élément **`CounterSignature`** : Certaines signatures électroniques peuvent n'être valides que si elles portent plus d'une signature. C'est généralement le cas pour la signature d'un contrat entre deux parties. L'ordonnancement des signatures peut ou non revêtir une importance.

Le contenu de l'élément `CounterSignature` est la signature de l'élément `SignatureValue`.

L'élément `SignatureTimeStamp` encapsule l'estampille sur l'élément `ds:SignatureValue`.

III.4.1 Le contenu de la forme XAdES-T : Le signataire ou le vérificateur peuvent construire une signature XAdES-T en ajoutant à la signature XAdES existante un élément XML qui encapsule une estampille sur la valeur de la signature digitale [XMLDSIG], générée par une autorité d'estampillage pour prouver que la signature électronique a été effectuée avant ce moment.

Cette forme rajoute L'élément `SignatureTimeStamp` à l'élément `UnsignedSignatureProperties`

III.4.2 A quoi sert cette forme ?

Une propriété importante des signatures de longue durée c'est que la validité d'une signature, qui a été validée une fois, devra perdurer des mois ou des années après.

Un signataire, un vérificateur ou les deux peuvent être tenus de fournir, à la demande, la preuve que la signature digitale a été créée ou vérifiée pendant les périodes de validité de tous les certificats qui constituent le chemin de certification. Auquel cas, le signataire, le vérificateur ou les deux seront également tenus d'apporter la preuve qu'aucun des certificats des utilisateurs et des autorités de certification utilisés n'a été révoqué lors de la création ou de la vérification de la signature.

Il ne serait vraiment pas acceptable de considérer une signature comme étant invalide parce que les clés ou les certificats auront été compromis par la suite. Il est donc nécessaire de pouvoir démontrer que la clé de la signature était valide au moment de la création de la signature pour fournir la preuve sur le long terme de la validité d'une signature.

L'estampillage par une autorité d'estampillage (TSA) peut apporter cette preuve. On obtient une estampille en envoyant la valeur de hachage des données en question à l'autorité d'estampillage. L'estampille qui est retournée est un objet-données signé contenant la valeur de hachage, l'identité de l'autorité d'estampillage et le moment de l'estampillage. C'est la preuve que les données en question existaient avant l'heure d'estampillage.

L'estampillage d'une signature électronique (XAdES), avant la révocation de la clé privée du signataire et avant l'expiration du certificat, apporte la preuve que la signature a été créée alors que le certificat était valide et avant sa révocation.

- Si un destinataire veut détenir une signature électronique valide, il devra s'assurer avoir obtenu une estampille valide pour celle-ci, avant que cette clé *(et toutes les clés impliquées dans*

la validation) ne soit révoquée.

Il est important de remarquer que les signatures peuvent être générées « hors ligne » et être estampillée plus tard par n'importe qui, par exemple, par le signataire ou tout destinataire concernés par la valeur de la signature. L'estampille peut donc être fournie par le signataire en même temps que l'objet-données signé, ou bien être fournie par le destinataire à la suite de la réception de l'objet-donnée signé.

Note importante :
La politique de validation de la signature peut spécifier un intervalle de temps maximal autorisé entre l'heure indiquée par l'élément `SigningTime` et celle indiquée par l'élément `SignatureTimeStamp`. Si ce délai est dépassé, alors la signature électronique est considérée comme invalide.

Remarque :
L'estampille XAdES-T devrait être créée à un instant proche de celui de la création de la signature XAdES, pour offrir une protection contre la répudiation.

Le signataire devra fournir au moins la forme XAdES, mais, dans certains cas, il peut fournir la forme XAdES-T et. Si le signataire ne fournit pas la forme XAdES-T, le vérificateur devra soit créer la forme XAdES-T à première réception d'une signature électronique, soit conserver un enregistrement sécurisé de l'heure courante accompagnant la forme XAdES.

IV.Conclusion:

- Nous avons abordé dans ce chapitre la sémantique et la syntaxe de la signature digitale des documents XML.

- La signature XML évolué est importante dans les transactions basé sur qui se produisent sur un réseau donné, elle peut être aussi sûre que signature sur papier *(si ce n'est plus)* grâce a un ensemble de politique et une série de vérifications et de validations et aussi par l'intervention d'autorités de certification pour donner une garantie de confiance.

Chapitre 4:

Démarche et Conception de l'outi

I.Introduction:

Dans La précédente partie nous avons abordé brièvement ce qu'était un système d'échange de formulaire, ainsi quelques standards de langages de balisage utilisés dans la création de ces derniers. Nous avons aussi expliqué pourquoi nous avons choisi le standard XForms.

Nous avons vus par la suite un rappel de la signature électronique et de ces exigences en matière de sécurité, en concluant par la définition et la syntaxe de la signature XML *XMLdSIG* et de son extension *XAdES.*

Dans le présent chapitre, nous présenterons toutes les étapes nécessaires pour la conception et la réalisation de notre système.

Il sera divisé en trois parties:

- Présentation de la problématique qui a induit au développement de l'outil.
- Objectifs à atteindre.
- Démarches de développement du système.

II.Présentation de la problématique:

Dans la vie d'une entreprise l'échange d'information demeure vital, que ce soit dans le domaine administratif ou bien commercial. Ces information, dans la plupart du temps, sont échangées sous forme de formulaires *(Demande d'emploi, Retrait d'un extrait de naissance, Achat de produit...etc.)*

Mais cependant il y a beaucoup d'obstacles qui ralentissent ou mettent en péril la circulation de ces informations.

Les problèmes existant dans l'échange de formulaires sont:

- Dans le cas des formulaires papier, cela demande une certaine gestion qui peut s'avérer énormément coûteuse en papier, en temps, et en argents que les employés ou clients s'en passerait volontiers.
- C'est vrai que des solutions ont été apportées en remplaçant les formulaires papiers par des formulaires électroniques, mais cependant la conception de formulaires électroniques n'est pas à la porté de tout le monde, elle demande une certaine connaissance en matière de programmation.
- Ceci dit les différentes entreprises sont entrains de s'interconnecter, alors comment garantir une compatibilité des données échangées.
- Un autre problème majeur qui se pose est comment garantir un climat de sécurité et de confiance pour tous.

III.Objectifs:

L'objectif de notre objet se récapitule comme suite:

- Offrir une interface de création de formulaires électroniques qui ne requière aucune connaissance de terme de programmation, l'outil doit générer automatiquement le code de formulaires.
- L'interface doit être ergonomique qui donnera une utilisation très facile.
- Permettre une compatibilité d'information en utilisant une norme de langage de formulaires qui est le XForms.
- Un tel système doit fonctionner dans un réseau en mode Client/Serveur, ce qui implique que le système devra se diviser en deux modules: module Serveur et module Client.
- Pour offrir un climat de sécurité il serait primordial que le système insert une signature numérique pour garantir l'authenticité, l'intégrité et la non répudiation.
- Le système devra naturellement vérifier la validité de la signature numérique.
- A la fin on devra stocker les informations dans une base de donnée.

IV.Démarches de développement du système:

Avant de développer un logiciel il est nécessaire de s'organiser et de diviser notre travail en plusieurs étapes, pour cela des méthodes de conception ont été développées.

Pour la conception de notre projet nous utiliserons la notation UML qui n'est pas une méthode mais plutôt un formalisme, en utilisant le modèle en cascade.
Ce dernier décrit le cycle de vie d'un logiciel par une suit de phases décrites dans le schéma suivant:

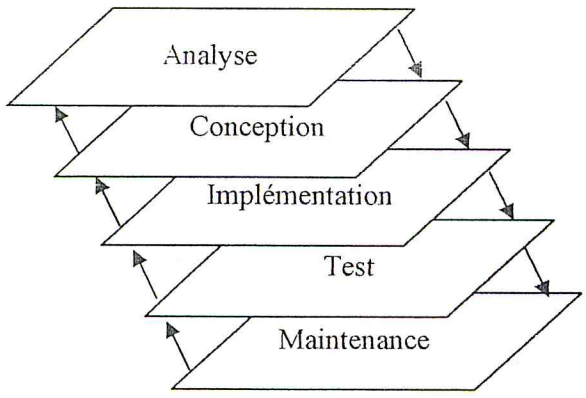


Figure 4.0 : *Le modèle en cascade* .

V.Analyse :

V.1 Analyse des besoins:

Le but de cette étape est la description des 4 parties suivantes:
- Fonctions du système.
- Utilisateurs du système.

Cette étape définit le comportement du système par des diagrammes de cas d'utilisation et de scénarios.

V.2 Les cas d'utilisation:

Les cas d'utilisation (use cases) ont été formalisés par Ivan Jacobson. Les cas d'utilisation décrivent sous la forme d'action et réaction, le comportement d'un système du point de vue d'un utilisateur. Ils permettent de définir les limites du système et les relations entre le système et l'environnement.

Un cas d'utilisation est une manière spécifique d'utiliser un système. C'est l'image d'une fonctionnalité du système, déclenchée en réponse a la simulation d'un acteur externe.

V.2.1 Les acteurs:

Ils représentent toute entité interagissant avec le système.
Comme notre système doit se diviser en deux sous système distinct, Client et Serveur, on dispose de deux acteurs, le créateur du formulaire et le client qui doit remplir le formulaire.

V.2.2 Cas d'utilisation généraux:

A cause de la distinction des opérations exécutées sur les deux sous systèmes, il est primordial de diviser les cas d'utilisation en deux parties
- Se qui se passe dans la partie Serveur.
- Se qui se passe dans la partie Client.

a- Partie serveur:

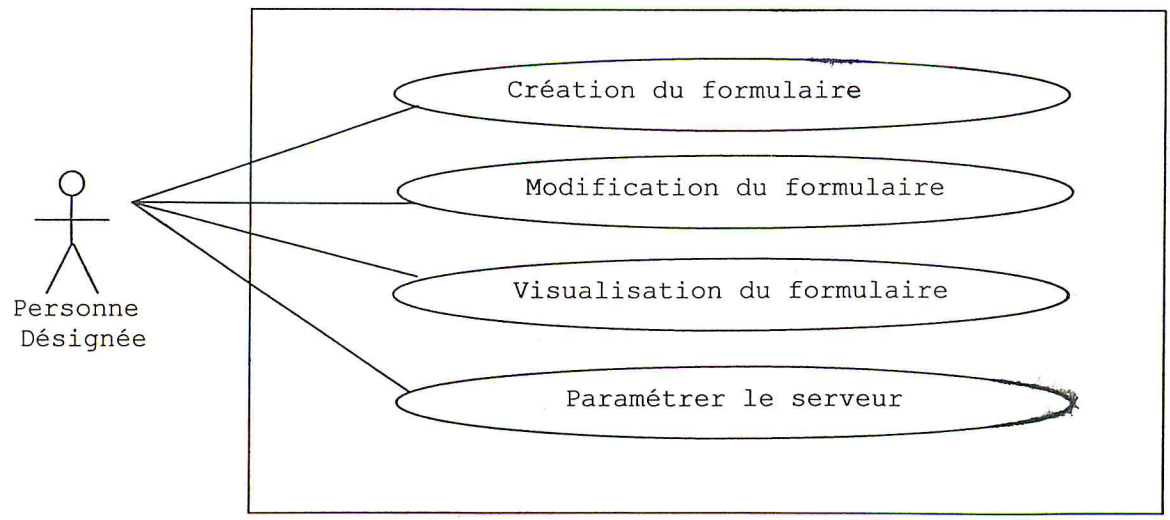


Figure 4.1 : *Diagramme des cas d'utilisation généraux de la partie serveur*

b- Partie client:

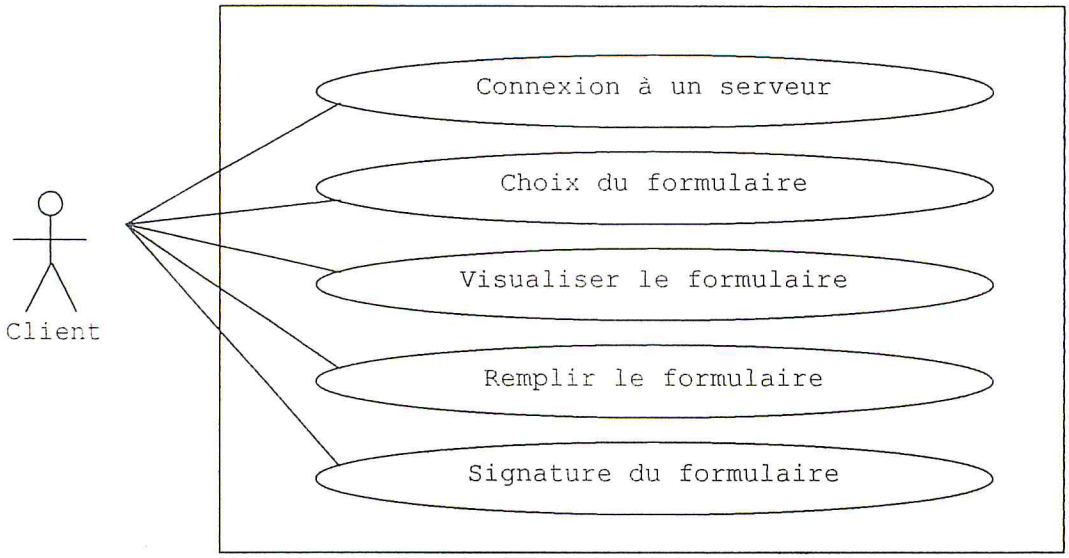


Figure4.2 : *Diagramme des cas d'utilisation généraux de la partie Client*

Nous détaillerons par la suite chaque cas d'utilisation des deux précédentes partie:

V.2.1.1 Partie Serveur:

1) Création de formulaire:

La création du formulaire doit se faire par une personne qualifiée et désignée.

Cette dernière crée le document en insérant des commandes IU *(Interfaces Utilisateur)* : champs de textes, boutons, labels,... etc. en désignant leur emplacement et leurs propriétés.

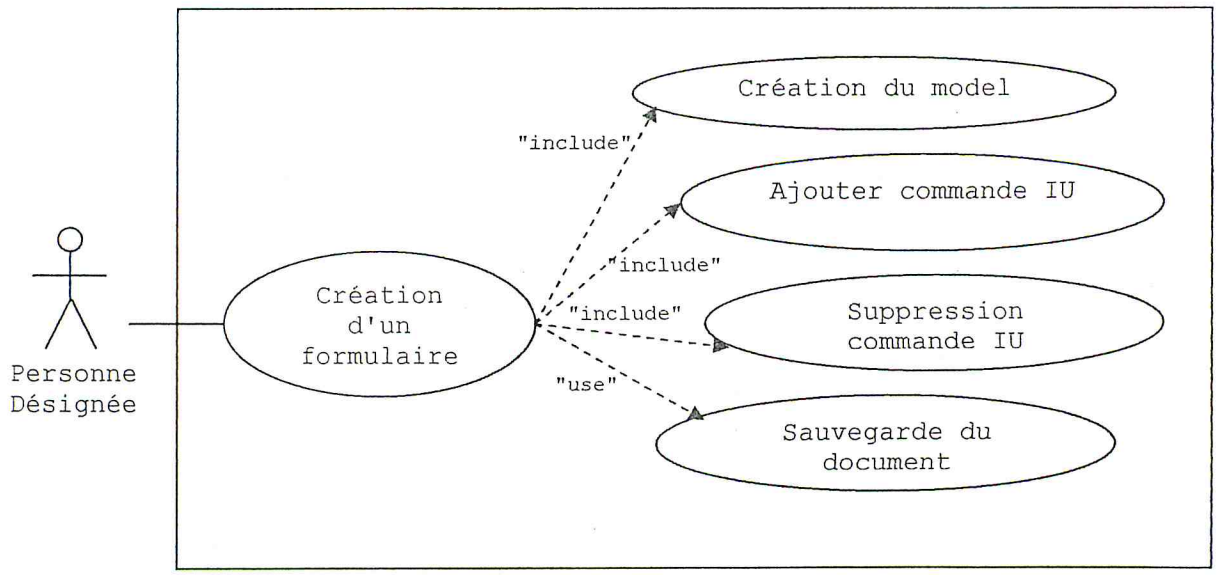


Figure 4.3 : *Diagramme des cas d'utilisation pour le cas création d'un formulaire*

1.1) Création du modèle: Dans le cas suivant l'utilisateur *(la personne désignée)* crée l'arbre de donnée XML, et définit la contrainte de chaque nœud.

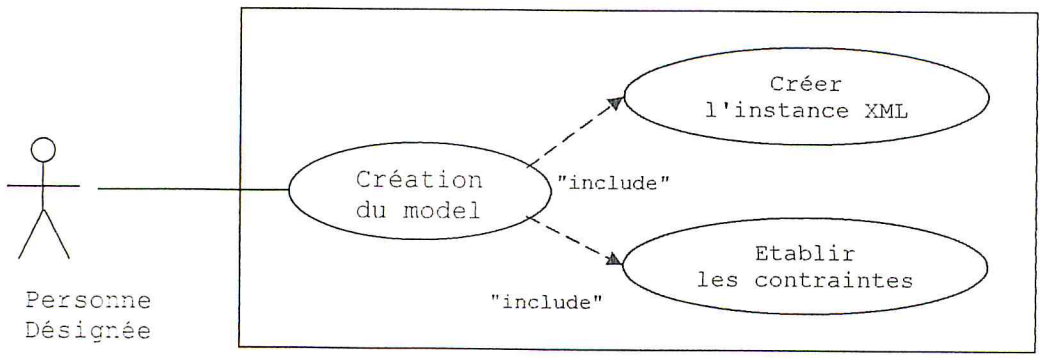


Figure 4.4 : *Diagramme des cas d'utilisation pour le cas création d'un model*

1.2) Ajouter commande UI: Dans ce cas l'utilisateur *(la personne désignée)* créer l'interface utilisateur correspondante au formulaire qu'il veut créer.

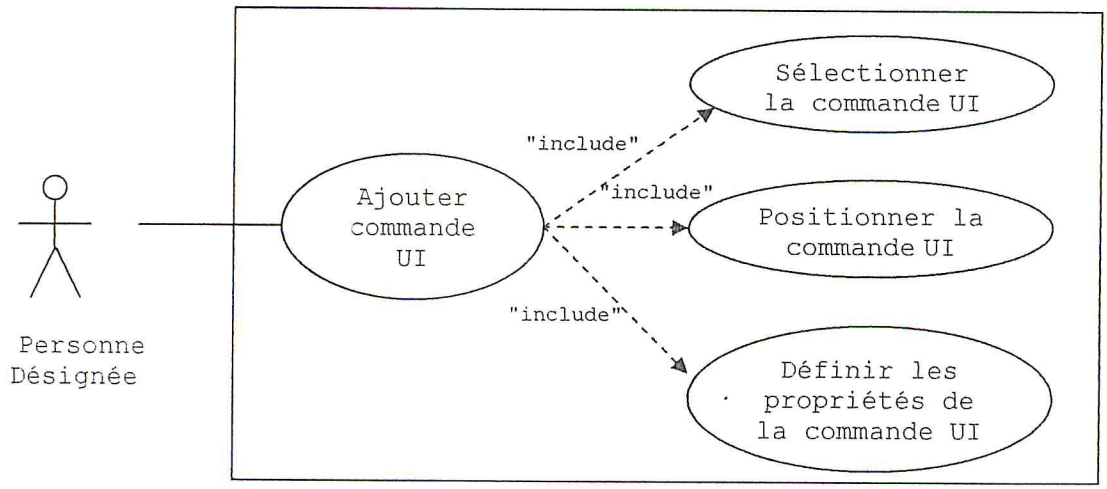


Figure 4.5 : *Diagramme des cas d'utilisation pour le cas de l'ajout d'une commande UI*

1.3) Suppression commande UI: Dans ce cas l'utilisateur *(la personne désignée)* sélectionne la commande UI a supprimer puis la supprime.

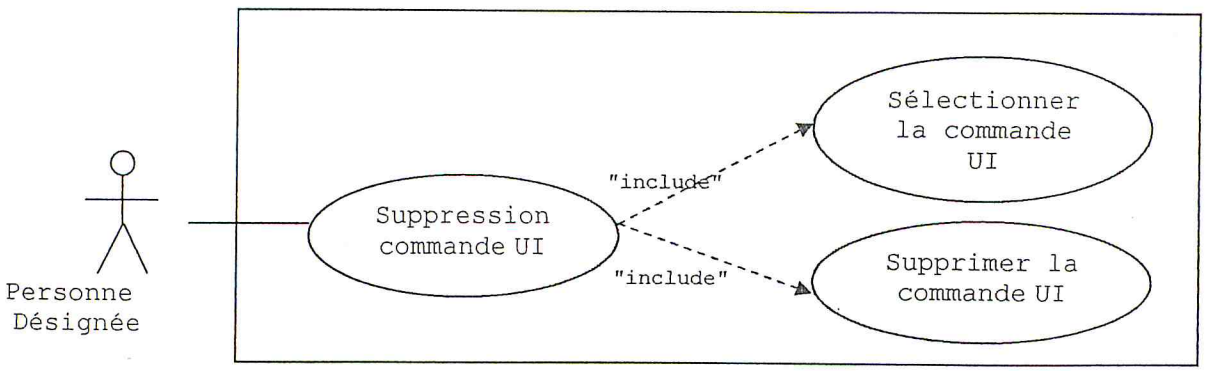


Figure 4.6 : *Diagramme des cas d'utilisation pour le cas de la suppression d'une commande UI*

1.4) Sauvegarde du document: Apres avoir crée le formulaire, la personne qualifiée devra choisir un emplacement puis le sauvegarder.

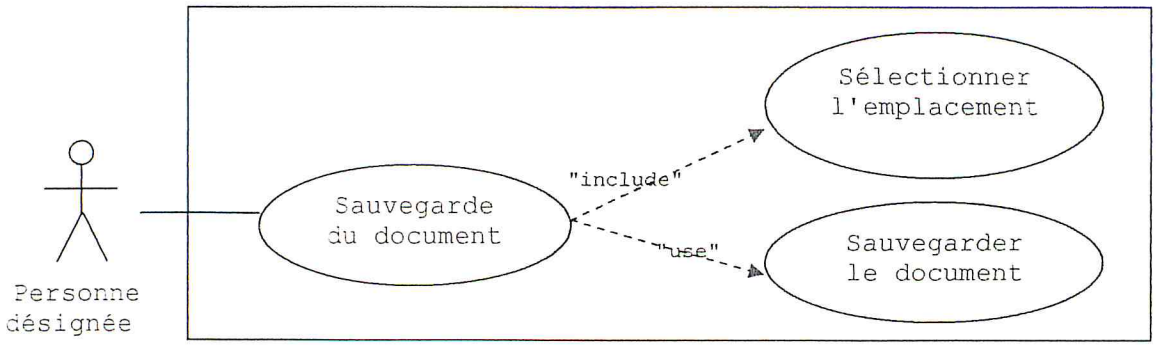


Figure 4.7 : *Diagramme des cas d'utilisation pour le cas de la sauvegarde du document*

2) Modification du formulaire: L'utilisateur peut accéder a un formulaire donné et le modifier en ajoutant des commande UI ou en les supprimant.

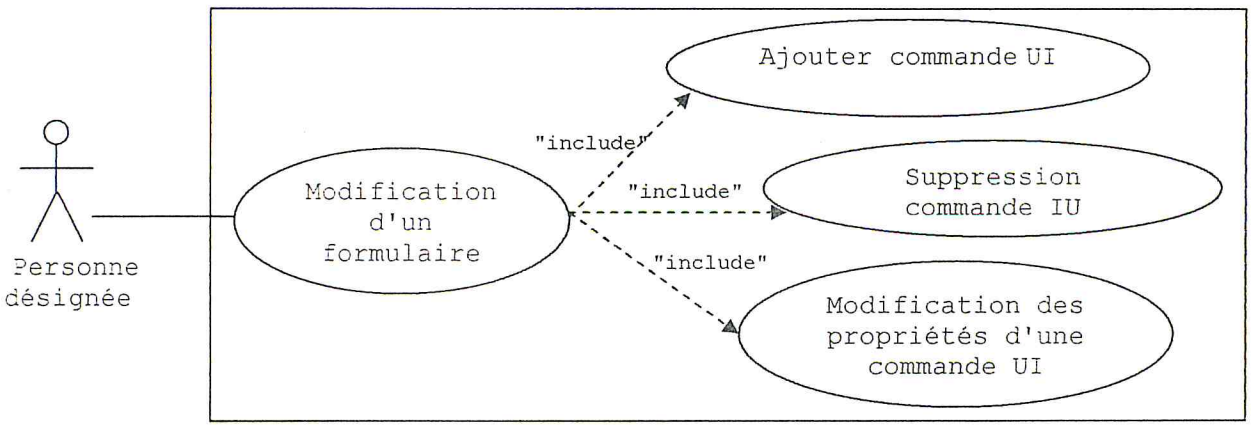


Figure 4.8 : *Diagramme des cas d'utilisation pour le cas de la modification d'un formulaire*

Les cas d'usager pour l'ajouter et la suppression d'une commande sont définis plus haut.

2.1) Modification des propriétés d'une commande UI:

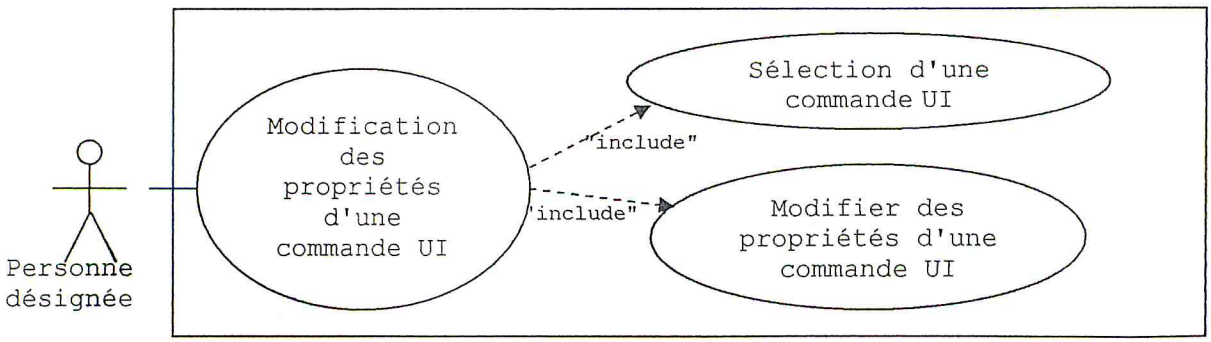


Figure 4.9 : *Diagramme des cas d'utilisation pour le cas de la modification des propriétés d'une commande UI*

3) Visualisation du formulaire:
Lors de la visualisation le formulaire ne pourra pas être édité.
L'aperçu sera géré par un module qu'on appellera Lecteur.

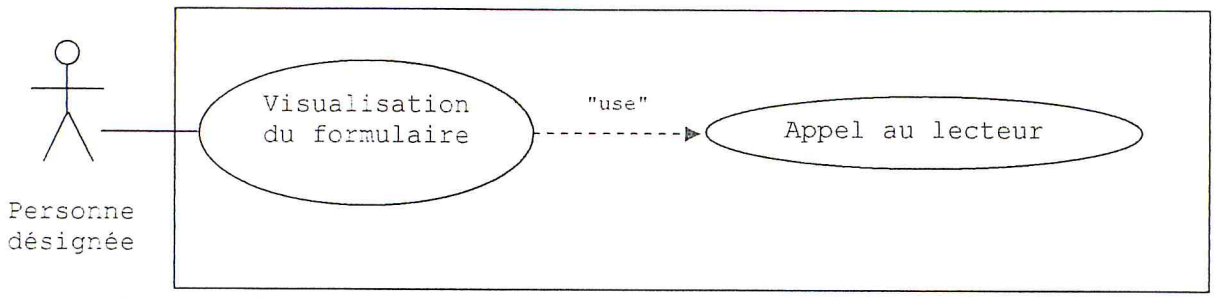


Figure 4.10 : *Diagramme des cas d'utilisation pour le cas de la Visualisation du formulaire*

4) Paramétrer le serveur :
Lors du paramétrage on choisira le numéro de port de connexion

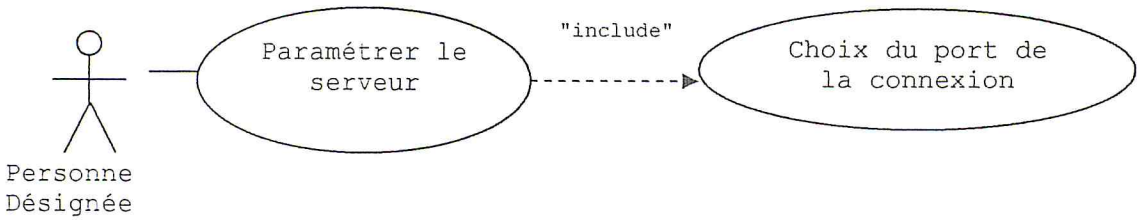


Figure 4.11 : *Diagramme des cas d'utilisation pour le cas du paramétrage du serveur*

V.2.1.2 Partie Client:

1) Se connecter à un serveur:
La connexion s'établira en choisissant le port et le serveur.

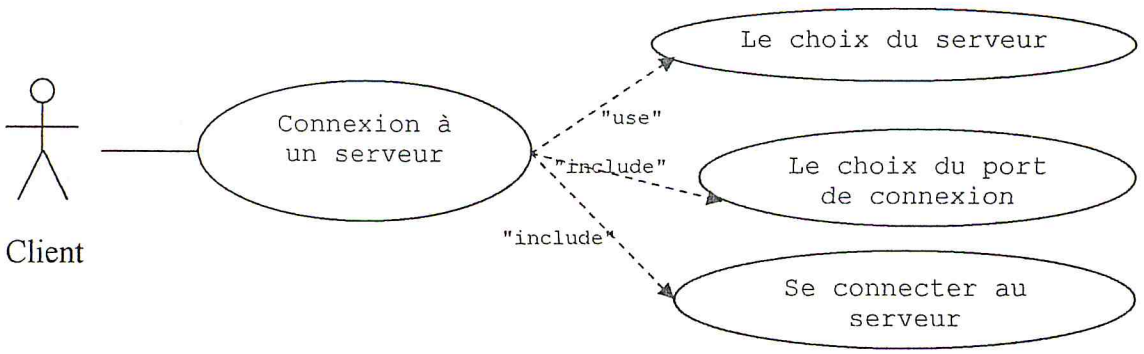


Figure 4.12 : *Diagramme des cas d'utilisation pour le cas de la connexion au serveur*

2) Le choix du formulaire:
Une fois la connexion établit, le client aura à choisir dans une arborescence le modèle de formulaires puis le formulaire qu'il désire *(dans un modèle y a plusieurs formulaires)*, puis valider son choix

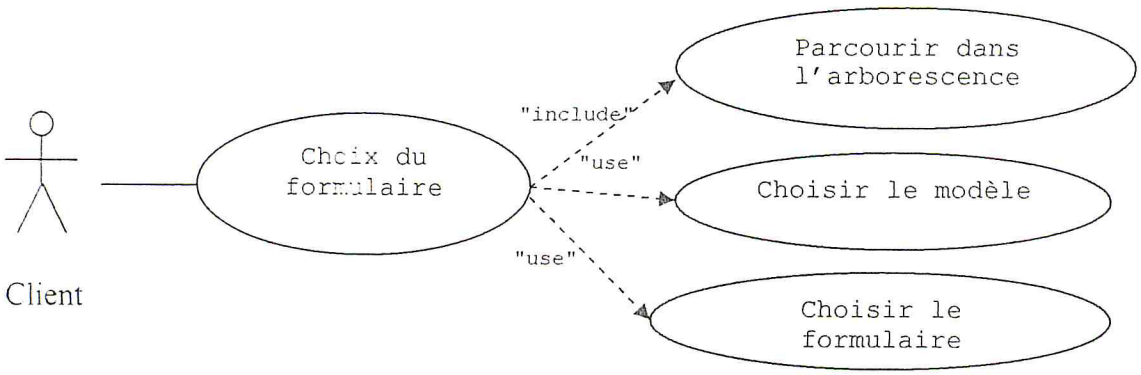


Figure 4.13: *Diagramme des cas d'utilisation pour le cas du choix du formulaire*

3) La visualisation d'un formulaire:
Après le choix du formulaire correspondant, l'utilisateur doit faire un appel au lecteur pour le visualiser.

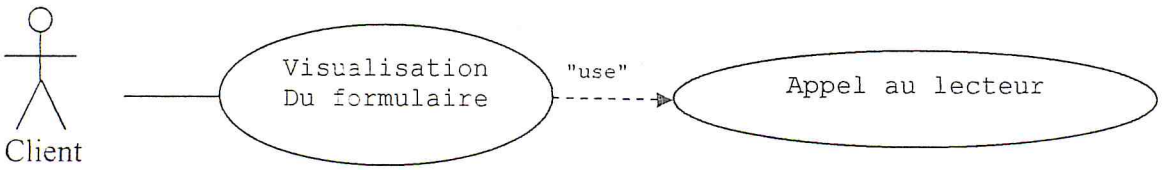


Figure 4.14 : *Diagramme des cas d'utilisation pour la visualisation formulaire*

4) Remplir le formulaire:

A partir de la visualisation du formulaire, le client pourra remplir des champs de saisie *(texte, code, .., etc.)*, répondre a des question en cochant des cases et sélectionner des éléments du formulaire, toutes ces opérations seront regroupées dans le cas d'utilisation "Opération de remplissage ".

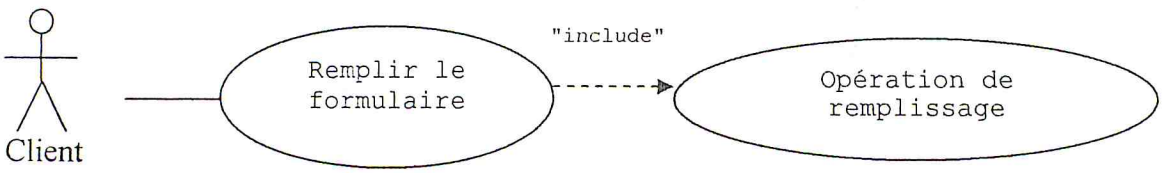


Figure 4.15 : *Diagramme des cas d'utilisation pour le remplissage du formulaire*

5) Signature du formulaire:
Une fois le formulaire rempli et valider une question au choix sera posée au client.
Ce dernier peut choisir de signer le formulaire pour garantir une authentification du client et l'intégrité des données.

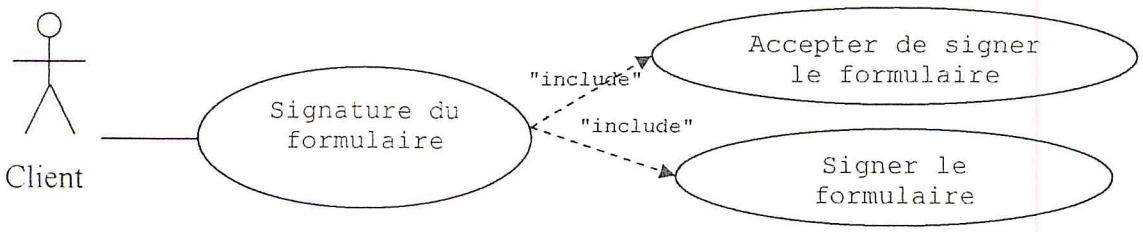


Figure 4.16 : *Diagramme des cas d'utilisation pour la signature du formulaire*

5.1) Signer le formulaire: La signature se fera en sélectionnant l'algorithme de signature et introduire clé privée.

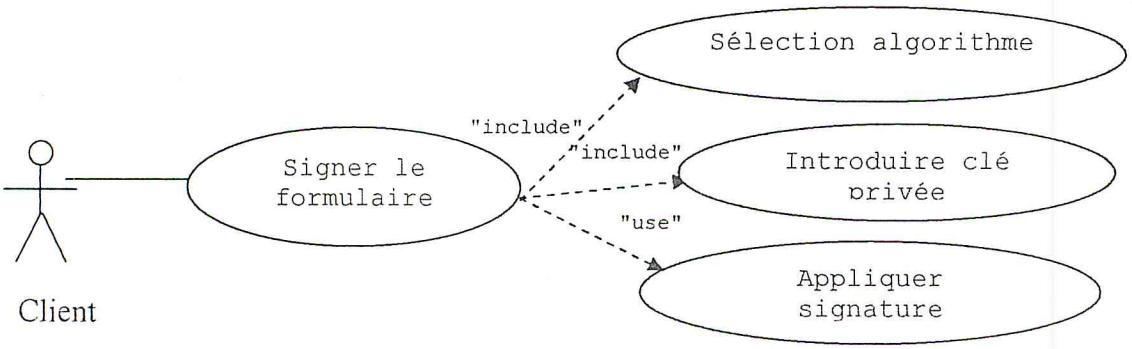


Figure 4.17 : *Diagramme des cas d'utilisation pour signer le formulaire*

V.2.3 Le Diagramme des cas d'utilisation:

a) Partie serveur:

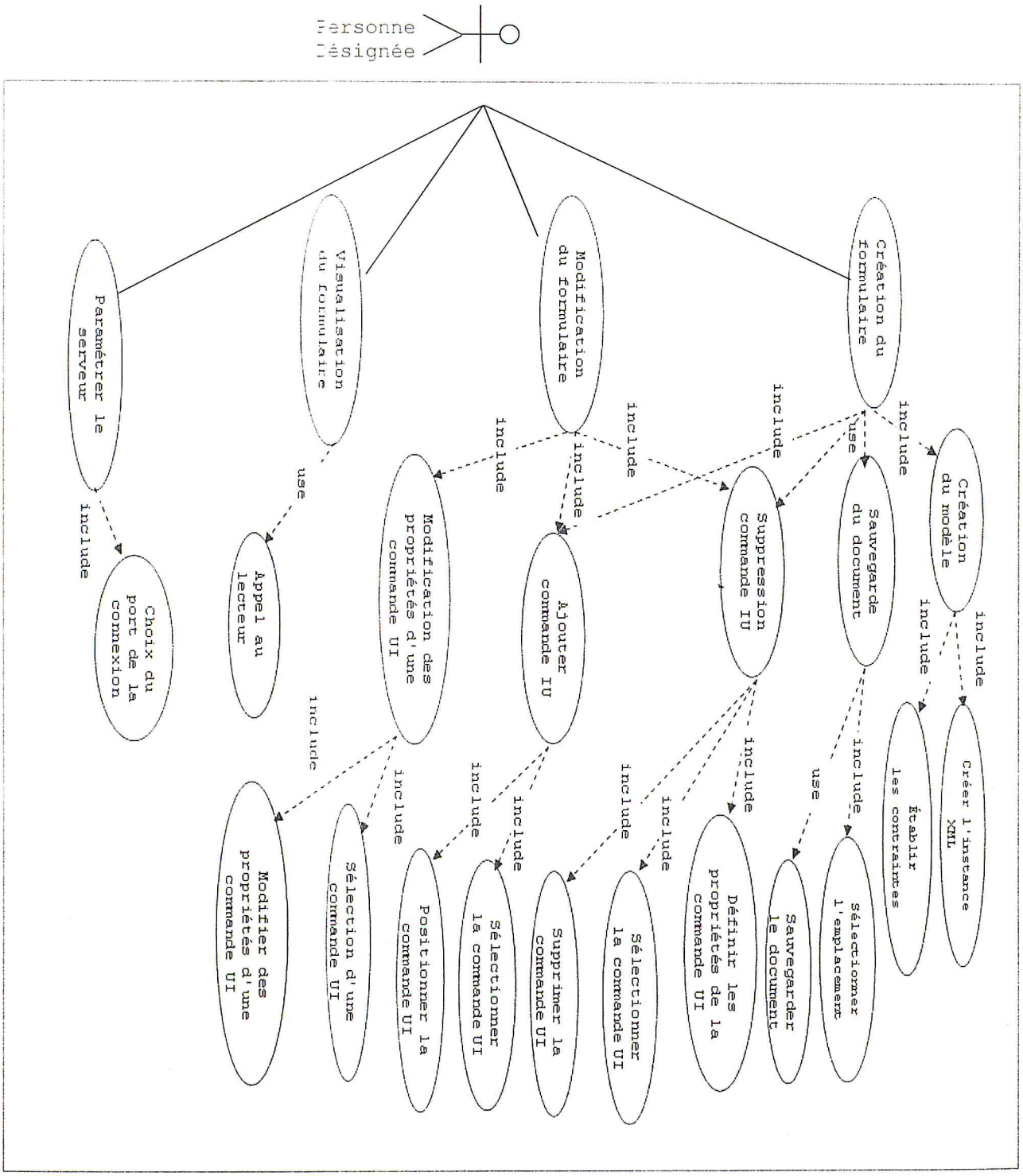


Figure 4.18 : *Diagramme de cas d'utilisation côté serveur*

b) Partie client:

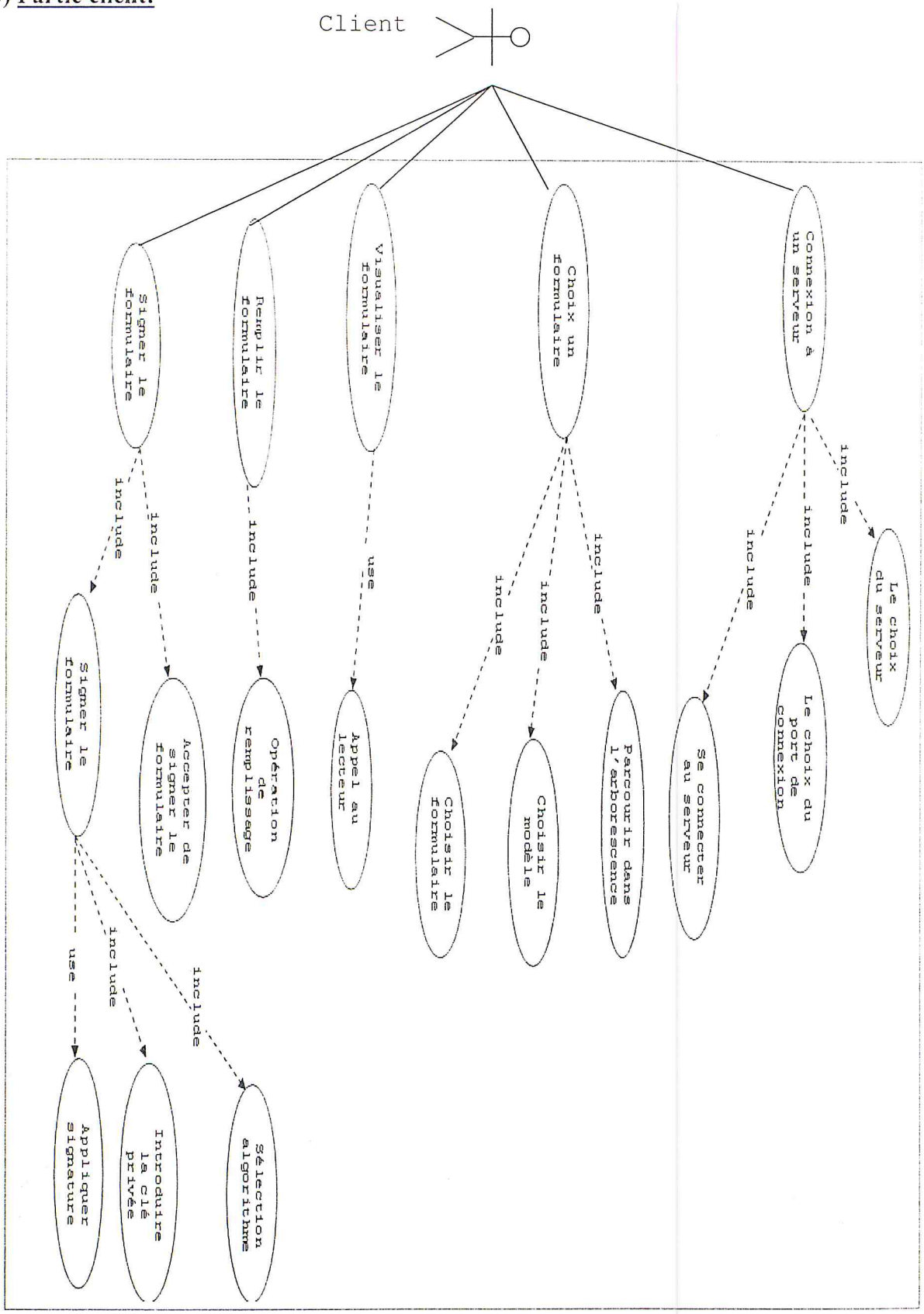


Figure 4.19 : *Diagramme de cas d'utilisation côté client*

V.3 Diagramme de séquence:

Les diagrammes de séquence montre les interactions entre objet selon un point de vue temporel. La représentation ce concentre sur l'expression des interactions.

- Représentation des interactions:
Scénario "Déroulement sans erreurs":
On dispose des objets suivants: -Personne qualifiée -Module Serveur -Module Client - Client

1- La personne qualifiée demande la création d'un formulaire.

2- Le serveur ouvre un document vierge.

3- La personne qualifiée crée le modèle XML associé au formulaire.

4- La personne qualifiée ajoute les composants UI, spécifier leurs emplacements et définir leurs propriétés.

5- La personne qualifiée demande la sauvegarde du document.

6- Le serveur demande de choisir le nom du document et un emplacement mémoire *(Qui va déterminer la catégorie du formulaire)* pour le stocker.

7- La personne qualifiée donne le nom et l'emplacement puis valide.

8- Le client demande au module client une connexion au serveur.

9- Le module client demande au client de spécifier le nom du serveur et le port de connexion.

10- Le client détermine le nom du serveur et le numéro du port puis valide.

11- Le module client se connecte au serveur.

12- Le serveur accepte la connexion.

13- Le serveur envoi au module client l'arborescence de ses formulaires.

14- Le module client visualise l'arborescence de formulaires.

15- Le client choisi dans l'arborescence la catégorie puis le formulaire qui il désire.

16- Le module client visualise le formulaire choisi.

17- Le client rempli les informations nécessaires dans le formulaire puis demande à l'application client de le soumettre.

18- Le module client demande au client si il veut signer le formulaire.

V.3 Diagramme de séquence:

Les diagrammes de séquence montre les interactions entre objet selon un point de vue temporel. La représentation ce concentre sur l'expression des interactions.

- Représentation des interactions:

Scénario "Déroulement sans erreurs":

On dispose des objets suivants: -Personne qualifiée -Module Serveur -Module Client - Client

1- La personne qualifiée demande la création d'un formulaire.

2- Le serveur ouvre un document vierge.

3- La personne qualifiée crée le modèle XML associé au formulaire.

4- La personne qualifiée ajoute les composants UI, spécifier leurs emplacements et définir leurs propriétés.

5- La personne qualifiée demande la sauvegarde du document.

6- Le serveur demande de choisir le nom du document et un emplacement mémoire *(Qui va déterminer la catégorie du formulaire)* pour le stocker.

7- La personne qualifiée donne le nom et l'emplacement puis valide.

8- Le client demande au module client une connexion au serveur.

9- Le module client demande au client de spécifier le nom du serveur et le port de connexion.

10- Le client détermine le nom du serveur et le numéro du port puis valide.

11- Le module client se connecte au serveur.

12- Le serveur accepte la connexion.

13- Le serveur envoi au module client l'arborescence de ses formulaires.

14- Le module client visualise l'arborescence de formulaires.

15- Le client choisi dans l'arborescence la catégorie puis le formulaire qui il désire.

16- Le module client visualise le formulaire choisi.

17- Le client rempli les informations nécessaires dans le formulaire puis demande à l'application client de le soumettre.

18- Le module client demande au client si il veut signer le formulaire.

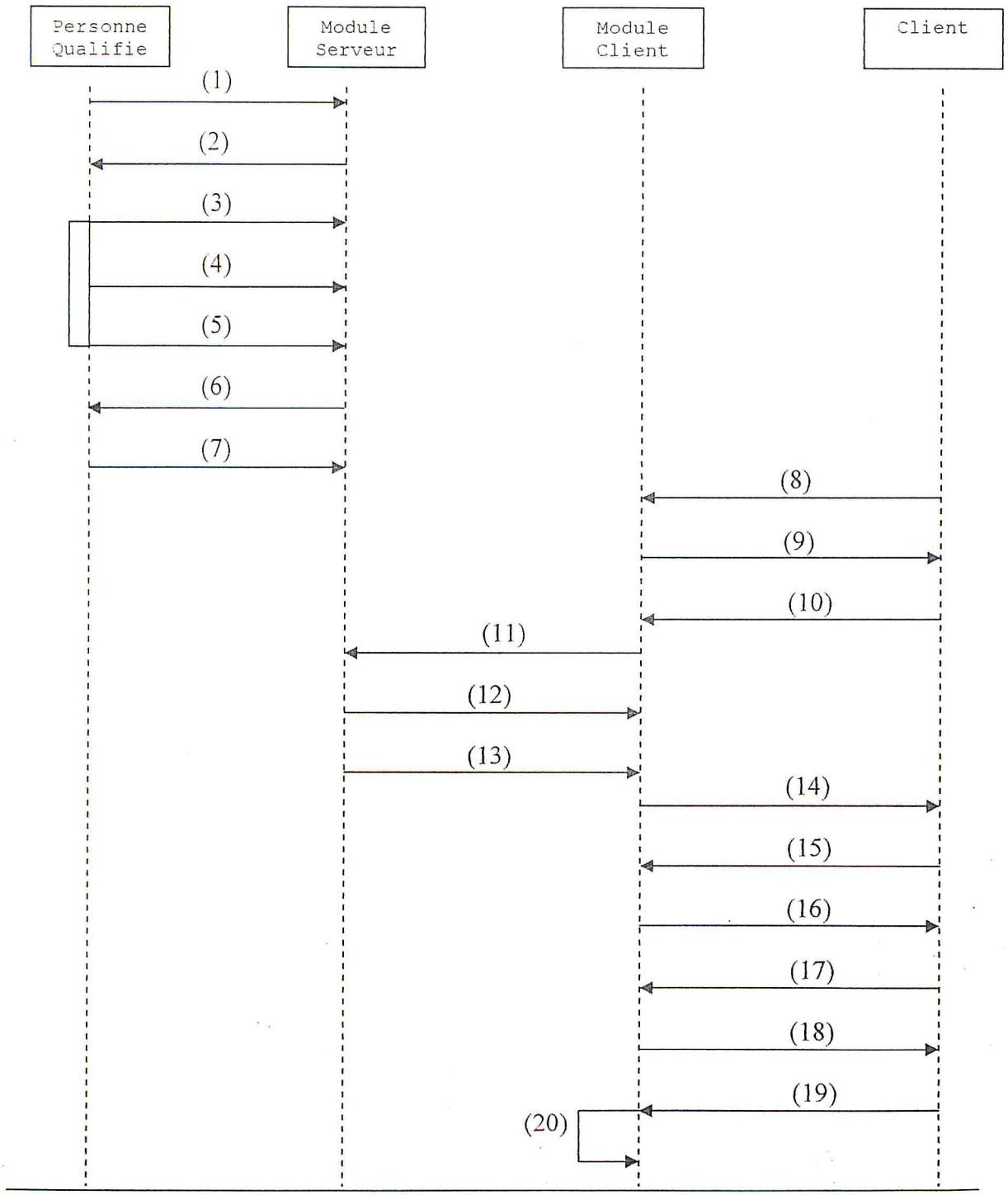
19- Le client accepte de signer le formulaire.
20- L'application client signe le formulaire.

21- L'application client envoi les informations signées au serveur.

22- L'application serveur vérifie la validité de la signature.

23- Après la validité de la signature le serveur stock les données dans une base de données.

24- Signaler le bon déroulement de la transaction au module client



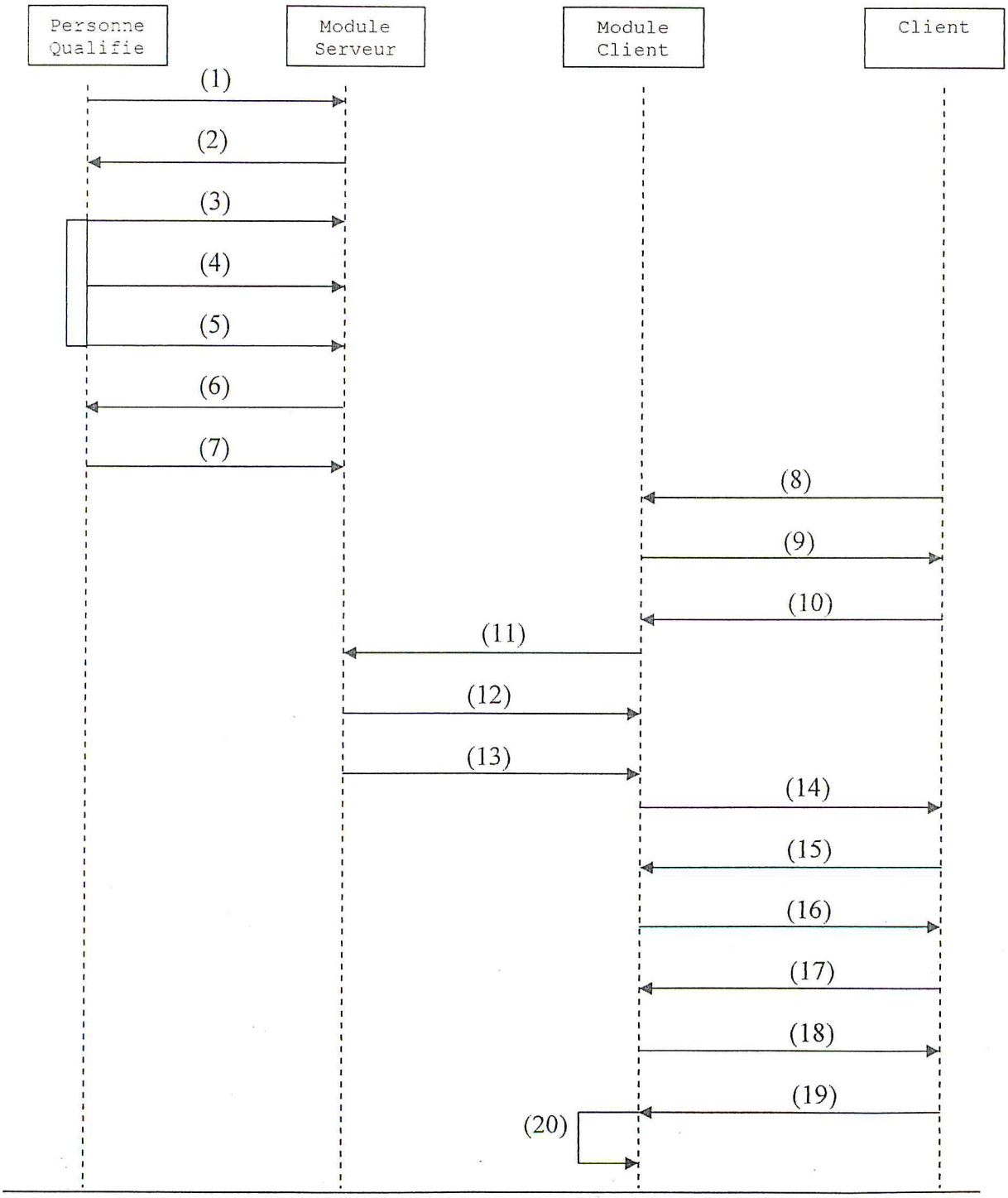
19- Le client accepte de signer le formulaire.
20- L'application client signe le formulaire.

21- L'application client envoi les informations signées au serveur.

22- L'application serveur vérifie la validité de la signature.

23- Après la validité de la signature le serveur stock les données dans une base de données.

24- Signaler le bon déroulement de la transaction au module client



(Suite)

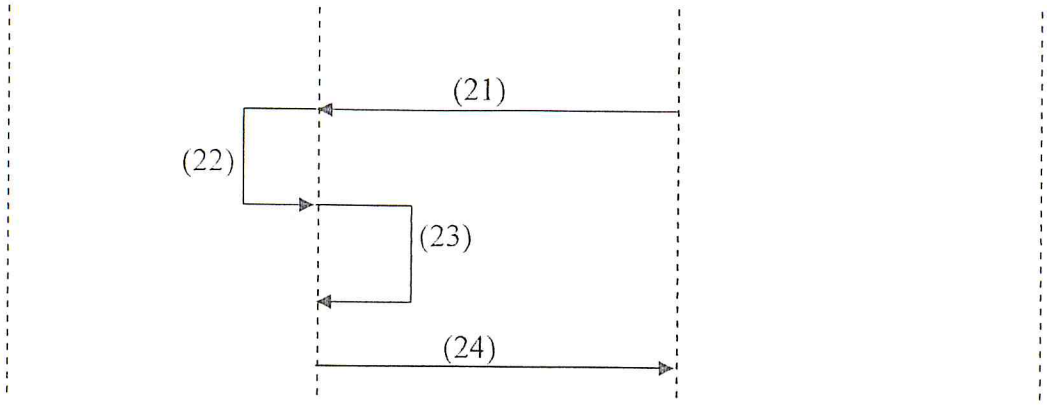


Figure 4.20 : *Diagramme de séquence pour l'interaction sans erreurs*

Scénario "Erreur de connexion au serveur":

1- La personne qualifiée demande la création d'un formulaire.

2- Le serveur ouvre un document vierge.

3- La personne qualifiée crée le modèle XML associé au formulaire.

4- La personne qualifiée ajoute les composants UI, spécifier leurs emplacements et définir leurs propriétés.

5- La personne qualifiée demande la sauvegarde du document.

6- Le serveur demande de choisir le nom du document et un emplacement mémoire *(Qui va déterminer la catégorie du formulaire)* pour le stocker.

7- La personne qualifiée donne le nom et l'emplacement puis valide.

8- Le client demande au module client une connexion au serveur.

9- Le module client demande au client de spécifier le nom du serveur et le port de connexion.

10- Le client détermine le nom du serveur et le numéro du port puis valide.

11- Le module client se connecte au serveur.

12- Il y a une erreur de connexion *(erreur sur le port ou serveur introuvable)*, Le module client signale au client le type d'erreur produite.

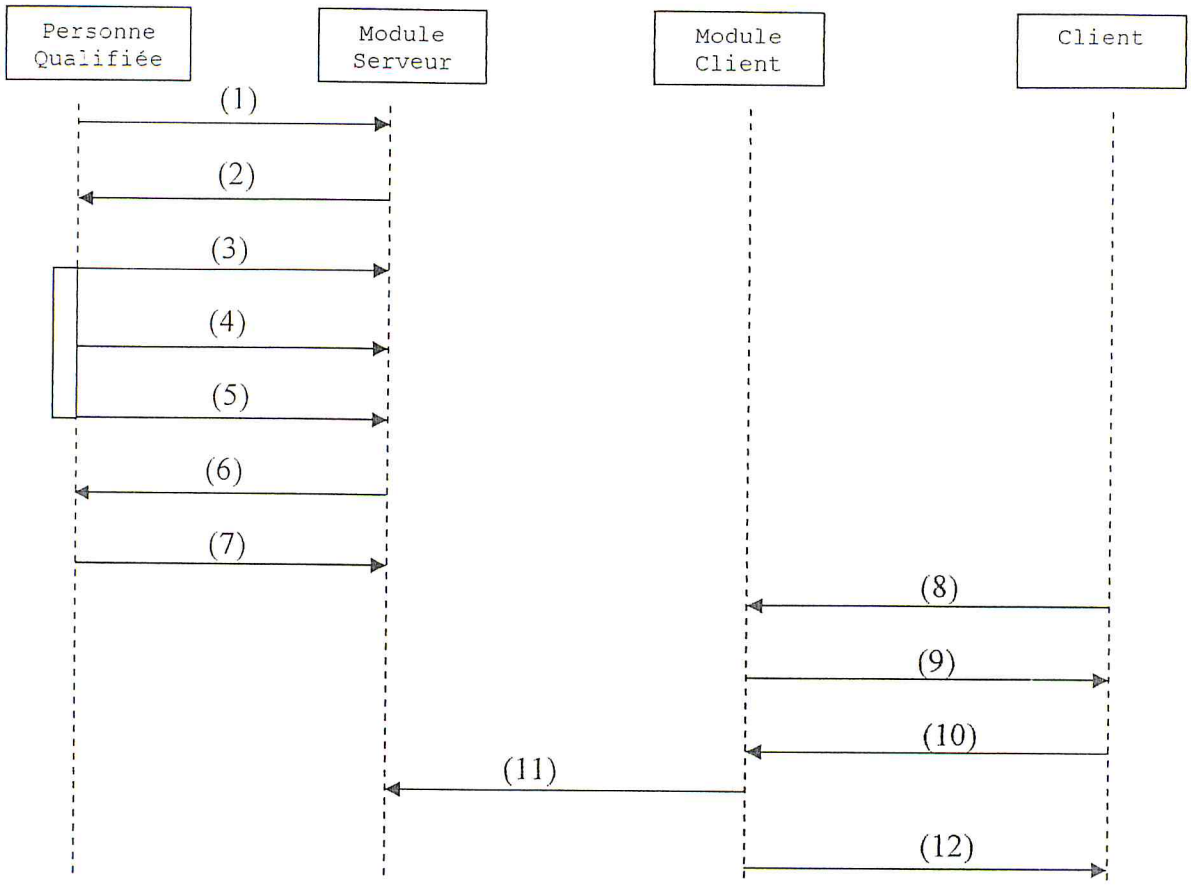


Figure 4.21 : *Diagramme de séquence pour l'erreur de connexion*

Scénario "Le client refuse de signer le formulaire":

1- La personne qualifiée demande la création d'un formulaire.

2- Le serveur ouvre un document vierge.

3- La personne qualifiée crée le modèle XML associé au formulaire.

4- La personne qualifiée ajoute les composants UI, spécifier leurs emplacements et définir leurs propriétés.

5- La personne qualifiée demande la sauvegarde du document.

6- Le module serveur demande de choisir le nom du document et un emplacement mémoire
(Qui va déterminer la catégorie du formulaire) pour le stocker.

7- La personne qualifiée donne le nom et l'emplacement puis valide.

8- Le client demande au module client une connexion au serveur.

9- Le module client demande au client de spécifier le nom du serveur et le port de connexion.

10- Le client détermine le nom du serveur et le numéro du port puis valide.

11- Le module client se connecte au module serveur.

12- Le serveur accepte la connexion.

13- Le serveur envoi au module client l'arborescence de ses formulaires.

14- Le module client visualise l'arborescence de formulaires.

15- Le client choisi dans l'arborescence la catégorie puis le formulaire qui il désire.

16- Le module client visualise le formulaire choisi.

17- Le client rempli les informations nécessaires dans le formulaire puis demande à l'application client de le soumettre.

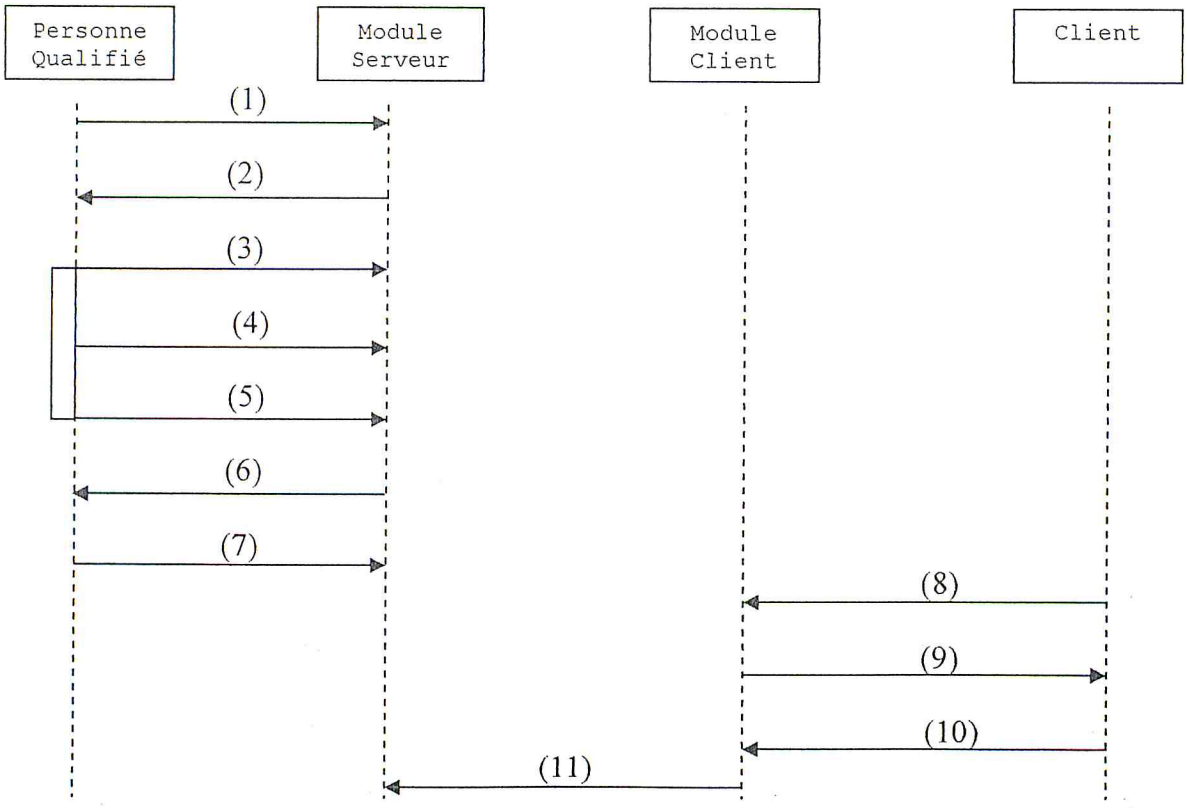
18- Le module client demande au client si il veut signer le formulaire.

19- Le client refuse de signer le formulaire.

20- L'application client envoi les données au serveur.

21- Le serveur stocke les données dans une base de données.

22- Signaler le bon déroulement de la transaction au module client



(Suite)

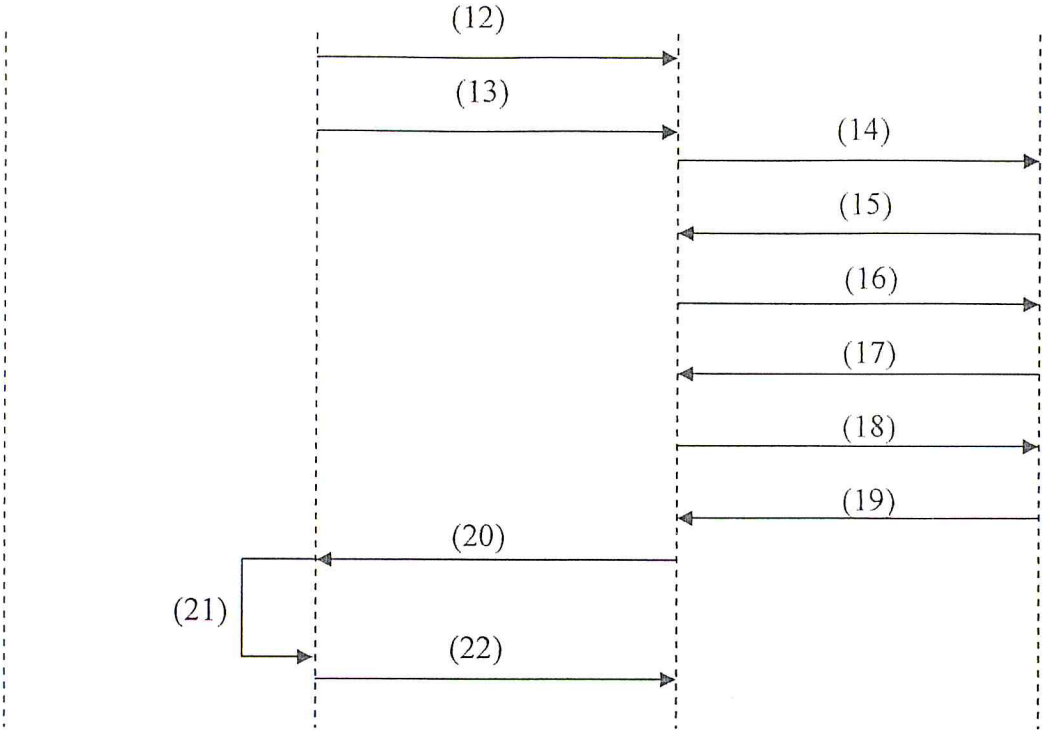


Figure 4.22 : *Diagramme de séquence pour le refus du client de signer le formulaire*

Scénario "Signature invalide":

1- La personne qualifiée demande la création d'un formulaire.

2- Le serveur ouvre un document vierge.

3- La personne qualifiée crée le modèle XML associé au formulaire.

4- La personne qualifiée ajoute les composants UI, spécifier leurs emplacements et définir leurs propriétés.

5- La personne qualifiée demande la sauvegarde du document.

6- Le serveur demande de choisir le nom du document et un emplacement mémoire *(Qui va déterminer la catégorie du formulaire)* pour le stocker.

7- La personne qualifiée donne le nom et l'emplacement puis valide.

8- Le client demande au module client une connexion au serveur.

9- Le module client demande au client de spécifier le nom du serveur et le port de connexion.

10- Le client détermine le nom du serveur et le numéro du port puis valide.

11- Le module client se connecte au serveur.

12- Le serveur accepte la connexion.

13- Le serveur envoi au module client l'arborescence de ses formulaires.

14- Le module client visualise l'arborescence de formulaires.

15- Le client choisi dans l'arborescence la catégorie puis le formulaire qui il désire.

16- Le module client visualise le formulaire choisi.

17- Le client rempli les informations nécessaires dans le formulaire puis demande à l'application client de le soumettre.

18- Le module client demande au client si il veut signer le formulaire.

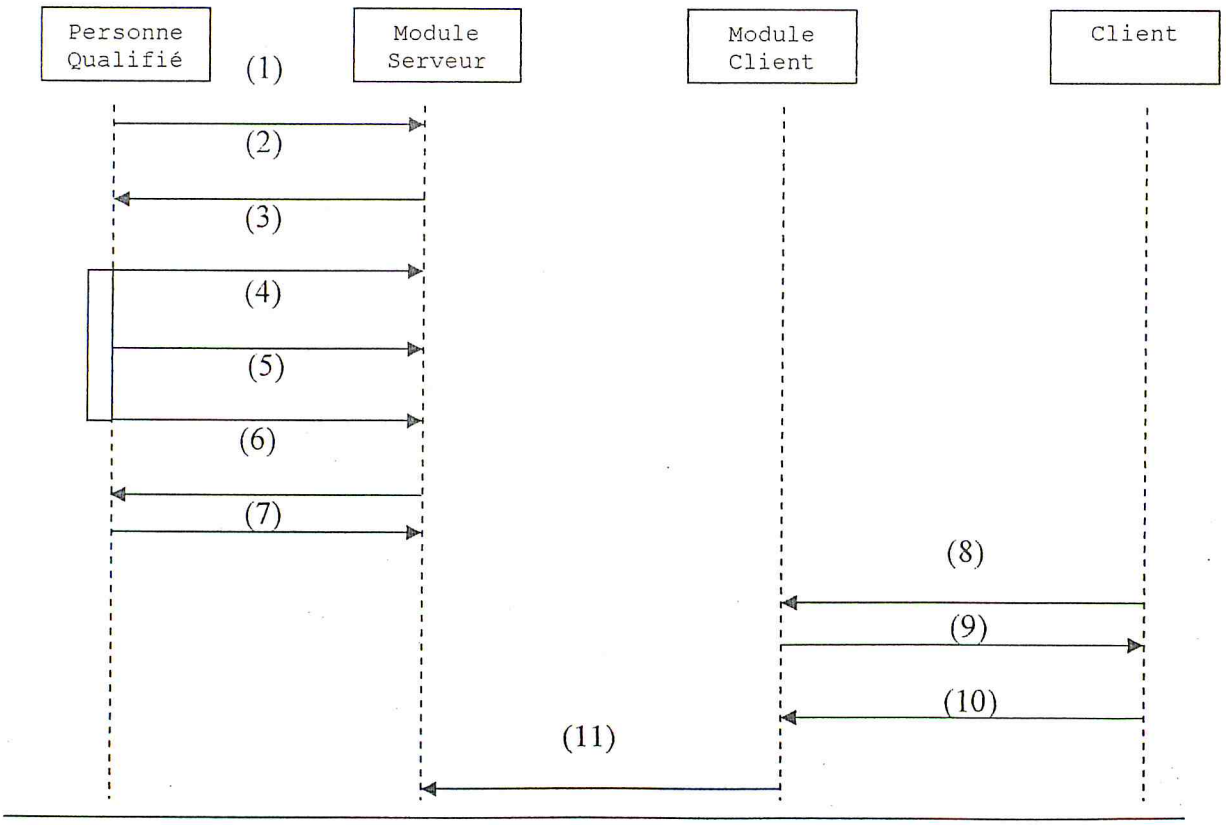
19- Le client accepte de signer le formulaire.

20- L'application client signe le formulaire.

21- L'application client envoi les informations signées au serveur.

22- L'application serveur vérifie la validité de la signature.

23- Le serveur conclu que la signature est invalide, rejette les données et signale l'erreur au module client.



(Suite)

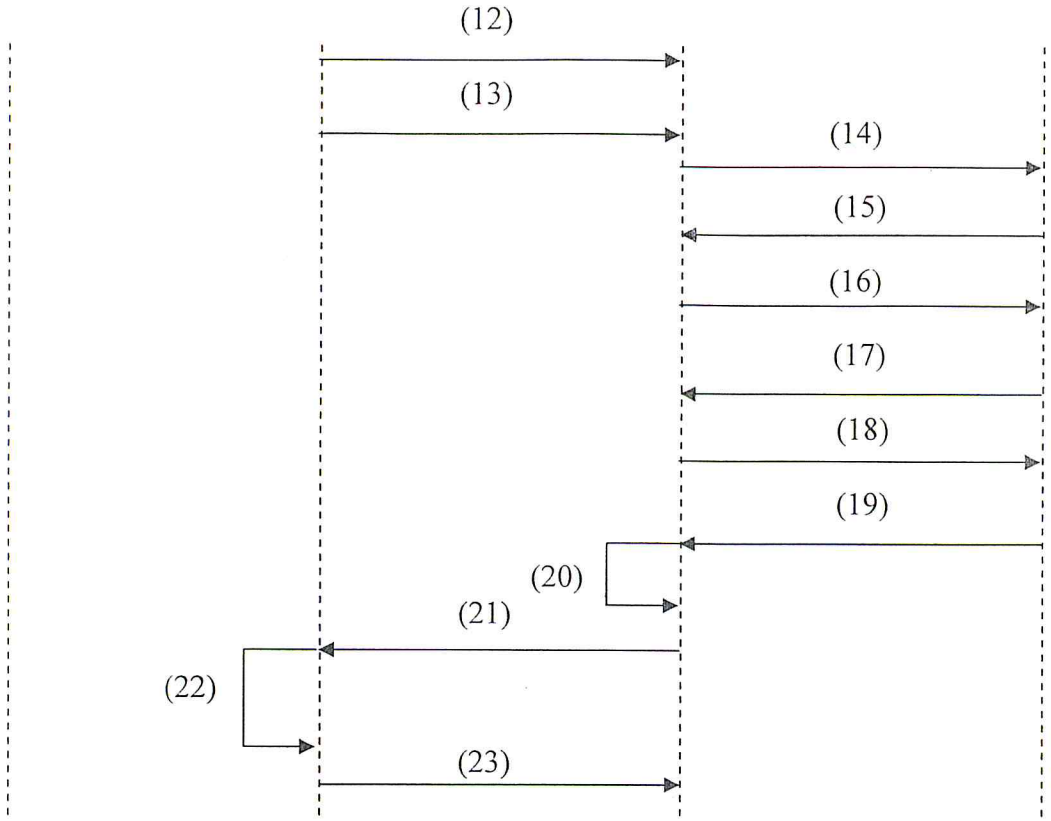


Figure 4.23 : *Diagramme de séquence pour signature invalide*

i) Création du modèle XML:
Scénario:
1- La personne qualifiée demande la création du modèle XML.
2- La personne qualifiée demande insérer un élément.
3- Le Serveur demande le nom du noeud
4- La personne qualifiée donne le nom du noeud.
5- Le serveur met a jour l'arbre XML.

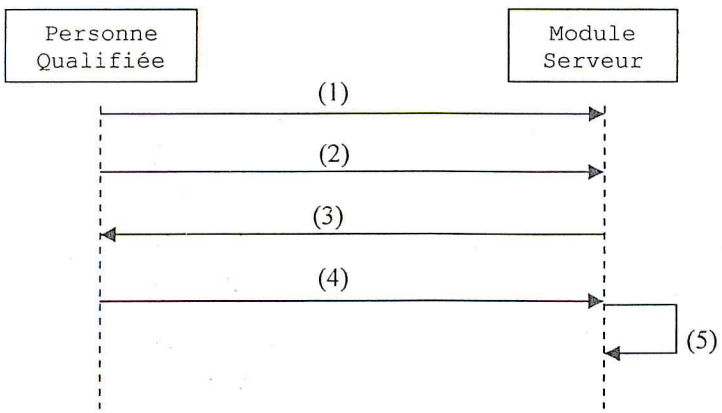


Figure 4.24 : *Diagramme de séquence pour la création du modèle XML*

i.1) Modification d'un nœud de l'arbre XML:

i.1.1) -Scénario "Modification du nom du nœud sélectionné":

1- La personne qualifiée demande la modification du nœud sélectionné.
2- Le serveur demande de donner choisir le nouveau nom du noeud.
3- La personne qualifiée donne le nouveau nom et valide.
4- Le serveur met a jour l'arbre XML.

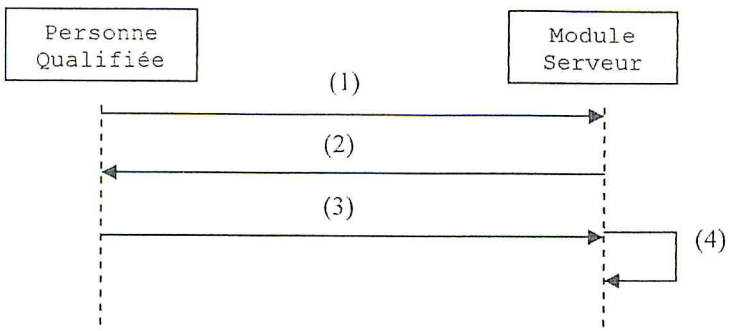


Figure 4.25 : *Diagramme de séquence pour la modification du nœud de l'arbre XML*

i.1.2) -Scénario "modification de la valeur nœud sélectionné":

Cas sans erreur:
Sénarion:

1- La personne qualifiée demande la modification de la valeur du nœud sélectionné.
2- Le serveur demande de donner la valeur nom du noeud.
3- La personne qualifiée donne la valeur du noeud et valide.
4- Le serveur vérifie la compatibilité entre la valeur donnée et le type du noeud.
5- Le serveur met a jour l'arbre XML.

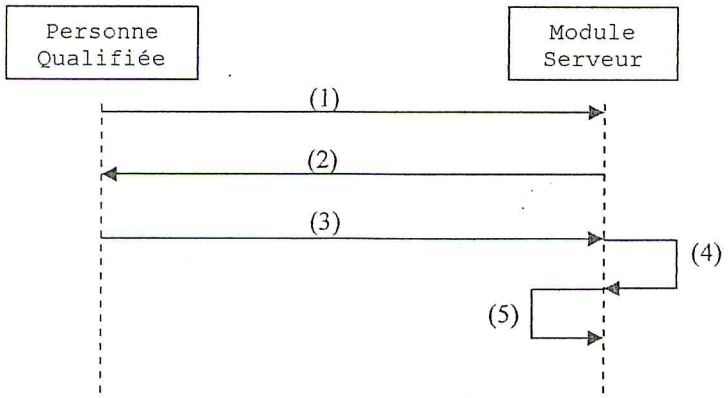


Figure 4.26 : *Diagramme de séquence pour la modification sans erreur de la valeur du nœud sélectionné*

Cas avec erreur :
Sénarion:

1- La personne qualifiée demande la modification de la valeur du nœud sélectionné.
2- Le serveur demande de donner la valeur nom du noeud.
3- La personne qualifiée donne la valeur du noeud et valide.
4- Le serveur vérifie la compatibilité entre la valeur donnée et le type du noeud.
5- Le serveur signale l'erreur à la personne qualifiée.

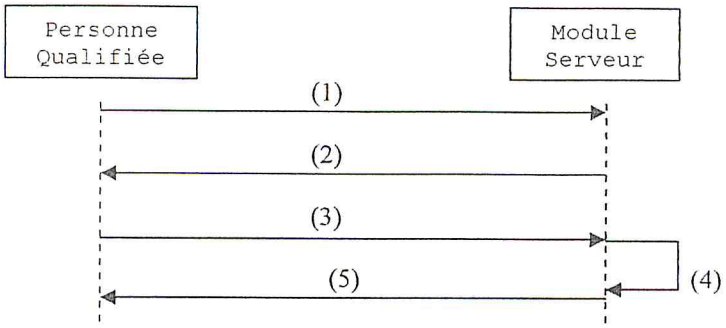


Figure 4.27 : *Diagramme de séquence pour la modification avec erreur de la valeur du nœud sélectionné*

i.1.3) -Scénario "modification de l'attribut du nœud sélectionné":

1- La personne qualifiée demande la modification de l'attribut du nœud sélectionné.
2- Le serveur demande de choisir l'attribut dans une liste.
3- La personne qualifiée choisit l'attribut et valide.
4- Le serveur demande de donner les modifications.
5- La personne qualifiée donne les modifications *(nom et valeur)* et valide.
6- Le serveur met a jour l'arbre XML.

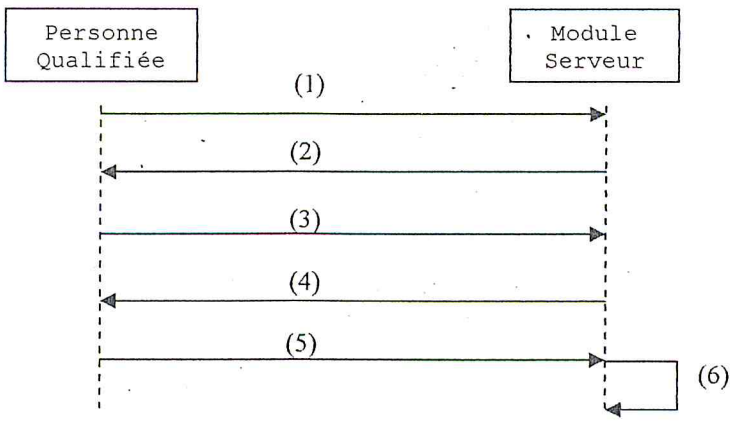


Figure 4.28 : *Diagramme de séquence pour la modification d'attribut du nœud sélectionné*

i.1.4) -Scénario "Suppression du nœud sélectionné":

1- La personne qualifiée demande la suppression du nœud sélectionné.

2- Le serveur supprime le nœud et met a jour l'arbre XML.

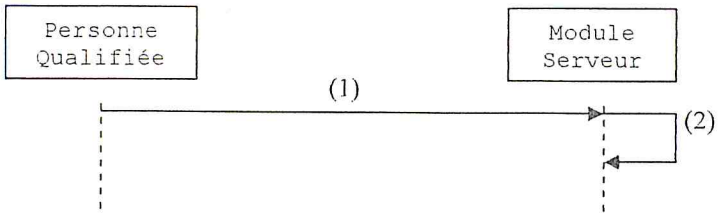


Figure 4.29 : *Diagramme de séquence pour la suppression du nœud de l'arbre XML*

i.1.5) -Scénario "Suppression d'un attribut du nœud sélectionné":

1- La personne qualifiée demande la suppression d'un attribut.

2- Le serveur demande de choisir l'attribut dans une liste.

3- La personne qualifiée choisit l'attribut.

4- La personne qualifiée supprime l'attribut.

5- Le serveur met a jour l'arbre XML.

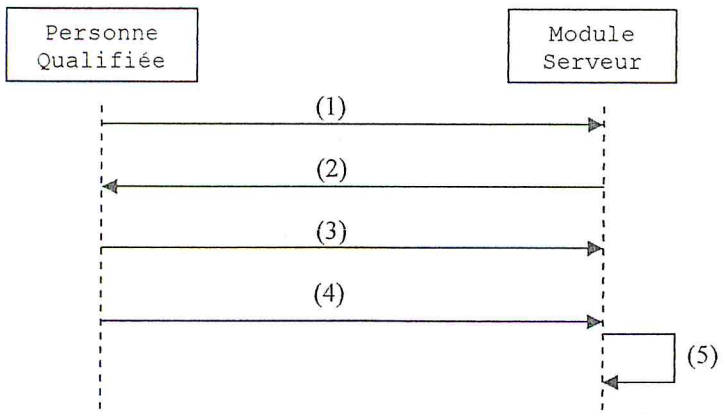


Figure 4.30 : *Diagramme de séquence pour la suppression d'un attribut du nœud de l'arbre XML*

i.1.6) -Scénario "Ajout d'un élément fils au nœud sélectionné":

1- La personne qualifiée demande l'ajout d'un élément fils.

2- Le serveur demande de donnée le nom et le type du noeud.

3- La personne qualifiée donne le nom et le type du noeud.

4- Le serveur met a jour l'arbre XML.

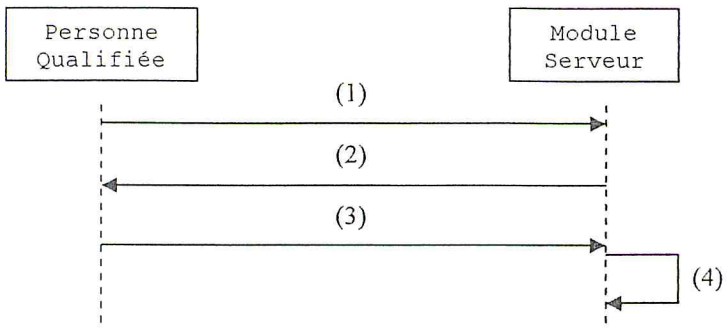


Figure 4.31 : *Diagramme de séquence pour l'ajout d'in nœud fils au nœud de l'arbre XML*

i.1.7) -Scénario "Ajout d'un attribut au nœud sélectionné":

1- La personne qualifiée demande l'ajout d'un attribut.

2- Le serveur demande de donnée le nom et le la valeur de l'attribut.

3- La personne qualifiée donne le nom et la valeur de l'attribut.

4- Le serveur met a jour l'arbre XML.

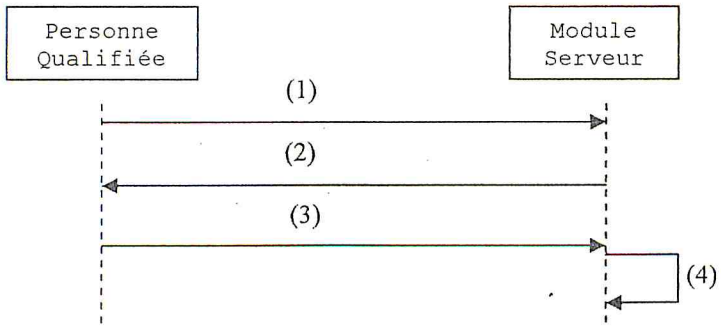


Figure 4.32 : Diagramme *de séquence pour l'ajout d'in attribut au nœud de l'arbre XML*

ii) Sauvegarde du document:

Scénario:

1- Demande d'enregistrement.

2- Enregistrement.

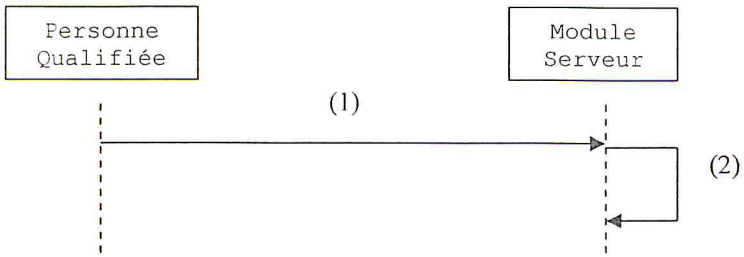
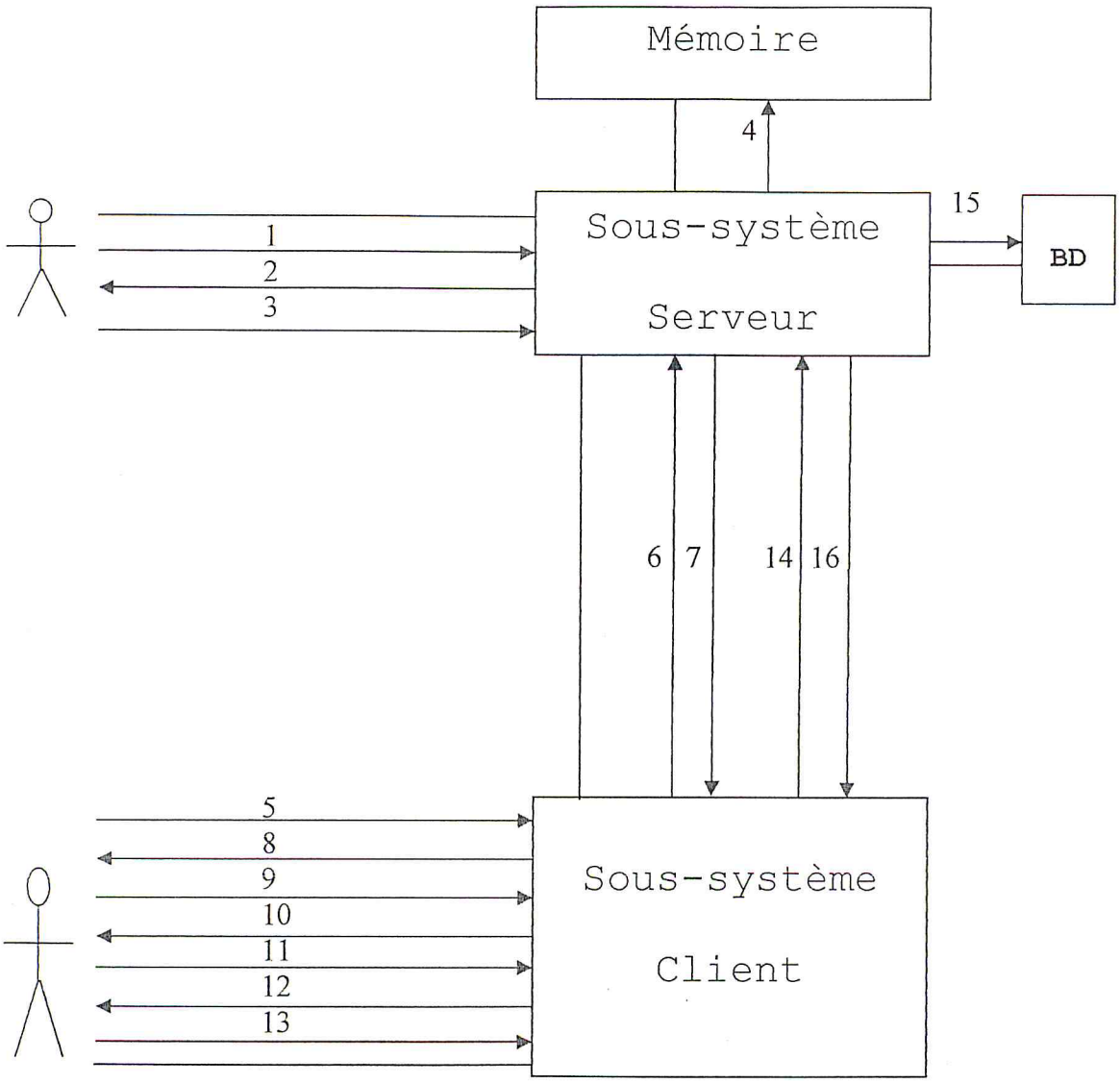


Figure 4.33 : *Diagramme de séquence l'enregistrement du document*

V.4 Diagramme de collaboration : Exprimant la même sémantique que les diagrammes de séquence, les diagrammes de collaboration montrent l'interaction de quelques objets dans une situation donnée. Cette interaction se fait par des messages échangés entre ces objets.
Vu que notre système se compose de deux sous-système on donnera un seul diagramme de collaboration incluant les deux modules :



1. Demande de création.
2. Afficher un document vierge.
3. Création d'un nouveau formulaire.
4. Stockage.
5. Demande de connexion.
6. Connexion.
7. Accepter la connexion et envoyer l'arborescence.
8. Visualiser l'arborescence.

9 : Choisir un formulaire.
10 : Visualiser le formulaire choisi.
11 : Remplir le formulaire.
12 : Demande de signer le formulaire.
13 : Accepter de signer.
14 : Envoi du formulaire au serveur.
15 : Sauvegarde des données.
16 : Confirmation de la sauvegarde des données.

Figure 4.34 : *Diagramme de collaboration du système d'échange de formulaire.*

V.5 Diagramme d'activité : Un diagramme d'activité représente l'état de l'exécution d'un cas d'utilisation sous la forme d'un déroulement d'étapes regroupées séquentiellement dans des branches parallèles de flot de contrôle. Chaque activité est placée dans le couloir correspondant à l'acteur qui assume cette activité. Nous donnerons ci-dessous un diagramme d'activité pour le cas d'utilisation "création du formulaire".

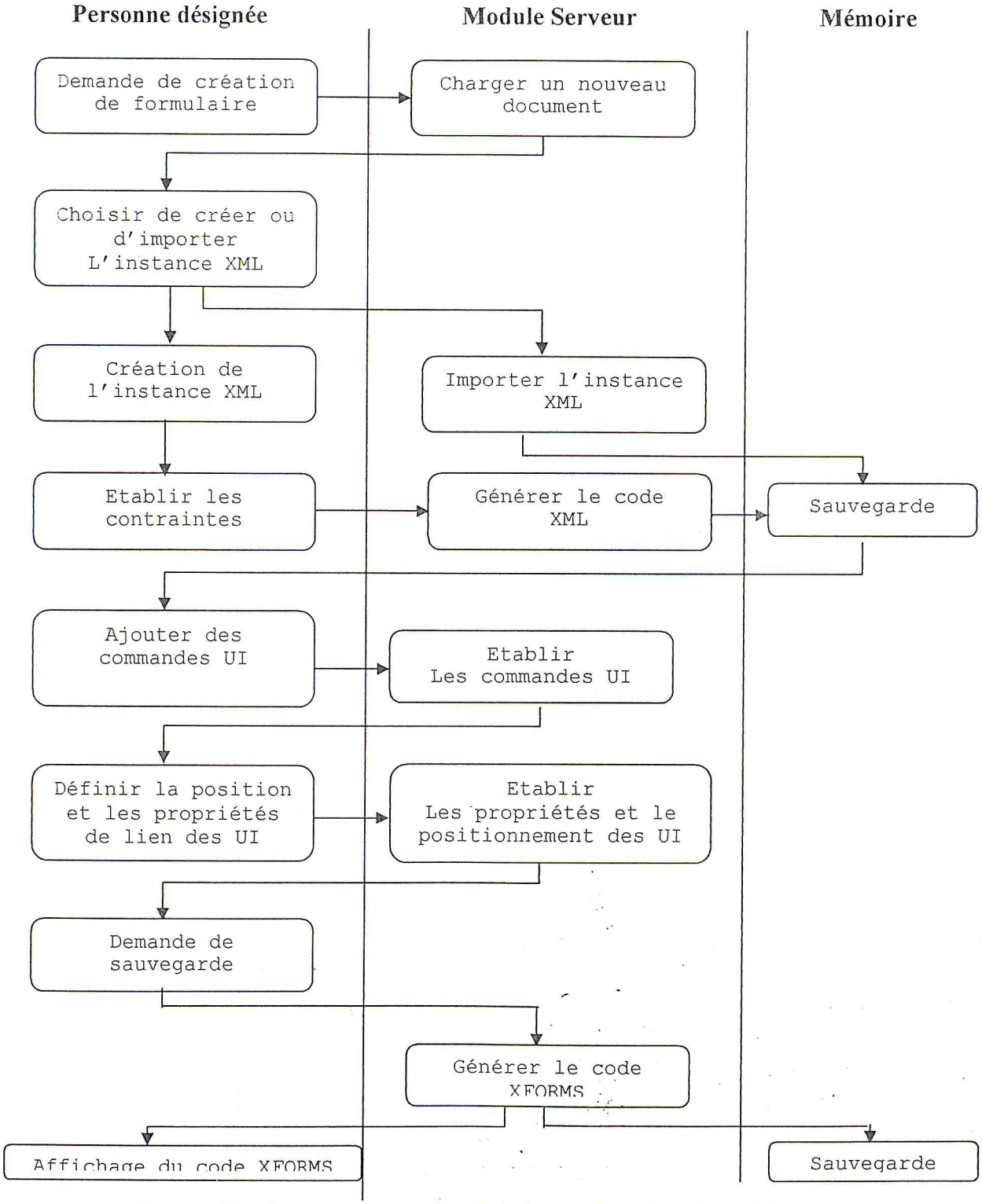
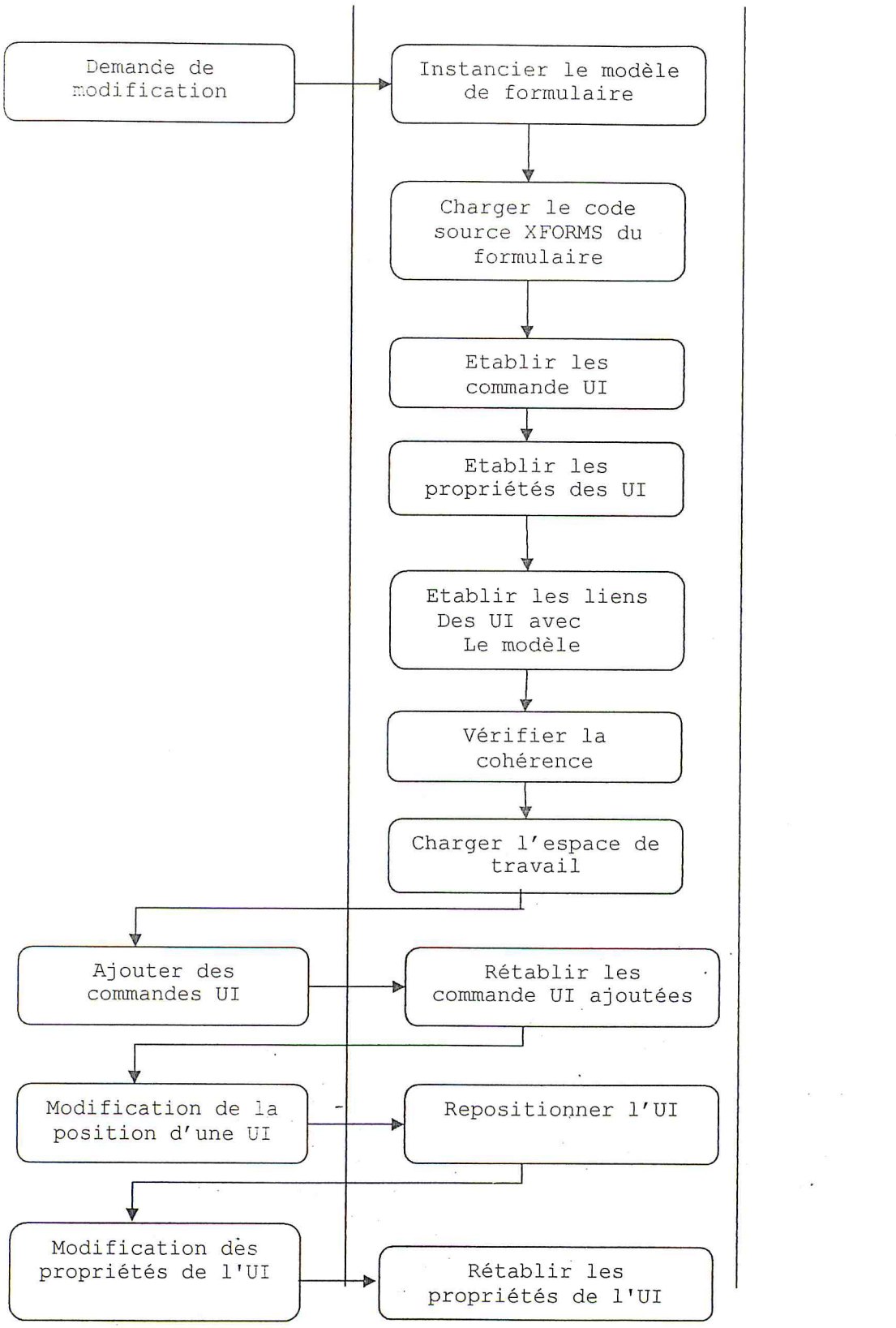


Figure4.52 : *Diagramme d'activité de la création d'un formulaire*


Personne qualifiée
Module Serveur
Mémoire
Demande de modification
Instancier le modèle de formulaire
Charger le code source XFORMS du formulaire
Etablir les commande UI
Etablir les propriétés des UI
Etablir les liens Des UI avec Le modèle
Vérifier la cohérence
Charger l'espace de travail
Ajouter des commandes UI
Rétablir les commande UI ajoutées
Modification de la position d'une UI
Repositionner l'UI
Modification dès propriétés de l'UI
Rétablir les propriétés de l'UI

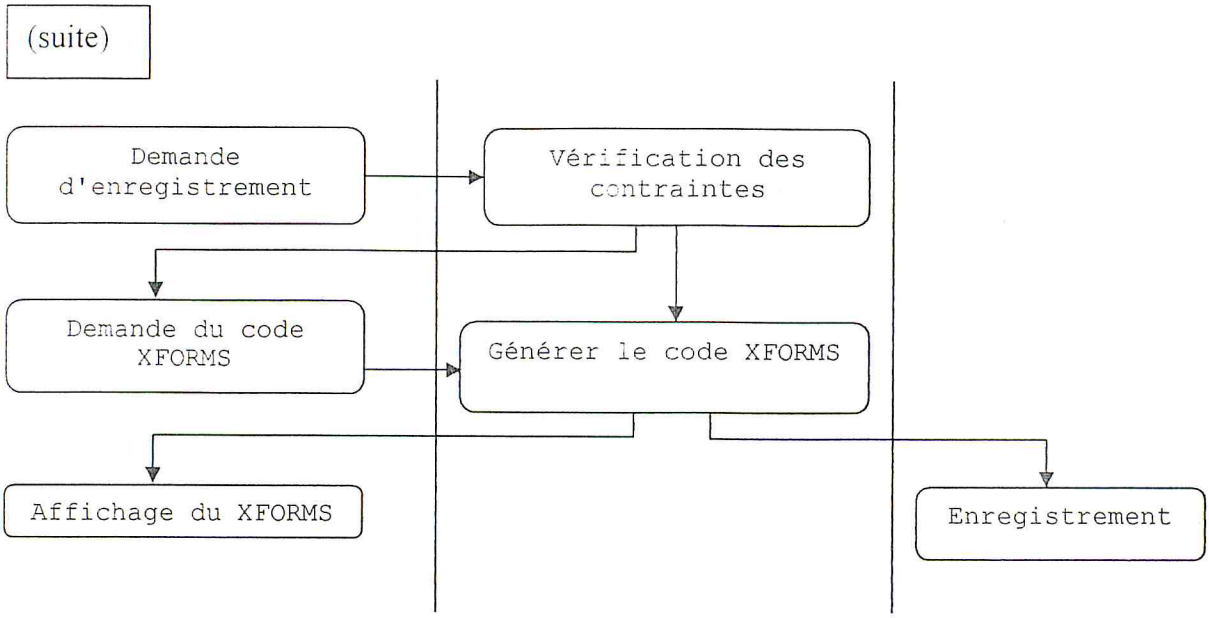


Figure 4.35 : *Diagramme d'activité de la modification du formulaire*

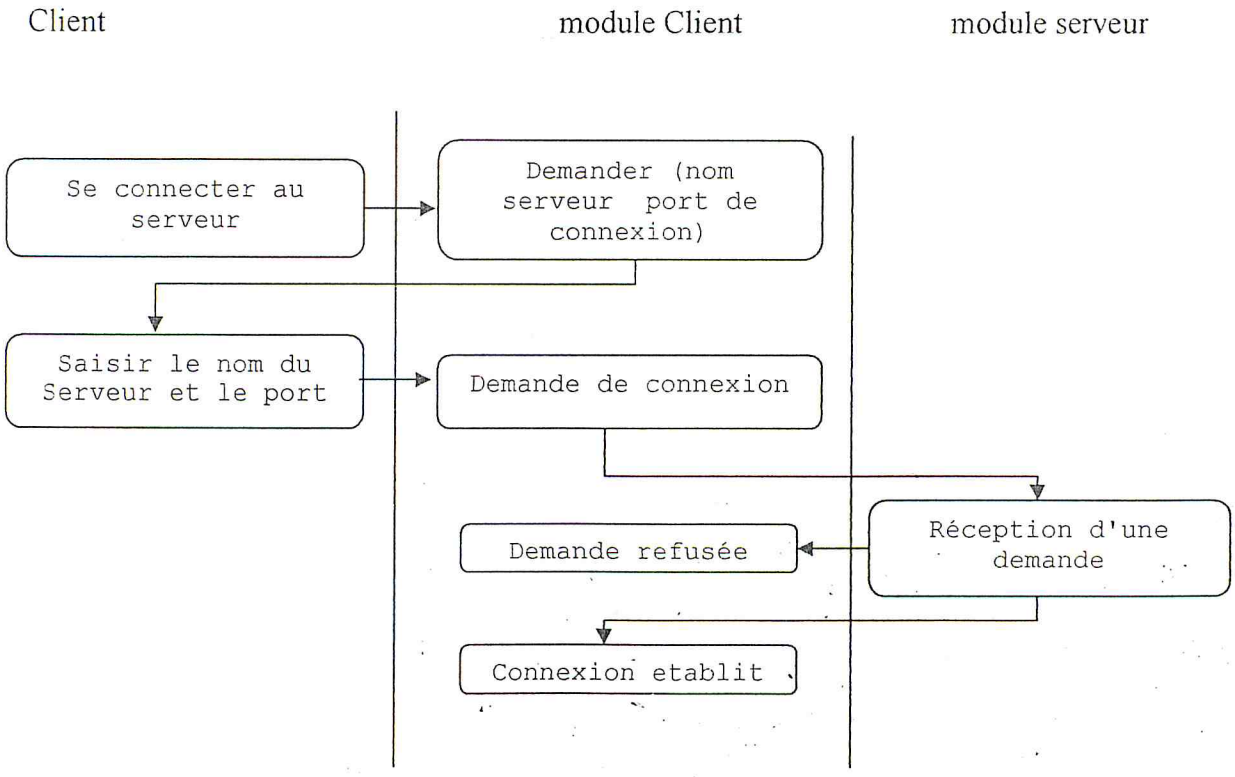


Figure 4.36 : *Diagramme d'activité de la connexion a un serveur*

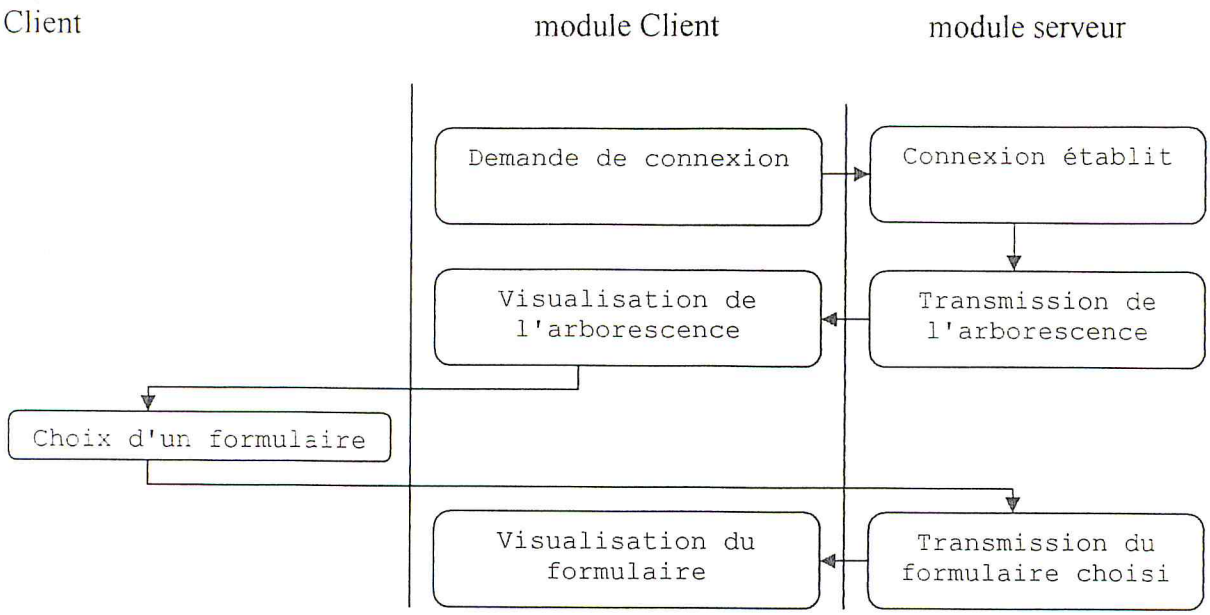


Figure 4.37 : *Diagramme d'activité du choix du formulaire*

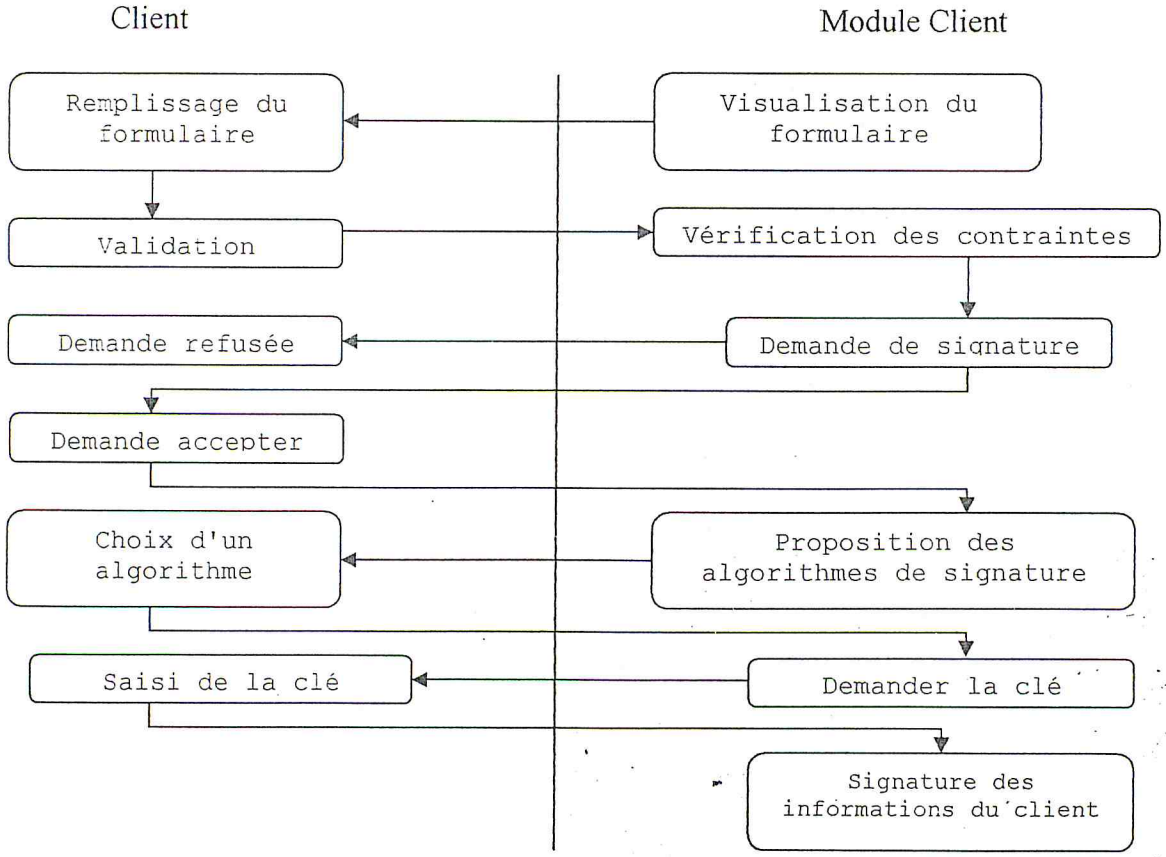


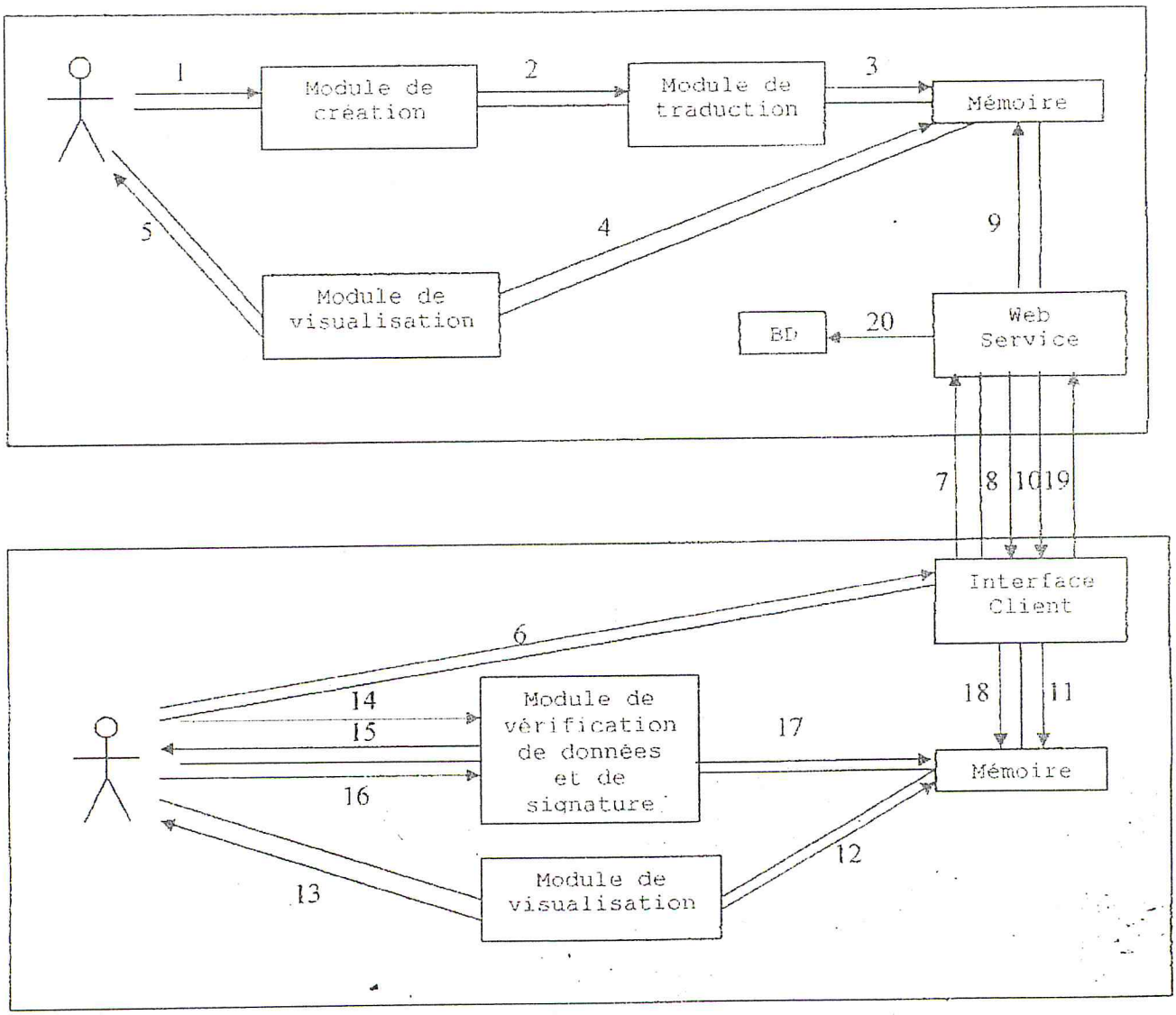
Figure 4.38 : *Diagramme d'activité de la signature du formulaire*

VI.Conception :

Dans cette partie nous nous intéressant au "comment", pour cela nous commençant par une conception générale décrivant l'architecture globale de notre système et ces différent composant et nous détaillerons par la suite chaque composant de notre systéme.

VI.1 Conception générale :

A fin d'obtenir une description de l'architecture du système et un ensemble de spécification de ses composants nous décomposons le logiciel en modules et on précise les interfaces et les fonctions de chaque module.
Pour commencer nous établirons un diagramme de collaboration dont on montre les différent modules composant notre système et leur interaction :



1: Demande de création.
2: Lire le code java
3: Génération et sauvegarde du code XForms.
4: Lecture du code XForms a partir de la mémoire
5: Visualisation
6: Demande de connexion au module client
7: demande de connexion au module serveur
8: Accepter la connexion.
9: Lecture du code XForms
10:Transfert des formulaires XForms
11 : Transfert des formulaires en mémoires.
12 : Lecture du formulaire choisi.
13 : Visualisation.
14 : Validation des informations du client.
15 : Demande de signer les informations du client.
16 : Réponse du client.
17 : Ecriture des informations signées ou non en mémoire.
18 : Lecture de l'information.
19: Transfert de l'information au serveur
20: Sauvegarder les information dans une BD

Figure 4.39 : *Diagramme de collaboration des différents modules*

VI.1.1 Architecture du système:

Cette partie donne un aperçu plus précis sur les différents modules proposé dans le diagramme de collaboration.

- **Serveur:**

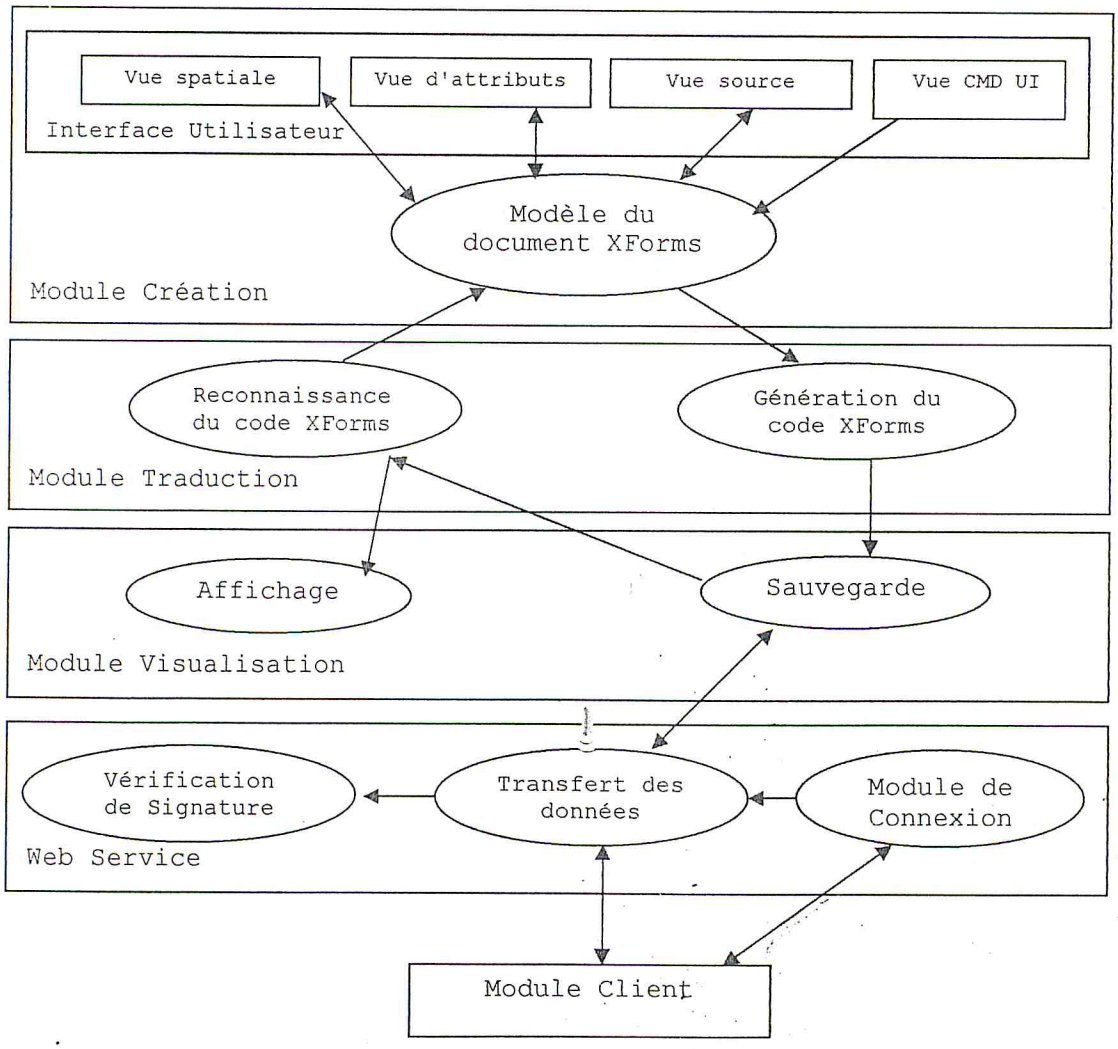


Figure 4.40 : *Architecture du sous-système Serveur*

- Classe " Lien ":

Elle gère les différents liens entre les commandes UI et les nœuds du modèle XML *(bind)*.

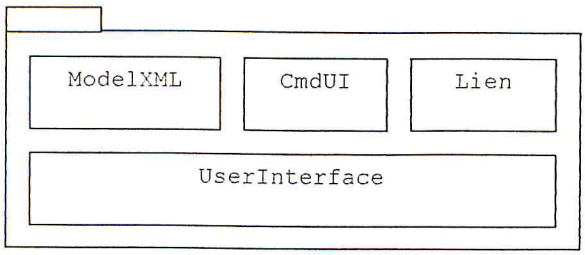


Figure 4.42 : *Package du module création*

VI.2.2 Le module traduction:

Ce module se divise en deux sous modules:

* Génération du code XForms.
* Reconnaissance du code XForms.

"Génération du code XForms": Ce module offre un service pour le module "création" afin de générer le code XFrorms adéquat au formulaire crée.

Il se compose de la classe JavaToXForms.

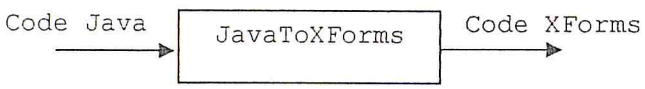


Figure 4.43 : *La classe JavaToXForms*

"Reconnaissance du code XForms": Utilisée par le module "création" *(lors de l'édition des formulaires, ie: modification)* et "visualisation".

Il a pour rôle de prendre en entrée du code XForms *(définit dans des documents XML, XSLT)* et de générer en sortie du code Java.

Il se compose de deux classes:

- CompilXForms: A pour rôle d'analyser les documents XML et XSLT et de générer un ensemble d'objets.
- TranslateObj: prend en entrée l'ensemble d'objets généré par la classe CompilXForms et de générer en sortie du code Java.

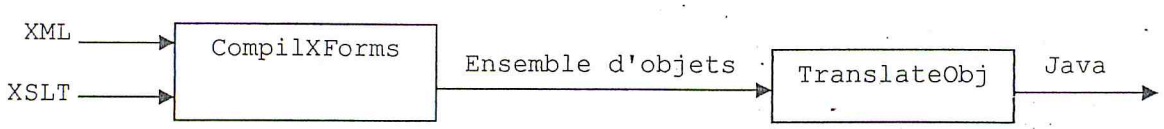


Figure 4.44 : *Lien entre CompilXForms et TranslateObj*

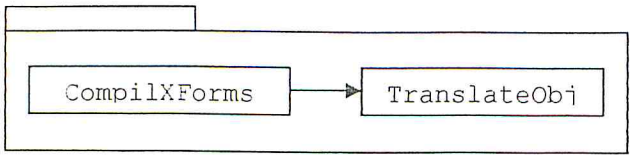


Figure 4.45 : *Package du sous module "Reconnaissance du code XForms"*

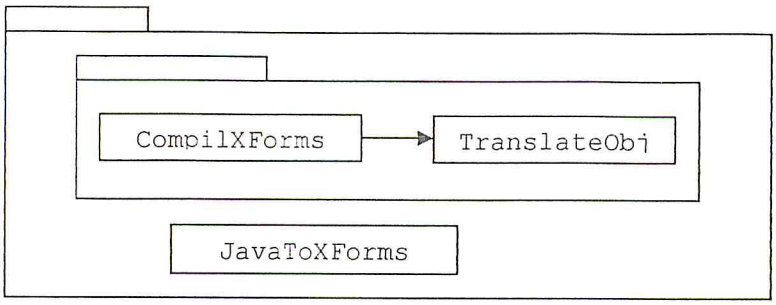


Figure 4.46 : *Package du module Traduction*

VI.2.3 Le module visualisation:

Serveur:

Ce module se compose de deux classes:

* SaveXForms: elle sauvegarde le formulaire XForms sous la forme de deux documents XML, XSLT.

* Affichage: Permet d'afficher le formulaire a partir du code java généré par le sous module "Reconnaissance du code XForms".

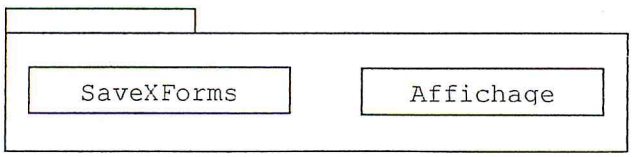


Figure 4.47 : *Package du module Visualisation*

Client:

A la différence du module de visualisation du serveur, celui-ci ne comporte pas de classe "SaveXForms", mais il incorpore le package "Reconnaissance du code XForms".

VI.2.4 Le Web Service:

Offre les fonctionnalités suivantes:

- Une interface de communication avec le client réaliser par le module de connexion *(géré dans notre cas par Tomcat).*
- Un module de transfert de donnés.
- Un support de vérification de signature numérique.

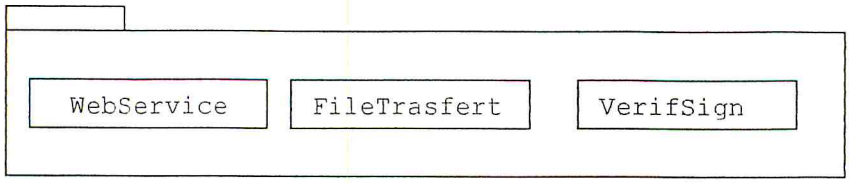


Figure 4.48 *: Package du Web Service*

VI.2.5 Le module Interface Client:

C'est le module responsable de la connexion et du transfert de données vers le Web Service, il se compose notamment de deux classes:

- ConnectToWS: Responsable de l'établissement de la connexion au Web Service. Son constructeur prend en argument l'adresse du Web Service.
- TransertData: Responsable du transfert de donnée entre le Web Service et le client.

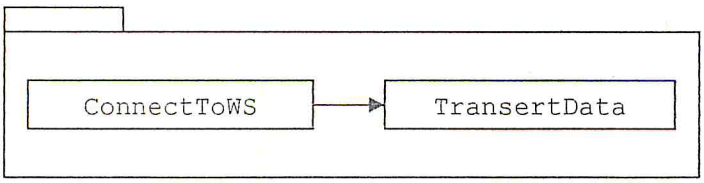


Figure 4.49: *Package Interface client*

VI.2.6 Le module de vérification et de signature:

C'est le module responsable de la vérification des contraintes imposées sur les données du formulaire, et de leur signature.

Il se compose de deux classes:

- CheckConst: Classe permettant de vérifier les contraintes sur les données du modèle XML, et demande a l'utilisateur s'il souhaite signer ces données.

Par exemple: l'heure d'arrivée doit être toujours supérieure a l'heure du départ.

- Signature: classe permettant d'appliquer une signature XML sur les données saisies par l'utilisateur.

Son constructeur prend en argument l'arbre de données XML, l'algorithme de signature la clé privée, et son certificat numérique.

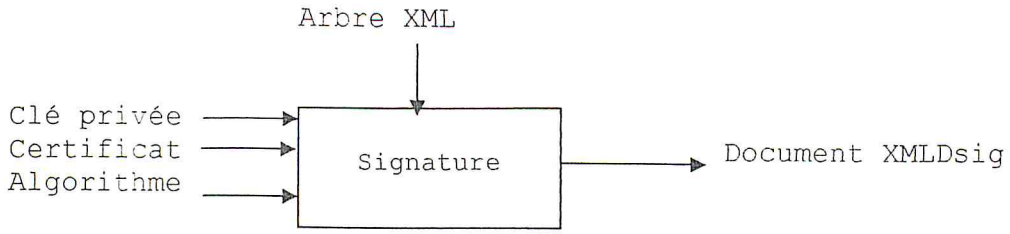


Figure 4.50 : *La classe signature*

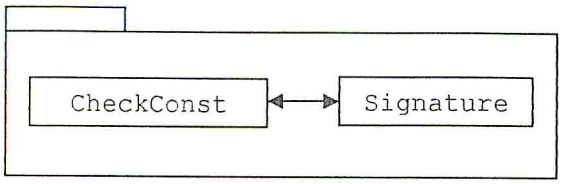


Figure 4.51 : *Package vérification et de signature*

Implémentation

- interface utilisateur:

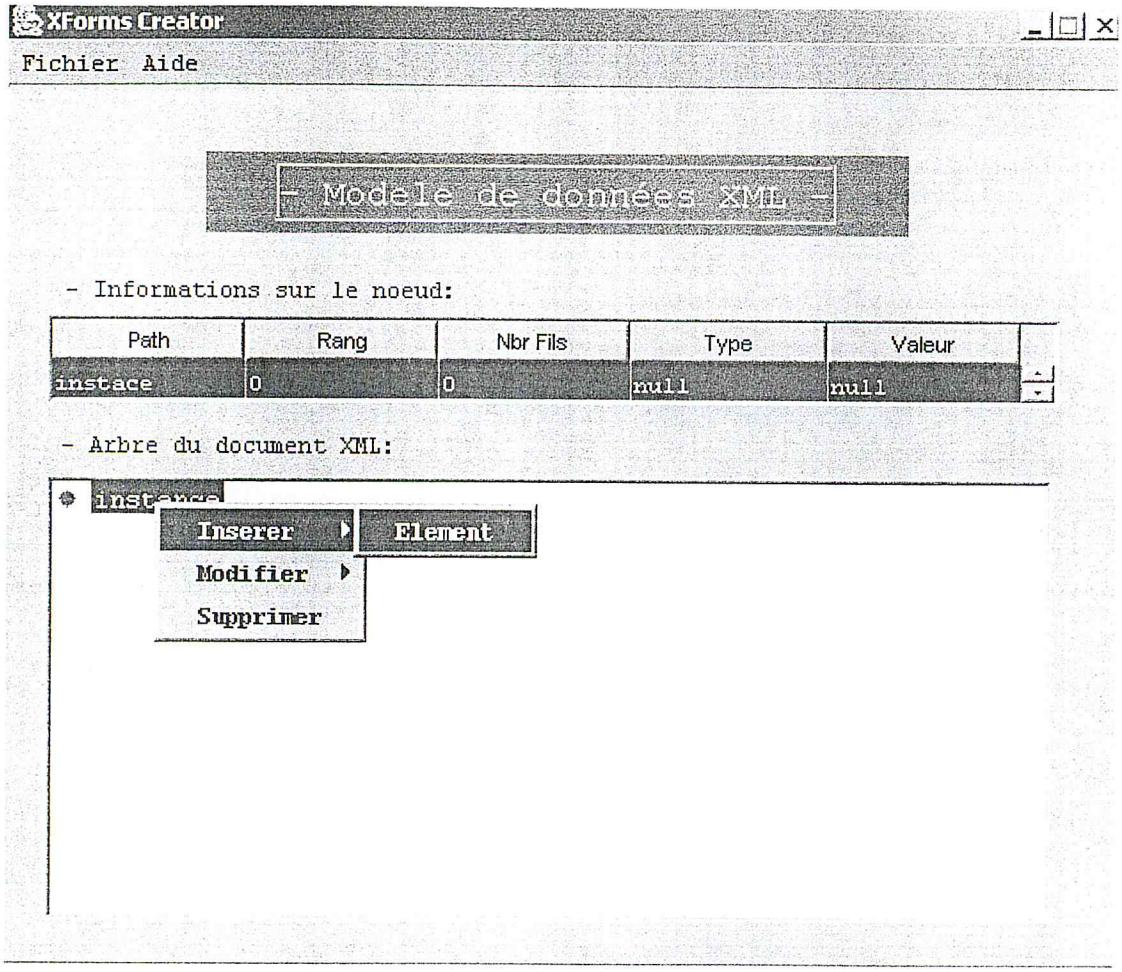


Figure 4.53 : *Insertion d'un élément.*

- Supprimer un nœud:

- Algorithme :

```
procedure supprimerNoeud(noeudSelectioné :Noeud)
debut
si NœudSelectioné != null alors
    si noeudSelectioné != racine alors
                supprimer(noeudSelectioné);
    sinon
          afficher('Impossible de supprimer <instance>');
    finsi
 finsi
fin
```

* Interface utilisateur :

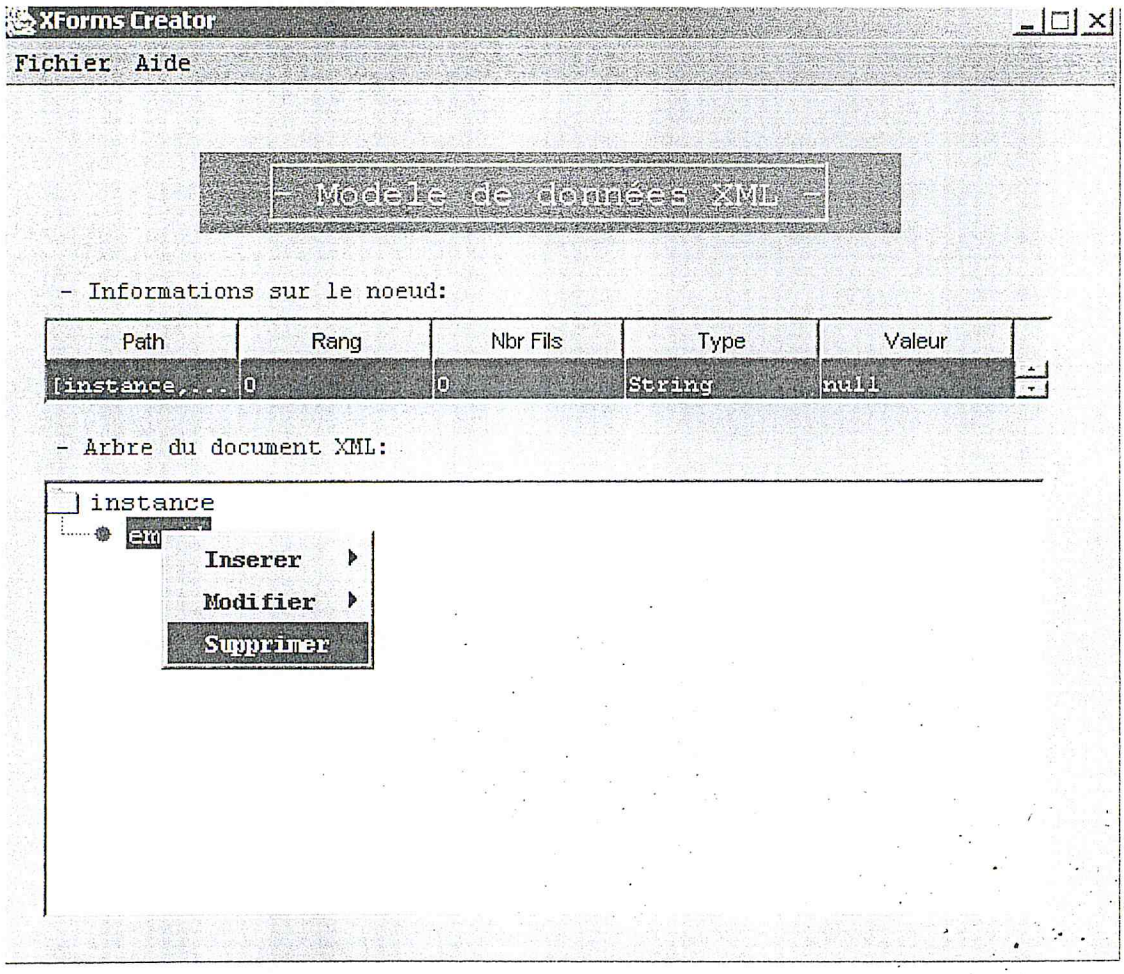


Figure 4.54 *: suppression d'un élément*

- Modifier le nom d'un nœud :

*Algorithme :

```
procedure ModifierNom(noeudSelectioné :Noeud)
debut
si NoeudSelectioné != null alors
  name : Chaine;
   NouveauNoeud : Noeud;
   afficher('Donnez un nom');
   Noeud papa = NoeudSelectioné.pere();
 si papa != null alors
                  NouveauNoeud(nom);
                  copier(NouveauNoeud , NoeudSelectioné);
        supprimer(NoeudSelectioné);
        NoeudSelectioné = NouveauNoeud;
        insererNoeud(NoeudSelectioné,papa);
   finsi
 sinon          afficher('Vous n'avez selectioné aucun noeud');
 finsi
fin
```

* Interface utilisateur :

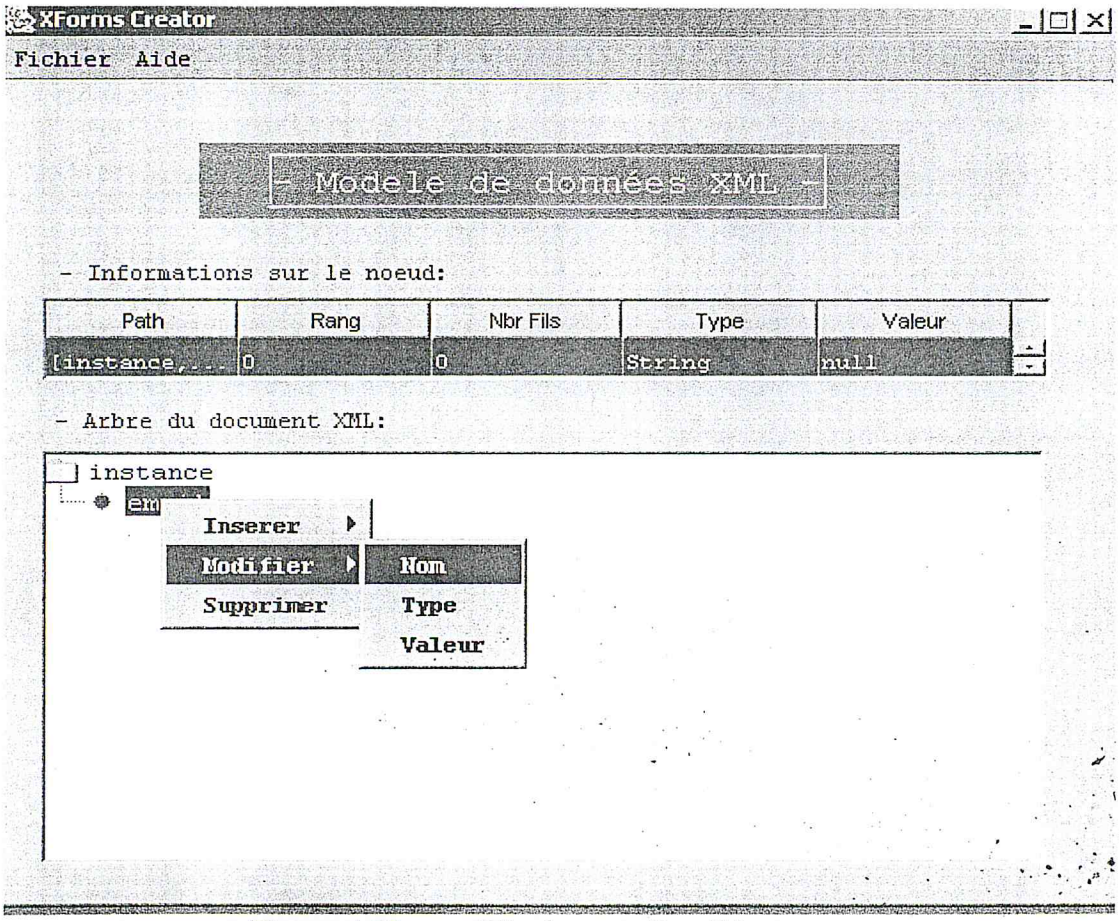


Figure 4.55 *: modifier le nom d'un élément*

- Modifier type:

* Algorithme :

```
procedure ModifierType(noeudSelectioné :Noeud)
debut
si noeudSelectioné != racine alors
  nouveauType : Type;
   afficher('Donnez type:');
   lire('nouveauType ');
   noeudSelectioné.changerType(nouveauType);
  sinon
       afficher('Le type de <instance> ne peut pas etre modifié');
  finsi
fin
```

*interface utilisateur :

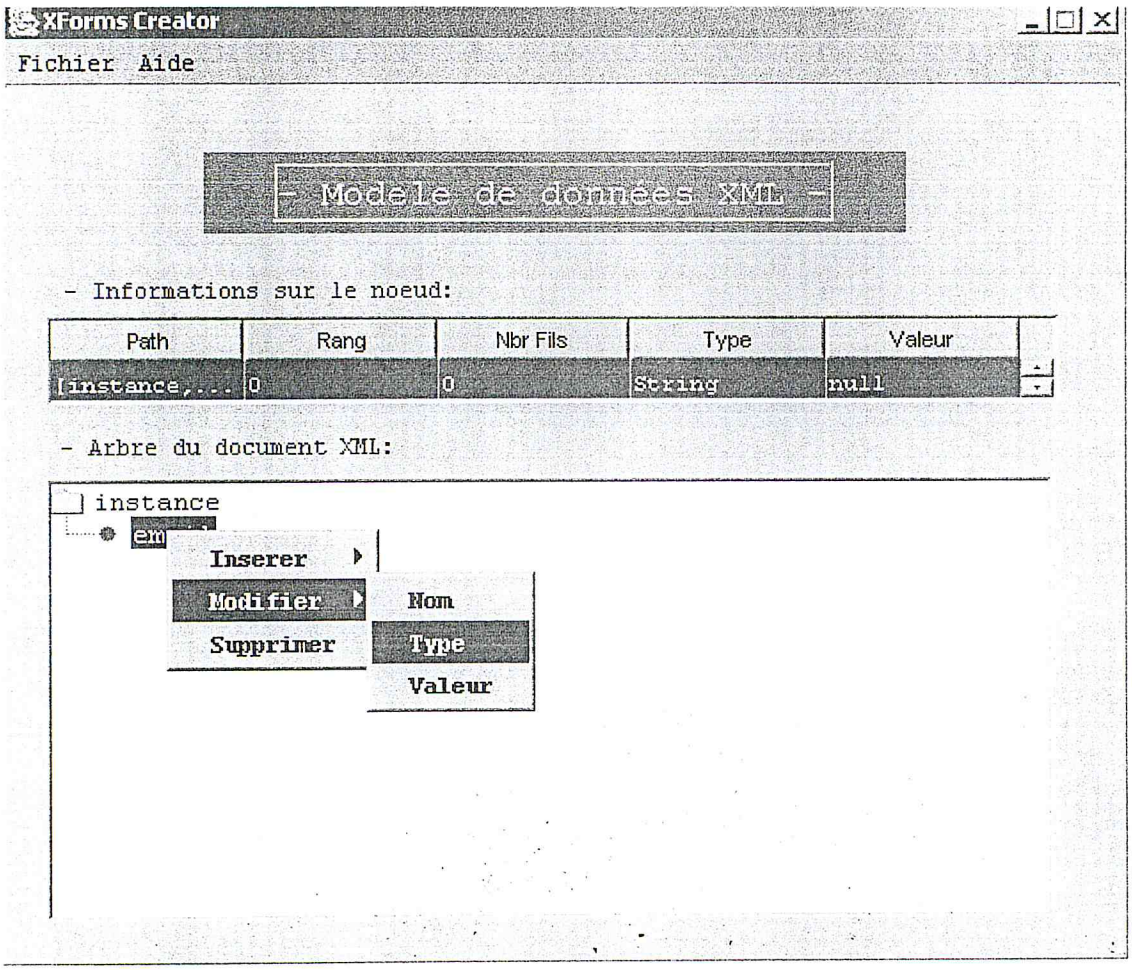


Figure 4.56 : *modifier le type d'un élément*

- Modifier la valeur d'un noeud

*Algorithme :

```
procedure ModifierValeur(noeudSelectioné :Noeud)
debut
valeur : Chaine;
 si noeudSelectioné != racine alors
      si noeudSelectioné.nbrFils == 0 alors
                  noeudSelectioné.insrervaleur(valeur);
        sinon
                afficher('Il faut que le nombre de fils soit égale a 0');
        finsi
 sinon
  afficher('la valeur de la racine ne peut pas etre MAJ');
finsi
fin
```

Interface utilisateur :

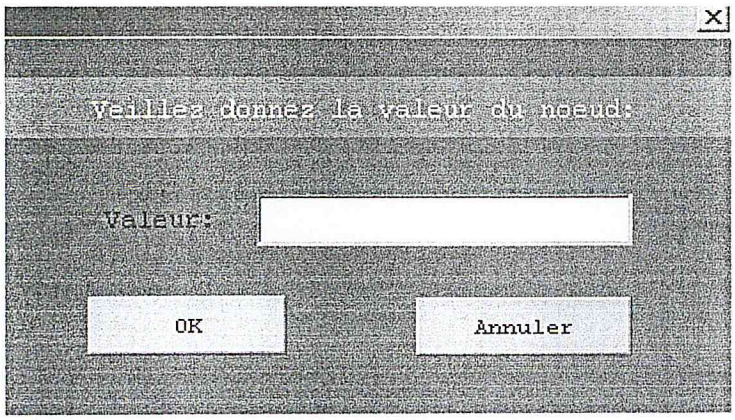


Figure 4.57 *: Modifier la valeur d'un noeud*

- Sauvegarde de l'instance XML :

* Algorithme:

```
Procedure  SauvegadeInsanceXML (chaine: nom, chaine: emplacement)
```

* Interface utilisateur :

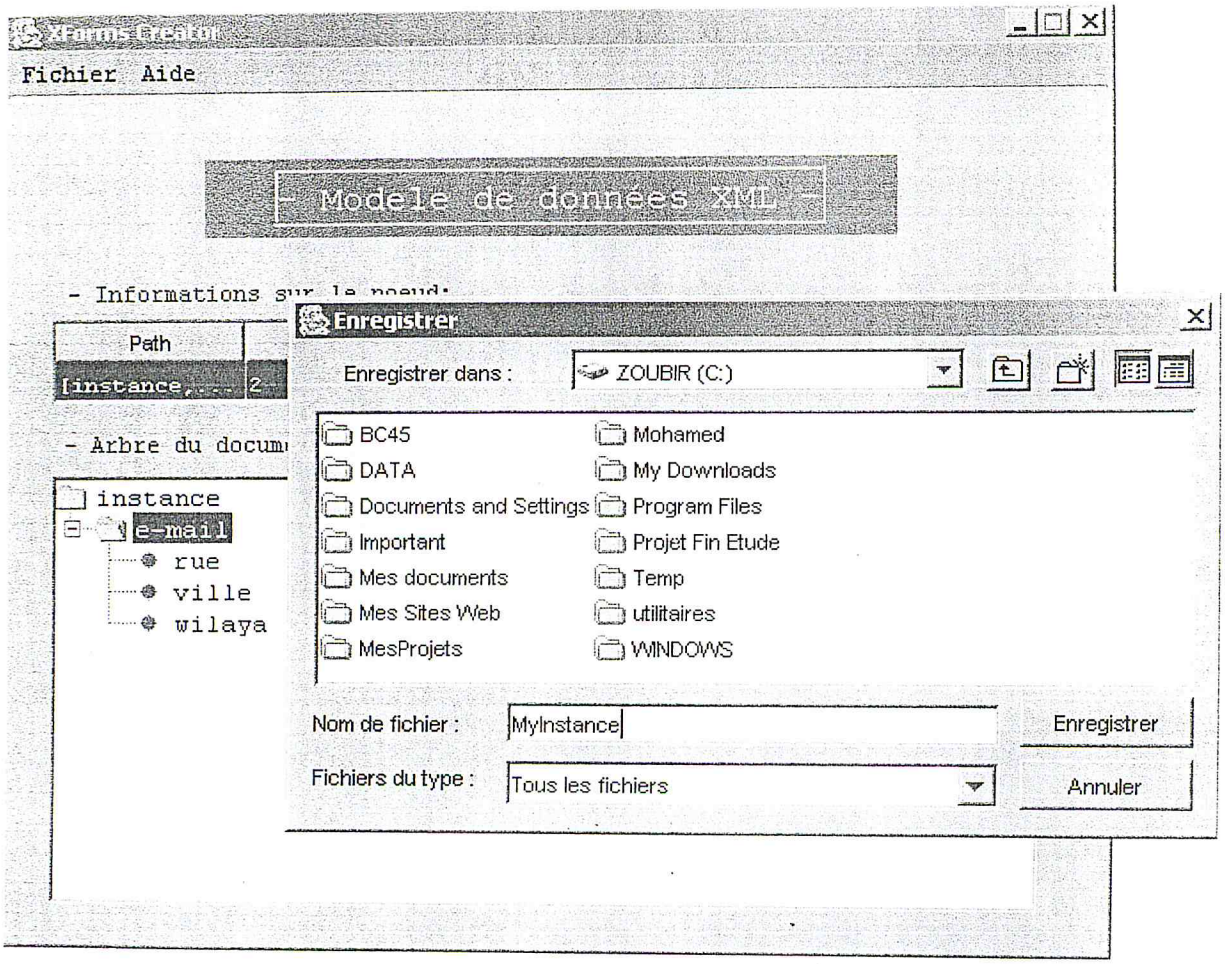


Figure 4.58 : *sauvegarde de l'instance XML*

VII.2.1.2 Implémentation de la classe "CmdUI":

Après la génération du model les commandes UI s'activerons a l'auteur XForms.

Cette classe gère la sélection des commandes d'interface utilisateurs définies dans le vocabulaire XForms *(input, secret, ...etc.)*

* Algorithme :

```
Procedure selectionner (cmd CUI)

 Debut

     Deselectioner(cuiSelectionée);
     cuiSelectionée = cmd;
  fin
```

* Interface utilisateur :

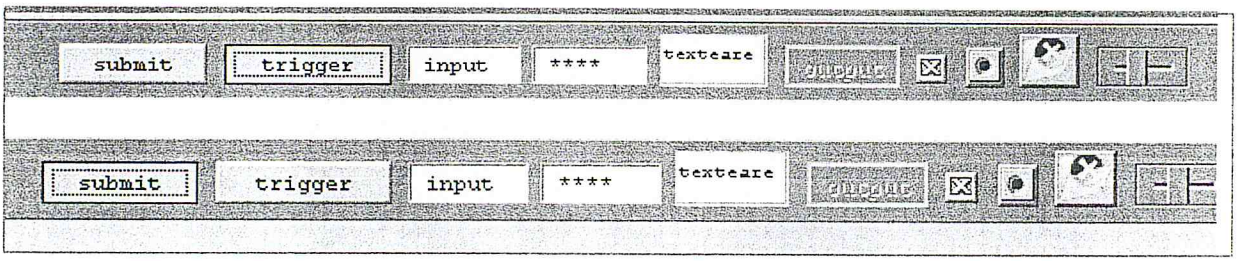


Figure 4.59 : *La sélection d'une commande UI*

VII.2.1.3 Implémentation de la classe "Lien":

- Algorithme:

L'utilisateur sélectionne une commande UI, et le nom du noeud qui va être lié a cet commande la fonction branchementNoeud les prends comme argument et établit la liaisons entre eux.

```
procedure branchementNoeud(cmd : CUI, noeud : Noeud)
  debut
          si Nœud=racine
                 afficher("instance ne peut pas être référencée")
          sinon
                cmd.texte = noeud.valeur;
                 cmd.type = noeud.type;
          finsi
  fin
```

- Interface utilisateur:

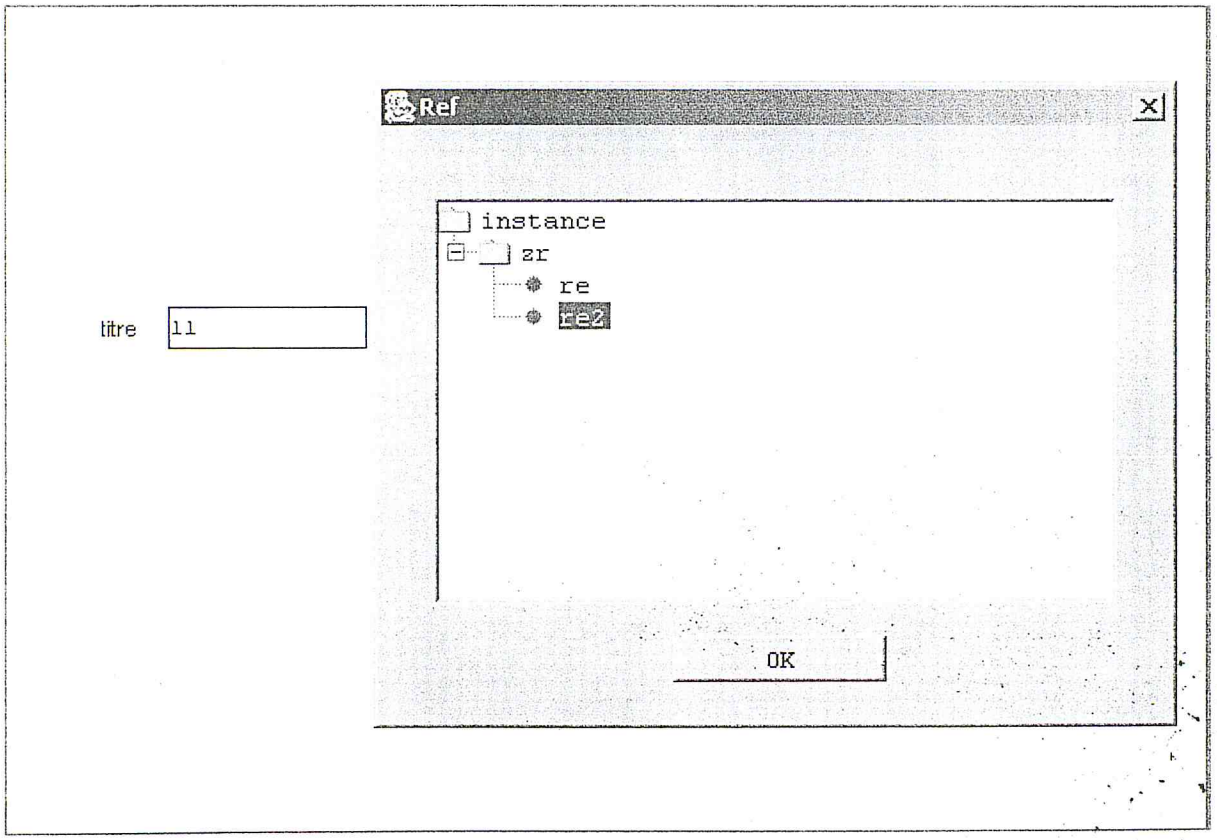


Figure 4.60 : *Lien entre le nœud "re2" et titre*

VII.2.1.4 La Classe " UserInterface " : c'est la classe responsable de l'interface utilisateur, plusieurs opérations tels que :

Insérer une commande UI, déplacer une commande UI, supprimer une commande, et la modification des attributs d'une commande, nous définissons ci-dessous les algorithmes de ces opérations:

- Les algorithmes :

```
procedure insererCUI(cmd : CUI,souris :Souris)

 debut
     cmd.x = souris.x;
     cmd.y = souris.y;
     cmd.largeur = 100;
     cmd.hauteur = 21;
     insrer(cmd,panneau);
 fin
```

```
procedure glisserCUI(cmd : CUI,souris :Souris)

  debut
     cmd.x = souris.x;
     cmd.y = souris.y;
  fin
```

```
procedure ChangerLargeurCUI(cmd : CUI, largeur : Entier)

 debut
     cmd.largeur = largeur;
 fin
```

```
procedure ChangerHeuteurCUI(cmd : CUI, Heuteur : Entier)

 debut
      cmd.Heuteur = Heuteur;
 fin
```

```
procedure SupprimerCUI(cmd : CUI, string: prss)

  debut
      si press = Supprimer alors
           supprimer(CUI);
      finsi
  fin
```

*Interface utilisateur:

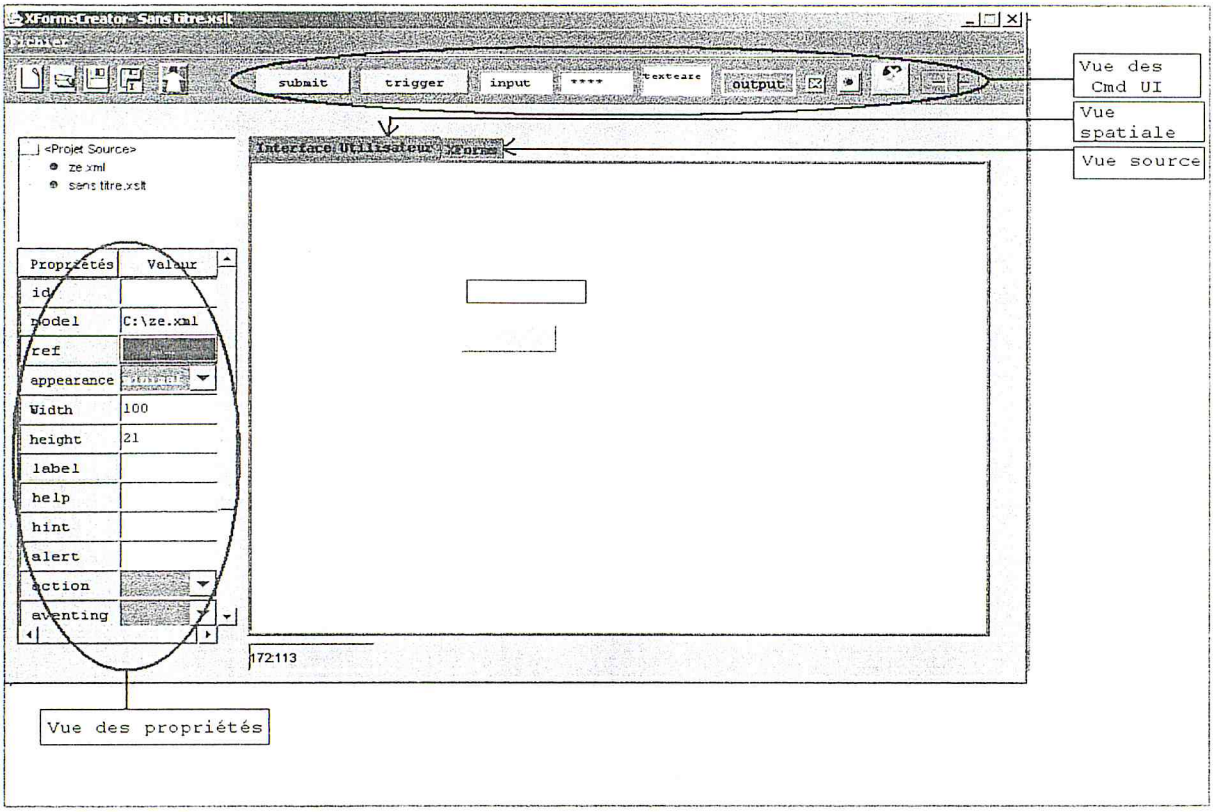


Figure 4.61 : *Interface utilisateur du module création*

VII.2.2 Implémentation du module "Traduction":

VII.2.2.1 Implémentation de la classe du génération du code XForms:

* Algorithme:

```
Procedure javaToXForms(objet : javaCUI,document Document)

Debut

Si objet = zoneDeTexte alors
  Inserer("<input>",document);
  Inserer(poroprietés-objet,document);
  Insrer("</input>");

finsi

Si objet = bouton alors
  Inserer("<trigger>",document);
  Inserer(poroprietés-objet,document);
  Insrer("</trigger>");

finsi

//... idem pour les autres commandes ui

Fin
```

* Interface utilisateur:

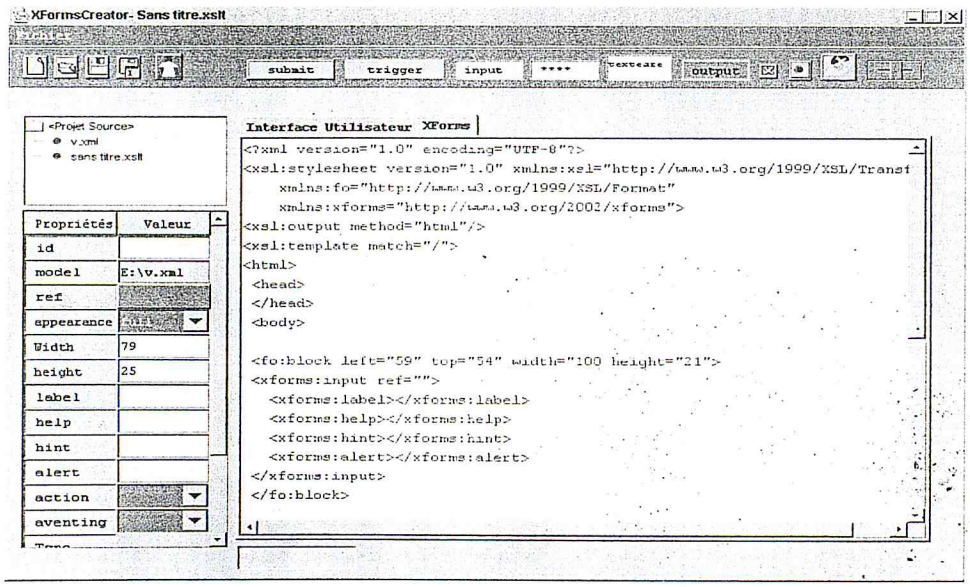


Figure 4.62 : *Génération du code XForms*

VII.2.2.2 Implémentation de la classe reconnaissance du code XForms:

Elle fait le travaille inverse de la classe précédente, c'est a dire qu'elle transforme le code XForms en code java pour cela nous utilisons l'API Xerces et implémentons l'interface DOM et SAX a la fois.

- La classe CompilXForms vérifie le code cohérence du code XForms et s'assure que le code est valide (*sans erreurs),* si l'opération se déroule sans erreur un ensemble d'objets est généré et stocké dans des tableaux dynamiques.

* Algorithme de compilation:

```
Precedure compilation(document : Document)

Debut
 docXMLAnalysé :Document;
 docXFormsAnalysé :Document;
 Objet : Tableau;
 docXMLAnalysé := AnylseXerces(document); // verifier les erreurs XML
  si erreur alors
            signaler;
            sortir;
  sinon
      docXFormsAnalysé  := analyserXForms(docXMLAnalysé);
      si erreur alors
            signaler;
            sortir;

      sinon
         objet := genererObjet(docXFormsAnalysé);

     finsi

  finsi

   retourner(objet);
Fin
```

- La classe TranslateObj:

Elle prend en argument le tableau d'objets généré par CompilXForms et génère du code java:

* Algorithme de compilation:

```
Procedure xFormsToJava(objet : Tableau,document: Document)

Debut
       Ouvrir(document);

       Pour i := 0 a objet.taille faire
   debut
                VerifierTypeObjet(objet.elment(i));
                insrerCodeJava(document);
  fin
      fermer(document);
Fin
```

VII.2.3. Implémentation du Web Service :

Le Web service est implémenté à l'intérieur du web Conteneur Tomcat de Apache, il est géré par son API Axis.

Toutes les interfaces de connexions sont gérées implicitement *(on n'implémente aucun socket).*

Le transfert de données se fait via le protocole SOAP *(permet d'envelopper un document XML).*

La seule procédure implémentée est celle de la vérification de la signature numérique.

Il dispose aussi des méthodes offrants divers service que le client peut invoquer:

- getArborescence : permettant d'avoir l'arborescence des formulaires se trouvant sur le serveur.
- getForm : permettant d'avoir le formulaire en question.

* Algorithme de vérification de signature:

```
Procedur verifSign (documentXML : Document)

Debut

  documentCanonizé : Document;
  docAnalysé : Document;
  teste : Boolean

  documentCanonizé := canonizer(documentXML);
  docAnalysé  := Parser(documentCanonizé);
  Element sign := recupererElemSignature(docAnalysé);

  Teste := testeSignature(sign);

  si teste = vrai
    continuer;

  sinon

    afficher('Signature invalide');

  finsi

Fin
```

* Algorithme d'obtention de l'arborescence de formulaires:

```
Procedure getArborescence(racine : repertoire)

Debut

      Formulaires : liste;
      repFils : liste;

      insererListeFormulaires(arbre);
           // insérer nom formulaires dans arbre

      rep = sousRepertoires(racine);

      si rep = null alors
                          retourner (arbre);
      sinon

        pour i := 0 a rep.taille
              getArborescence(rep.element(i));

      finsi

Fin
```

* Algorithme d'obtention du formulaire:

```
Procedure getrForms (nom : Chaine)

Debut

      fileXML : Fichier;
      fileXslt : Fichier;
      document : Document;
      nomXslt : Chaine;

      fileXML = Fichier("nom"+".xml");
      document := analyser (fileXML);
      nomXslt := document.getXSLT();

      fileXslt := Fichier(nomXslt);

      tabFile : Tableau;
      tabFile[0] := fileXML;
      tabFile[1] := fileXslt;

      retourner(tabFile);

Fin
```

Le code source du Web service sera enregistré sous l'extension de JWS dans le reperoire racine d'Axis.

VII.2.4. Implémentation du module de signature:

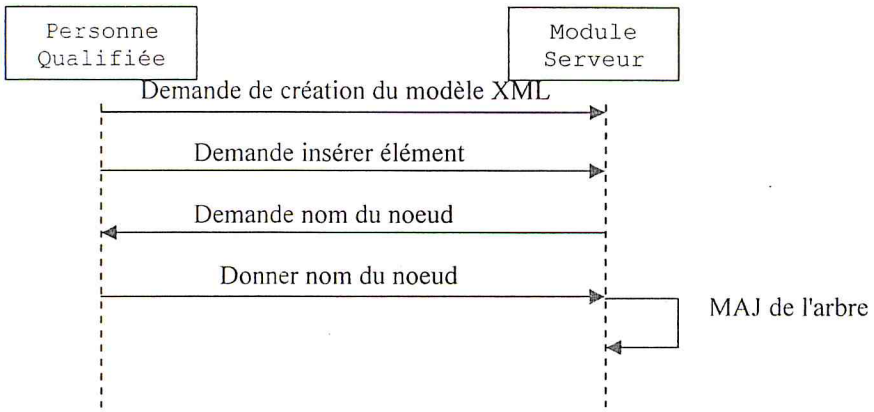
```
Procedure signer(keyp:Cléprivée, alg:Algorithme, cert:Certificat, inst:Document)

Debut
     SignedInfo,Signature :Element
     Digestvalue := algorithmedeDigetion(document);
     Inserer(SignedInfo,digestvalue);
     Inserer(SignedInfo,digestmethode);
     Inserer(SignedInfo,alg);
     SignedInfoCanoniser := AlgorithmedeCanonisation(signedinfo);
     SignatureValue := sign(alg,SignedInfoCanoniser,keyp);
     Inserer(Singature,signedInfo);
     Inserer(Singature,SignatureValue);
     Inserer(Singature,cert);
     Retourner(Signature);
Fin
```

Dans cette partie nous allons tester quelque cas d'utilisation, des deux application crées (Client, Serveur) qui se résume en la création, visualisation et signature d'un formulaire XForms.

VIII.1 Création d'un modèle XML :

Pour créer un formulaire l'utilisateur doit passer par la création du model.

La création se fait en appuyant sur le bouton "nouveau":



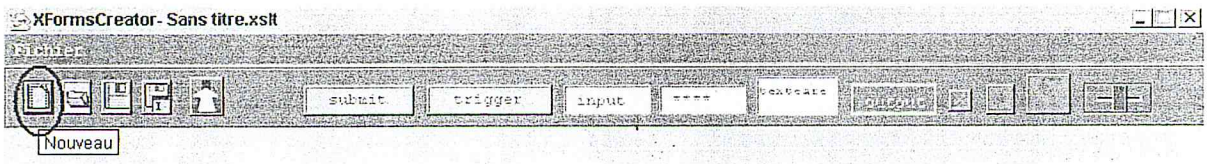


Figure 4.63 : Demande de création du model

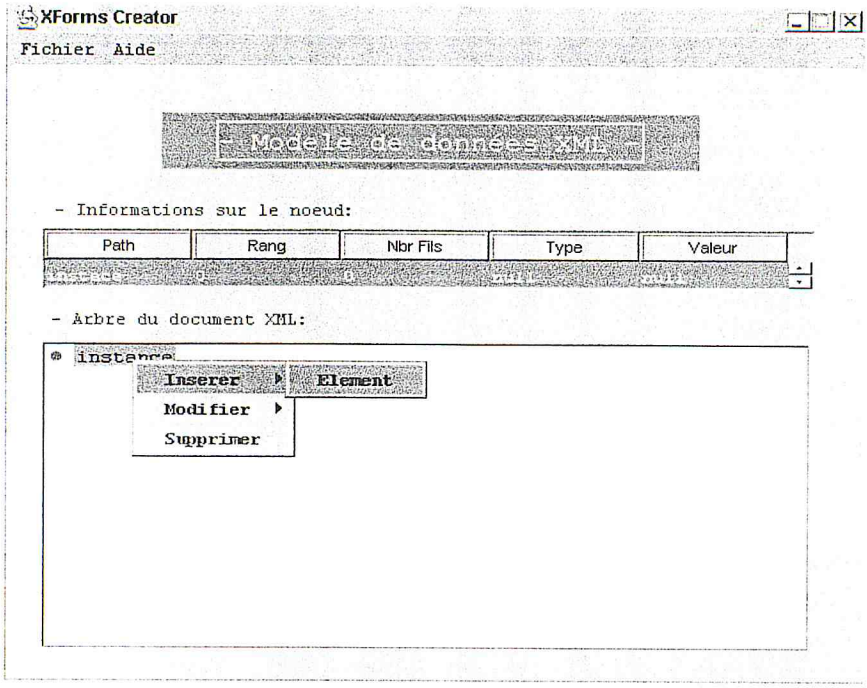


Figure 4.64 : Demande insertion du noeud

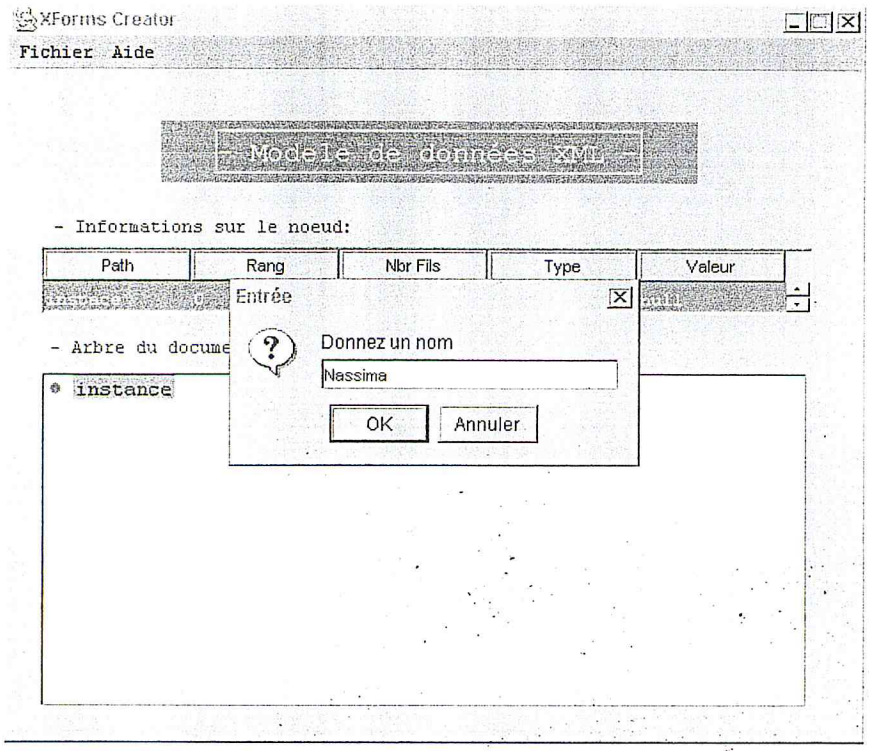
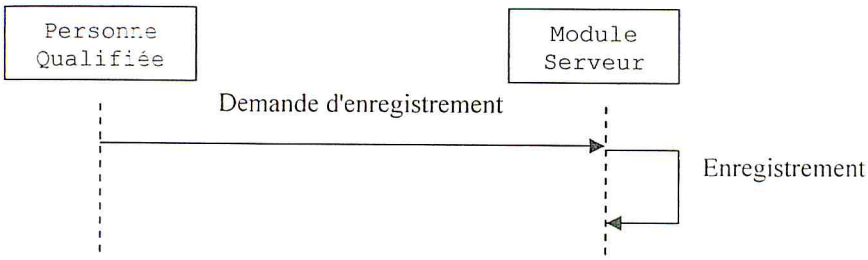


Figure 4.65 : Demande nom du noeud

VIII.2 Enregistrement du modèle XML :



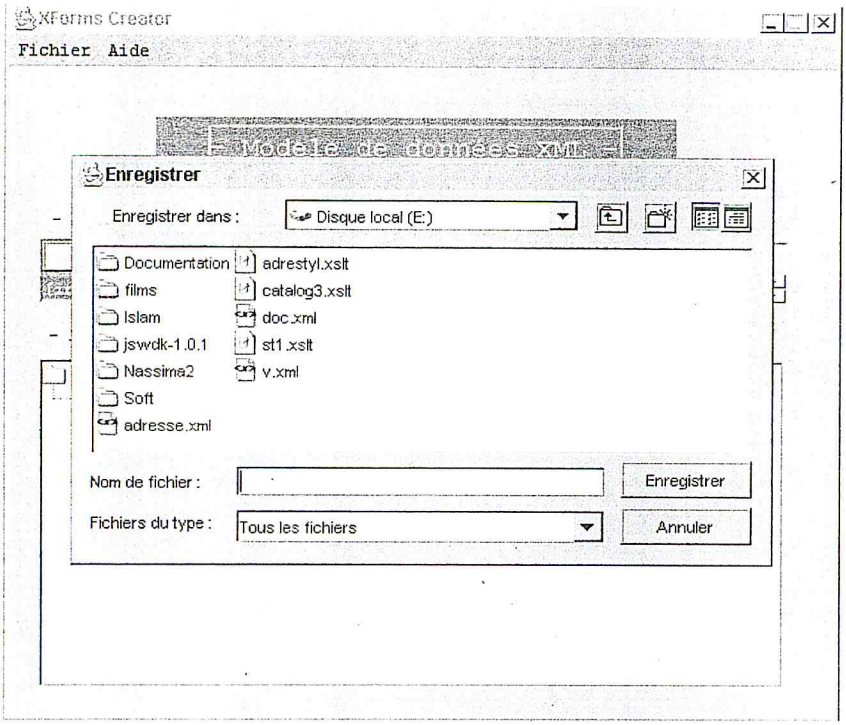
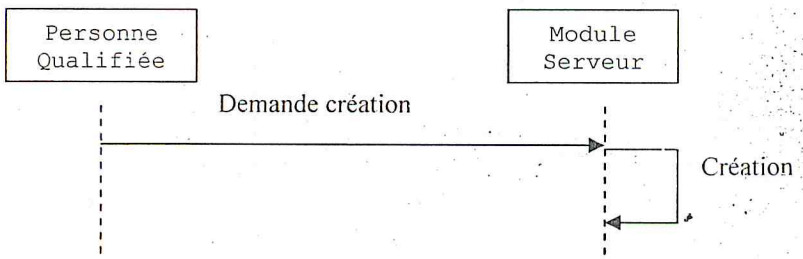


Figure 4.66 : Enregistrement du modèle XML

VIII.3 Création du formulaire:



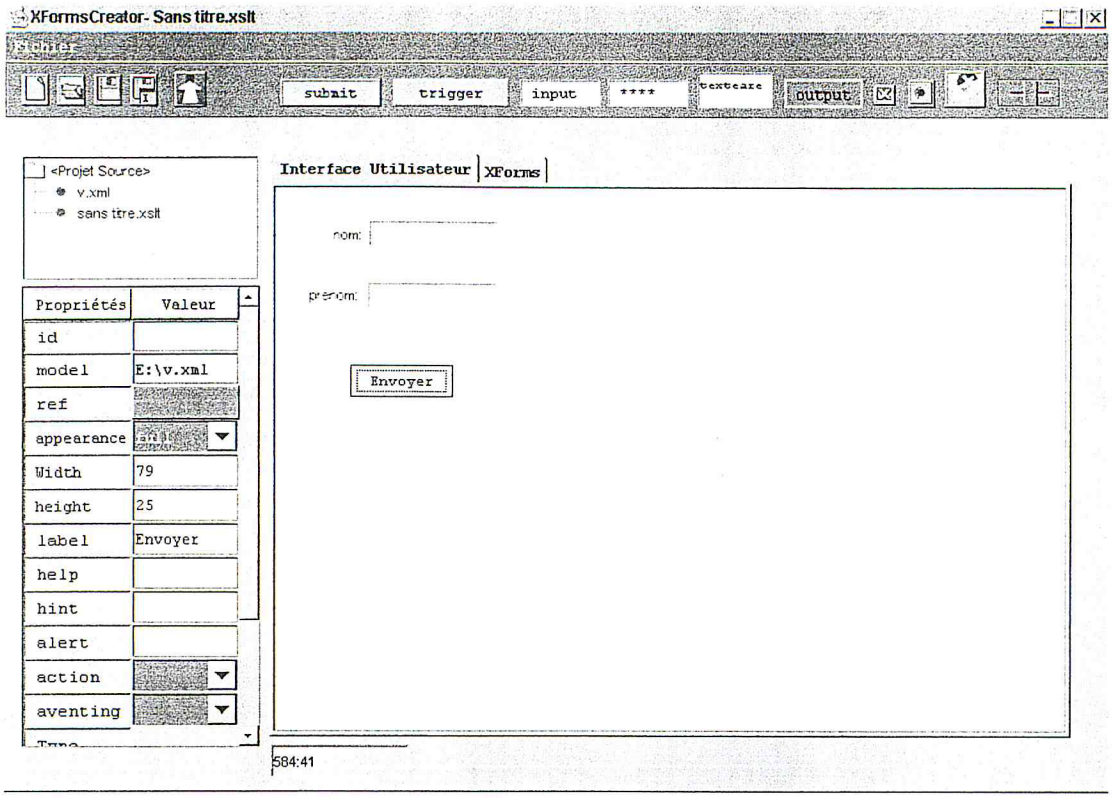
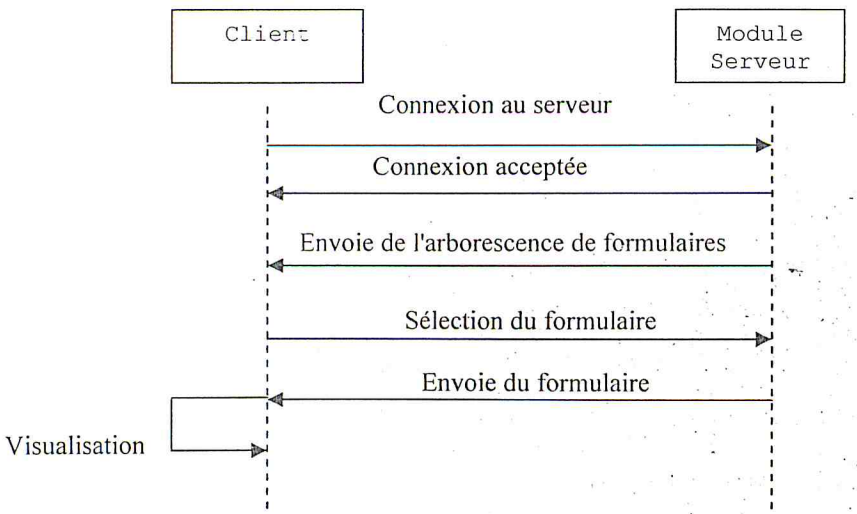


Figure 4.67 : Création du formulaire

VIII.4 Visualisation du formulaire:

Pour visualiser un formulaire, le client doit se connecter au serveur, le serveur lui envoi l'arborescence des formulaires, après ça le client choisi le formulaire pour la visualisation.



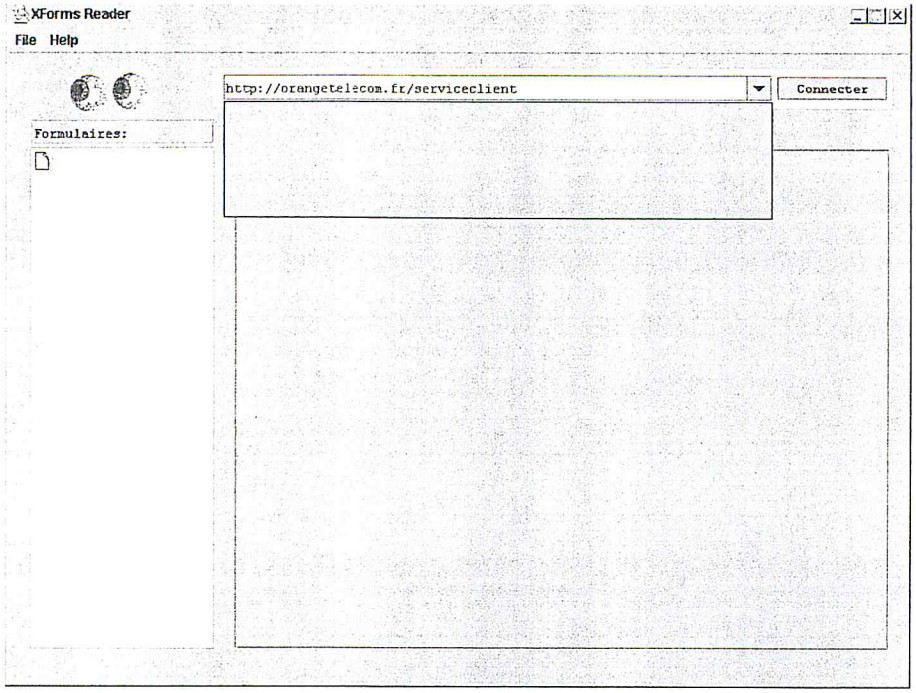


Figure 4.68 : Connexion au serveur

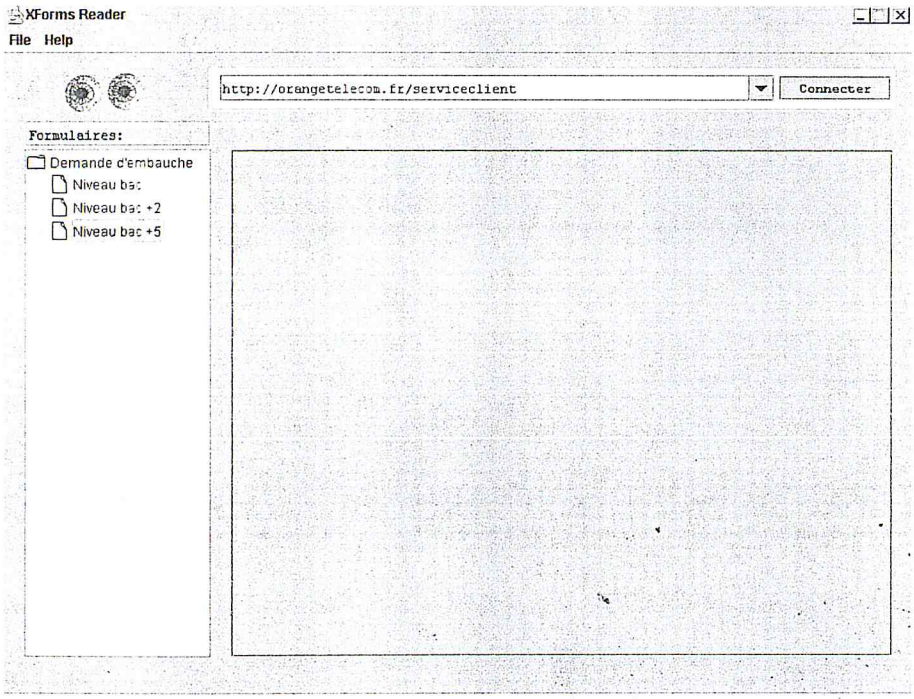


Figure 4.69 : Choix du formulaire

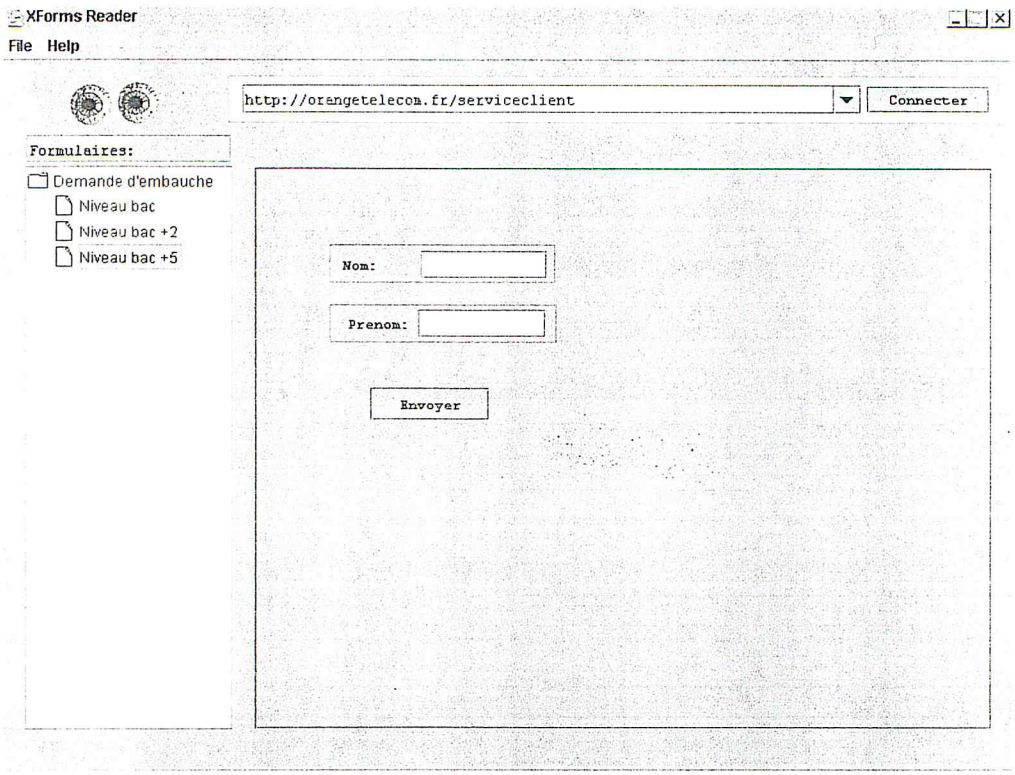
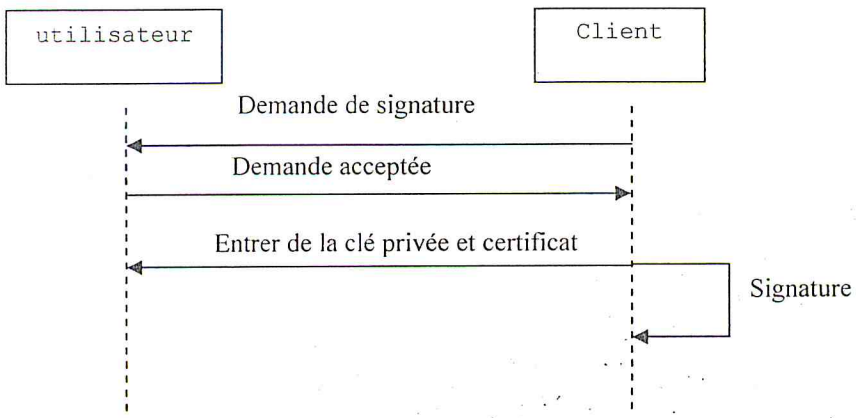


Figure 4.70 : Visualisation du formulaire

VIII.5 Signature du formulaire:

En voulant soumettre le formulaire, il est demandé à l'utilisateur de le signer

L'utilisateur donne sa clé privée le chemin vers son certificat.



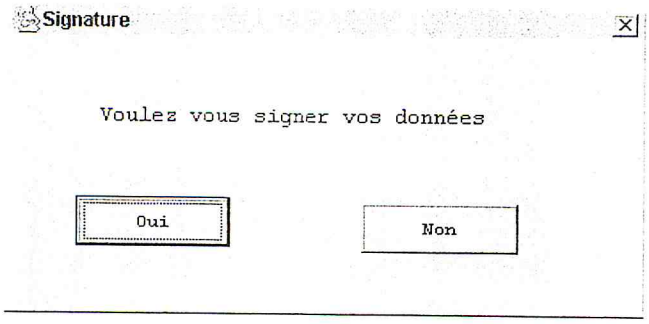


Figure 4.71 : Demande de signature

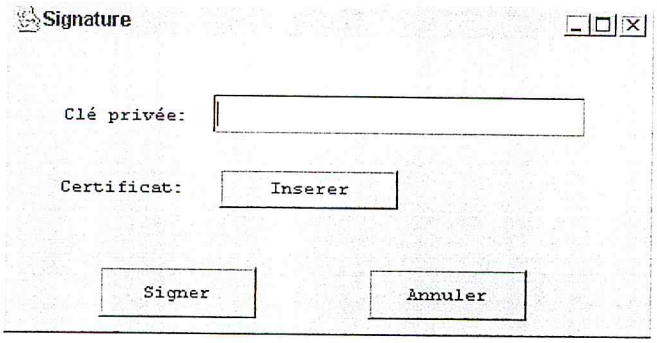


Figure 4.72 : Signature des données

IX Conclusion :

Au cours de ce chapitre, nous avons présenté, la mise en œuvre de notre système en commençant avec l'outil d'édition de formulaire XForms sous le format XML lié avec une feuille de style XSLT, cette dernière fait appel aux élément (balise) XFoms par le biais de l'espace de nommage xmlns:xforms="http://www.w3.org/2002/xforms", ainsi nous avons implémenter un processeur XFomrs capable d'analyser les élément XForms et de générer la commande d'interface utilisateur correspondante. Il sera invoqué lors de la visualisation.

Nous avons aussi implémenté la partie client capable de se connecter a notre serveur et de visualiser les formulaires XForms, Cette même partie peut aussi d'appliquer une signature numérique XML sur les données saisies, qui sera alors vérifiée par le serveur.
Pour cela nous avons utilisé une conception orienté objet.

Nous avons entamé le développement par une identification des besoins de l'utilisateur en terme d'échange de formulaires électroniques *(Facilité de création, fiabilité de transfert des données, climat de confiance).* L'Architecture de notre système répond a toutes ces exigences.

Conclusion générale:

Dans ce mémoire nous avons conçu un système d'échange de formulaires normalisés et signés.
Comme norme d'échange nous avons choisi la spécification XML qui définit deux autres sous normes XForms dans le domaine de formulaires et XMLsDIG dans le domaine de sécurité.

Nous avons pus réaliser un simple prototype d'un système d'échange de formulaires, qui peut être sujet à d'éventuelles améliorations, ceci est due a une omission de quelques balises XForms tels que l'élément bind qui applique des contraintes sur les données *(readonly, requiered...)*, ainsi que des fonctions *(numériques, booléennes, ...)*, et certains actions et événements.
Il faut signaler que le XForms est a lui seul un langage vaste et complet qui demande des années de travail. Aussi un pareil système doit être réalisé par tout une équipe de programmeurs pour être mené a terme.

Ainsi dans le domaine de sécurité, notre système offre un climat de confiance, il répond a :

- L`intégrité des informations émises en appliquant des algorithme de hachage pour calculer une empreinte du document transmis *(c`est cette empreinte qui sera signée)* .
- L`authentification, en utilisant des algorithmes de signature très puissant tels que DSA, qui utiliser des clés privées très longues difficile a les décrypter.

- La non-repudiation est assurée en vérifiant les certificat de l`émetteur, mais puisque nous avons pas utiliser l`extension XAdES, elle n`est pas assurer a long terme.

Cet outil sera utilisé dans les deux domaines, commercial *(e-commerce)* par toutes entreprise cherchant un moyen de récolte d'informations, et administratifs *(demande de conge)*.