MILIMAIL

# Spécifications techniques MiliMail

**Date : 28/05/2007**
**Version : 020**
**Référence : Spécifications techniques Milimail**

# Table des Matières







# 1 Description du document

Ce document décrit les spécifications techniques visant l'implémentation de MiliMail.

L'architecture des différents modules à intégrer y est détaillé.

# 2 Introduction

Ce document a pour but de décrire techniquement les différents points suivants :

- L'architecture générale du client de messagerie MiliMail sous forme de plugins et extensions s'intégrant dans le logiciel Mozilla Thunderbird 2.
- L'intégration du Format XSMTP 1.1 pour l'envoi et la réception des messages.
- L'intégration d'extensions dans les manipulations de l'annuaire partagé au sein du client.
- L'intégration des technologies MDN et DSN pour la gestion des accusées de réception.
- L'intégration de plusieurs stratégies de sécurité non implémentées dans Thunderbird.

# 3 Présentation de l'architecture

Pour bien saisir l'architecture de MiliMail, nous allons définir deux mécanismes qui permettent à MiliMail d'être défini comme une extension de Mozilla Thunderbird.

## *3.1 Extensions*

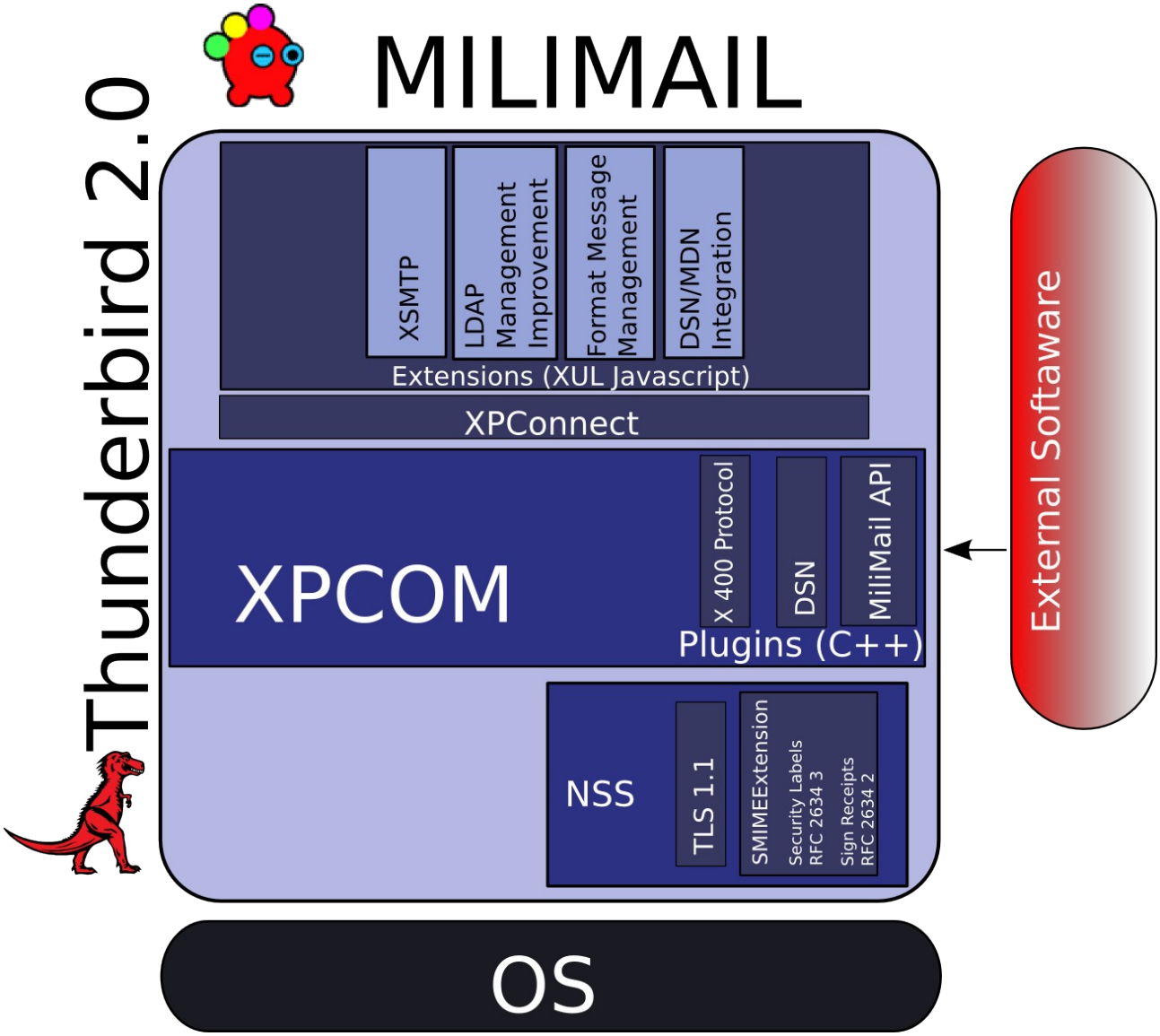
Une extension est selon les termes de Mozilla, un greffon à intégrer à Thunderbird pour lui apporter de nouvelles fonctionnalités ou modifier celles déjà existantes. L'extension est à distinguer du Plugin (définition ci-après). En effet, l'extension est par essence multi-plateforme car elle s'appuie sur deux langages interprétés par le client : Le Javascript et le XUL.

L'extension est distribué sous la forme d'une archive unique XPI.

À travers une extension, L'interface graphique de Thunderbird peut être modifier, et certaines fonctionnalités du coeur du logiciel (XPCOM) peuvent être instrumentalisées.

De plus, l'extension permet de modifier le coeur même du programme au niveau de la couche XPCOM. Grâce à ce processus, on ajoute des interfaces (XPIDL) scriptables par la couche (XUL/Javascript). Le développement s'effectue en C++ ou en Javascript (le C++ est néanmoins conseillé) et s'appuie sur un framework bas-niveau pour assurer la portabilité sur tous les systèmes supportés par Thunderbird.

## 3.2 Architecture globale




### 3.2.1 Couche de présentation

La couche de présentation se base sur le langage XUL (définie ci-après). Ce langage est interprété dynamiquement par le moteur d'affichage Gecko. Les communications de la couche d'affichage vers la couche métiers XPCOM se font à travers XPCONNECT. Ce module permet l'invocation de méthodes sur certains objets XPCOM en Javascript.

### 3.2.2 XPCOM

XPCOM représente la couche métier de Thunderbird, elle propose à travers des composants C++ toutes les fonctionnalités métiers de l'application. Chaque composant est accessible via XPIDL, une couche à la CORBA, permettant l'interopérabilité entre plusieurs langages.

### 3.2.3 NSS

NSS (Network Security Service) est le composant gérant la sécurité de Thunderbird. Il est écrit en C et est appelé par la couche XPCOM pour toutes les fonctionnalités traitant de la sécurité.

# 4 Langages utilisées

Les langages utilisées dans l'élaboration de la solution sont définis ci-après.

## *4.1 XUL*

Le XUL (XML User Interface) est un dialecte XML créé par Mozilla qui permet de décrire des interfaces graphiques riches et multi-plateformes. Cette technologie est utilisée dans tous les produits proposés par Mozilla (Thunderbird, Firefox etc...). A l'instar du HTML, le XUL peut être utilisé conjointement avec des feuilles de styles CSS et du javascript pour ajouter des contrôles dynamiques et des requêtes métiers.

Références :

http://www.w3.org/TR/REC-xml/

http://www.mozilla.org/projects/xul/xul.html

http://developer.mozilla.org/en/docs/XUL_Reference

## *4.2 Javascript*

Le Javascript est utilisé an parallèle du XUL pour ajouter des contrôles dynamiques, des requêtes métiers et une manipulation plus fine de l'interface graphique.

La norme supporté par Thunderbird est :

- ECMA-262 Edition 3

http://www.ecma-international.org/publications/standards/Ecma-262.htm

Références :

http://developer.mozilla.org/en/docs/Core_JavaScript_1.5_Guide

http://developer.mozilla.org/en/docs/Core_JavaScript_1.5_Reference

http://developer.mozilla.org/en/docs/About_JavaScript

## 4.3 C++

Le langage C++ est utilisé dans la couche métier de Thunderbird appelée XPCOM. Cette couche orientée composant (à l'instar de COM sous Windows) est codée de façon à être indépendante de la plateforme. Une simple recompilation permet son utilisation sur Windows, Linux et MacOSX.

Références :

http://www.mozilla.org/hacking/coding-introduction.html

http://www.mozilla.org/hacking/mozilla-style-guide.html

## 4.4 XPIDL

La couche métier XPCOM de Thunderbird est accessible par la couche de présentation XUL/Javascript. Afin de stabiliser les interfaces, et proposer une description des fonctionnalités indépendantes de l'implémentation (C++). Mozilla a introduit un langage de description d'interface orienté Objet qui est inspiré du monde CORBA : le XPIDL (XPCOM Interface Description Langage).

En pratique, à chaque composant XPCOM est associé un fichier XPIDL de description de ces interfaces. Grâce à l'utilitaire XPIDL, peuvent être générés : les headers C++ permettant l'implémentation ainsi que des fichiers .XPT permettant au moteur XPCONNECT de fournir une instrumentalisation des objets XPCOM à la couche de l'interface.

Références :

http://developer.mozilla.org/en/docs/XPIDL

## 4.5 RDF (Resource Description Framework)

RDF est un format permettant de stocker des données hiérarchisées. Ces données peuvent être stockées dans un fichier RDF ou format XML, mais il est également possible de formatter des données depuis une source dans un autre format afin de les utiliser facilement. Par exemple, Mozilla utilise ce format pour les bookmarks, l'historique ou les mails.

Ces sources de données RDF peuvent servir à remplir des arbres, des listes déroulantes, des menus etc ...

Références :

http://www.xulplanet.com/tutorials/xultu/intrordf.html

# 5 Convention de Code

Milimail est en une solution Open Source s'appuyant sur Mozilla Thunderbird, le produit adopte les mêmes conventions de code que celles promues par la fondation Mozilla.

## 5.1 XML / XUL

Le XUL écrit au sein de MiliMail doit être lisible et bien formé. De ce fait, chaque fichier XUL écrit sera indenté et commenté et muni de la licence Mozilla dans son entête.

## 5.2 Javascript

Le Javascript utilisé en parallèle du XUL sera commenté le plus possible et formaté en accord avec la politique de développement de Mozilla.

Références :

http://developer.mozilla.org/en/docs/JavaScript_style_guide

Pour plus d'informations, il est intéressant de se référer au document normes_prog_milimail.pdf

## 5.3 C++

Le C++ utilisée dans la couche XPCOM de MiliMail doit lui aussi se conformer aux directives de Mozilla. En effet, c'est la couche la plus sensible du produit. La portabilité et la lisibilité du code doivent être assurées. La licence Mozilla sera intégrée dans l'entête de chaque fichier.

Références :

http://www.mozilla.org/hacking/mozilla-style-guide.html

http://www.mozilla.org/hacking/portable-cpp.html

Pour plus d'informations, il est intéressant de se référer au document normes_prog_milimail.pdf.

## 5.4 XPIDL

Les fichiers XPIDL de description des interfaces XPCOM sont eux aussi soumis à des règles de syntaxes et de formatages dictés par Mozilla. Pour une bonne intégration au sein de l'environnement Mozilla, Milimail se force à les respecter.

Références :

http://www.mozilla.org/scriptable/xpidl/idl-authors-guide/index.html

# 6 Version 0

## *6.1 XSMTP*

### 6.1.1 Principe général

Le format XSMTP est un un format recommandé par le document édité par la DGA 2005/101978/CELAR/ASC/AP/03875 version 1.1. Ce format consiste à intégrer des entêtes supplémentaires dans le corps du message.

Référence :

http://www.milimail.org/milimail/documents/Recommandation_format_XSMTP_V11.pdf

### 6.1.2 Liste des entêtes à intégrer

Cette liste d'entêtes est issue de la spécification XSMTP 1.1 DGA

| Désignation | Émission (écriture des entêtes) | Réception (Affichage) |
|---|---|---|
| X-P772-Version | Optionnel | Obligatoire |
| X-P772-Priority-Level-Qualifier | Optionnel | Optionnel |
| X-P772-Extended-Grade-Of-Delivery | Optionnel | Optionnel |
| X-P772-Primary-Precedence | Obligatoire | Obligatoire |
| X-P772-Copy-Precedence | Optionnel | Obligatoire |
| X-P772-Message-Type | Optionnel | Obligatoire |
| X-P772-Address-List-Indicator | Optionnel | Obligatoire |
| X-P772-Exempted-Address | Optionnel | Obligatoire |
| X-P772-Extended-Authorisation-Info | Obligatoire | Obligatoire |
| X-P772-Distribution-Codes | Optionnel - Conditionnel | Obligatoire |
| X-P772-MCA | Optionnel - Conditionnel | Obligatoire |
| X-P772-Handling-Instructions | Optionnel | Obligatoire |
| X-P772-Message-Instructions | Optionnel | Obligatoire |
| X-P772-Codress-message-indicator | Optionnel | Obligatoire |
| X-P772-Originator-Reference | Obligatoire | Obligatoire |
| X-P772-ReferenceIndication | Optionnel | Obligatoire |
| X-P772-Other-Recipient-Indicator | Optionnel | Obligatoire |
| X-P772-Acp-Message-identifier | Optionnel | Obligatoire |
| X-P772-Originator-PLAD | Optionnel | Obligatoire |

| | | |
|---|---|---|
| X-P772-Acp-Notification-Request | Optionnel | Obligatoire |
| X-P772-Acp-Notification-Response | Optionnel | Obligatoire |
| X-P772-Security-Classification | Obligatoire | Obligatoire |
| X-P772-Special-Handling-Instructions | Optionnel | Obligatoire |

### 6.1.3 Implémentation

Le format XSMTP doit être implémenté sous forme d'une extension Thunderbird (http://www.mozilla.org/projects/thunderbird/specs/extensions.html) . Cette extension couvre deux points techniques :

- La modification de l'interface graphique pour intégrer les entêtes dans
    - La fenêtre de réception des messages (INBOX)
    - La fenêtre d'envoi des messages
    - Un contrôle permettant d'activer ou non les fonctionnalités XSMTP (CIC-1)

#### *6.1.3.1 Présentation*

L'architecture technique de cette extension respecte les techniques mises en place dans les extensions de Mozilla déjà existantes.

Deux technologies sont utilisées : le XUL (XML User Interface Language) et le Javascript.

Le XUL est utilisé pour décrire l'interface graphique de l'extension et le Javascript permet d'utiliser les fonctions exposées par le framework Mozilla (XPCOM http://www.mozilla.org/projects/xpcom/). L'intégration du code de l'extension dans le code existant de Thunderbird s'effectue à travers une technique appelée par Mozilla *l'Overlay (*http://developer.mozilla.org/en/docs/XUL_Overlays*)*. Cette technique permet l'ajout de fonctionnalité sans toucher au code existant de Thunderbird.

#### *6.1.3.2 Packaging d'une extension*

L'extension est packagé sous la forme d'un fichier au format *XPI*. Ce format de fichier est une archive *ZIP* ayant la structure suivante :

**extension.xpi**

- extension/
  - install.rdf
  - chrome.manifest
  - content/
    - contents.rdf
    - *.css
    - *.xul
    - *.js
- install.js

Cette extension est installé dans Thunderbird grâce au menu extension.

#### *6.1.3.3 Architecture*

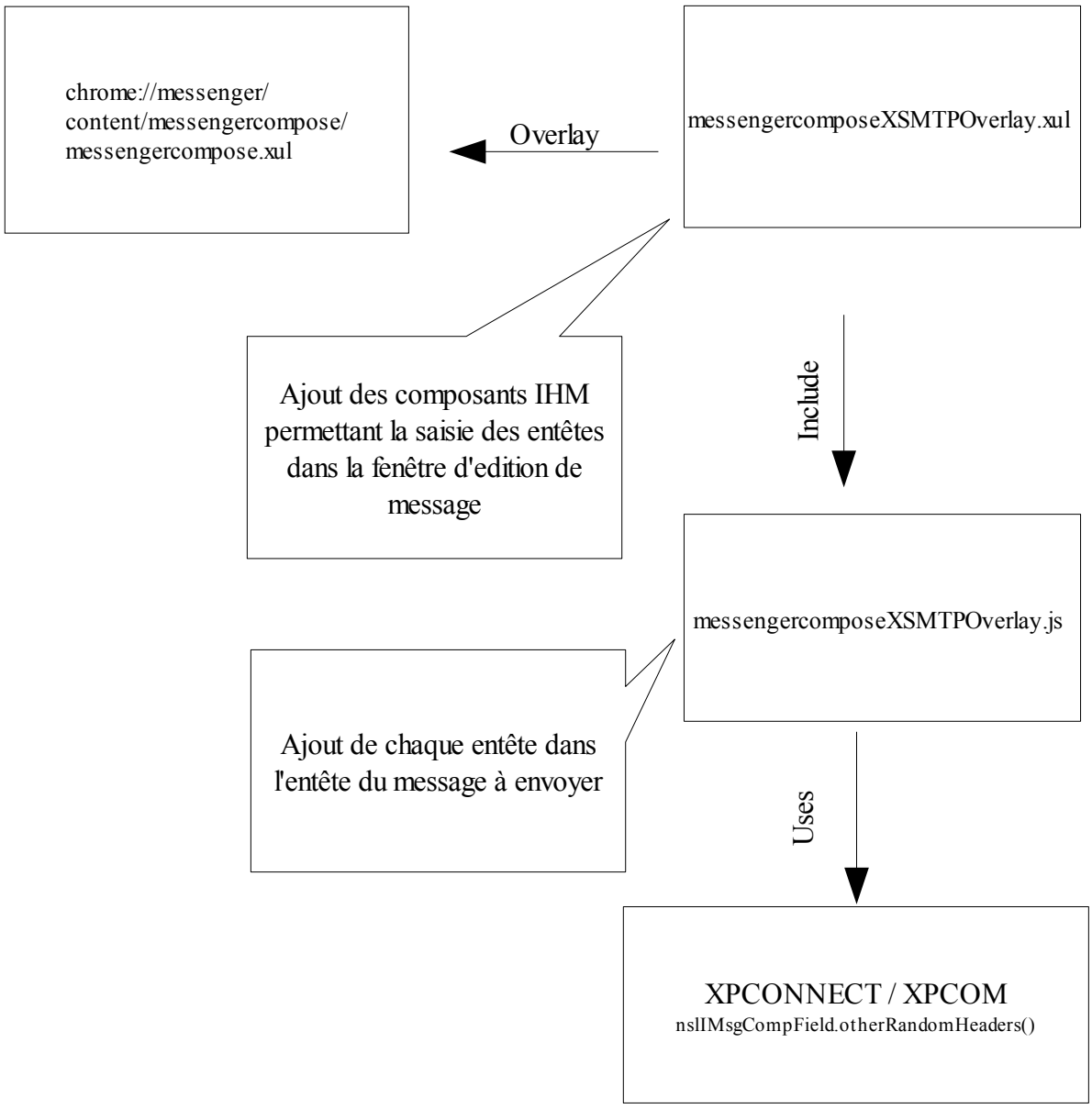
##### 6.1.3.3.1 Persistance des valeurs par défauts

Les valeurs par défaut de certains entêtes obligatoires doivent être paramétrées dans les configuration de l'utilisateur (voir spécification DGA) dans les préférences de Thunderbird.

#### 6.1.3.3.2 Ajout des entêtes dans la fenêtre d'envoi de mails

Fichier overlay à implémenter : messengercomposeOverlay.xul

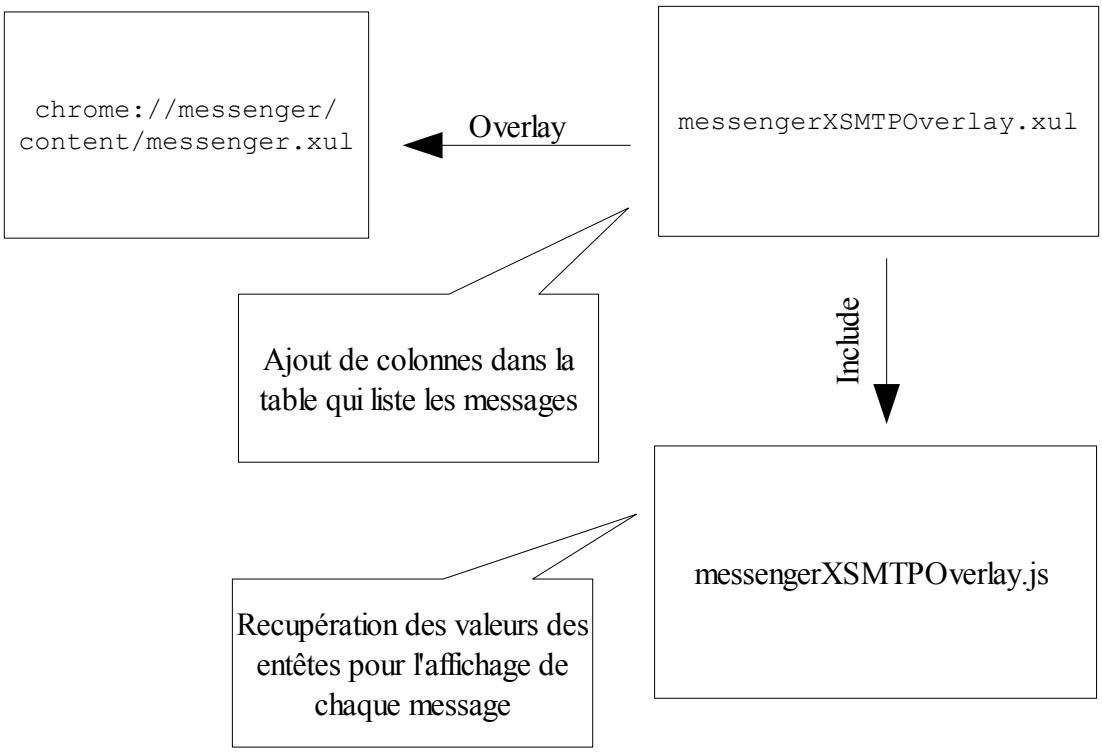
Fichier javascript contenant le code métier : messengerComposeOverlay.js



#### 6.1.3.3.3 Affichage des entêtes dans la fenêtre qui liste les messages

Fichier XUL Overlay à implémenter : messengerOverlay.xul (surcharge messenger.xul) à

Fichier Javascript à implémenter : messengerOverlay.js




### *6.1.3.4 Interface graphique*

#### 6.1.3.4.1 Saisie des informations relatives aux entêtes XSMTP

##### 6.1.3.4.1.1 Localisation

Tous les labels utilisés dans les différentes vues sont traduits en français et en anglais en utilisant le processus de Mozilla (http://www.mozilla.org/projects/l10n/)

##### 6.1.3.4.1.2 Sources de données

Les données nécessaires pour remplir les champs de saisie comme les listes déroulantes par exemple seront stockées dans des fichiers au format RDF.

#### 6.1.3.4.1.3 Intégration dans la fenêtre d'envoi

Lors de l'envoi d'un message, l'utilisateur doit pouvoir choisir de renseigner les entêtes XSMTP.

De ce fait un bouton doit être ajouté à la barre d'outil de la fenêtre d'édition de message qui lancera l'édition des champs XSMTP.

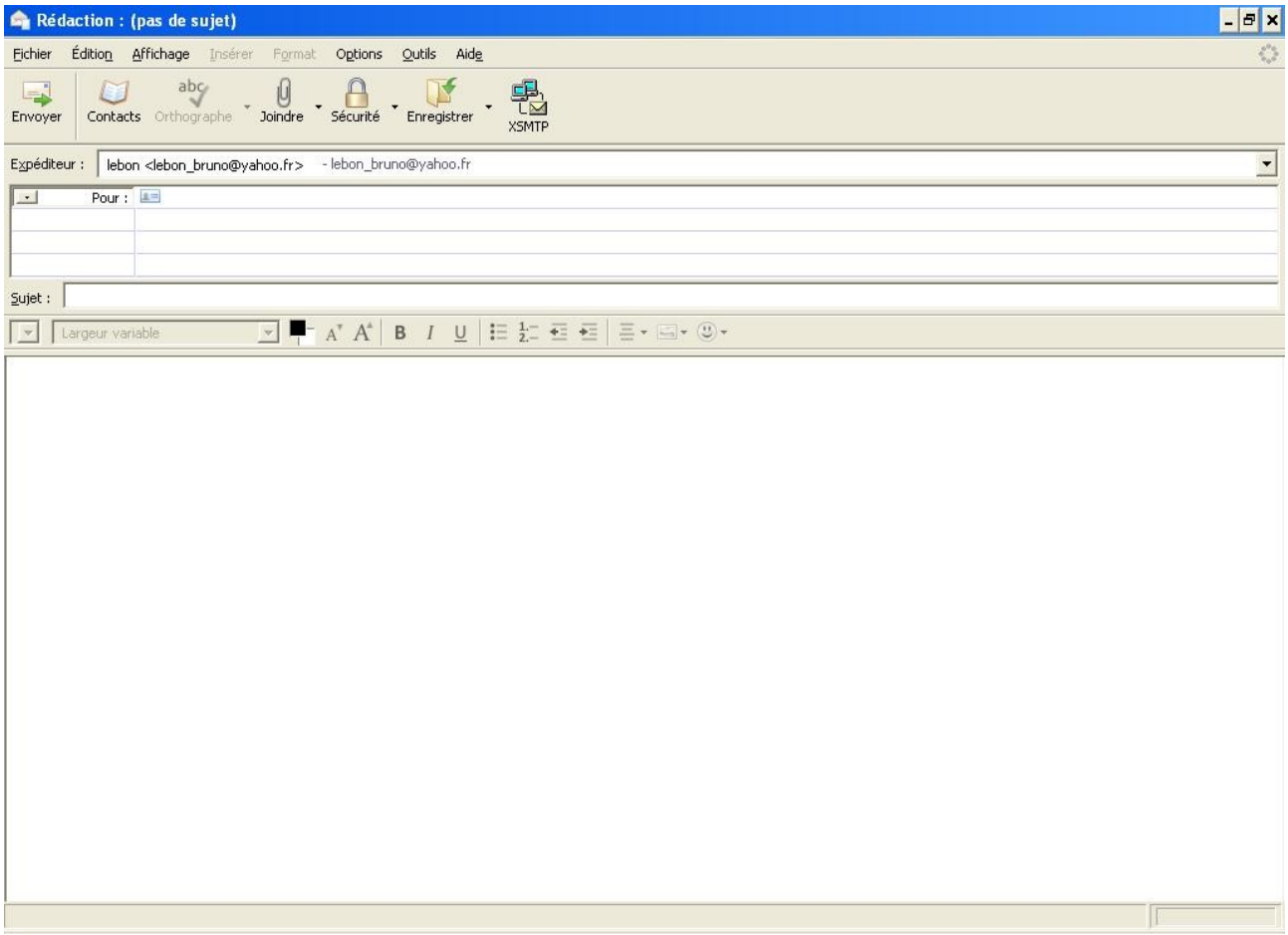
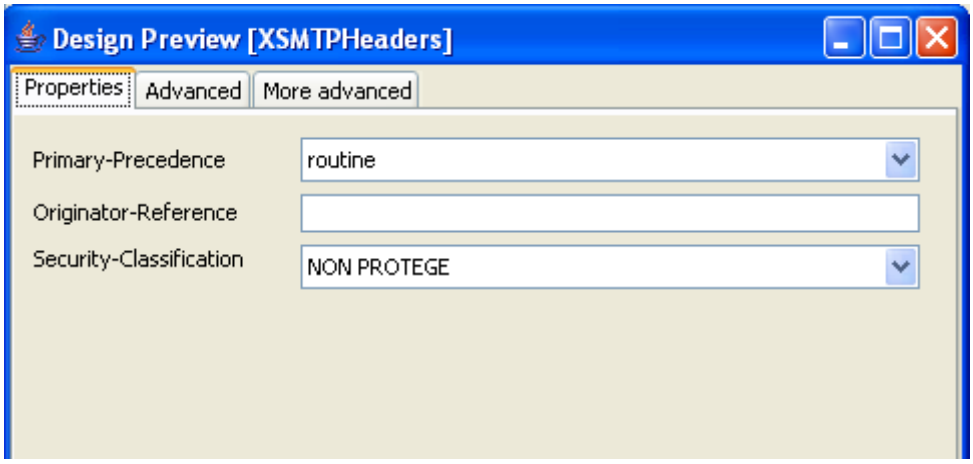



*Illustration 1: Fenêtre d'édition de message, avec ajout du bouton XSMTP*

Précautions :

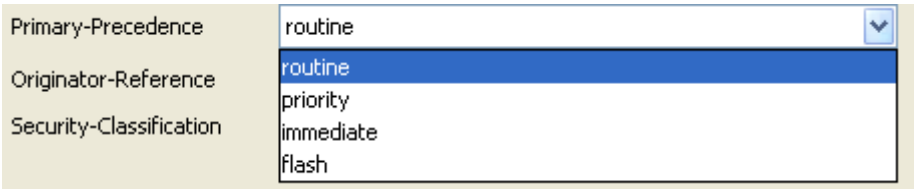
- Les champs doivent être persistant pour pouvoir éditer un message archivé avant envoi.

#### 6.1.3.4.1.4 Partie Obligatoire

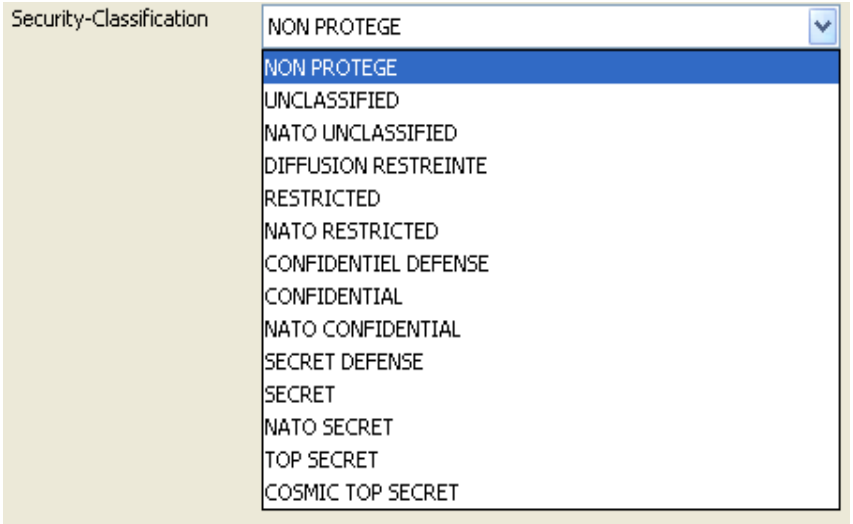



#### 6.1.3.4.1.5 Détails

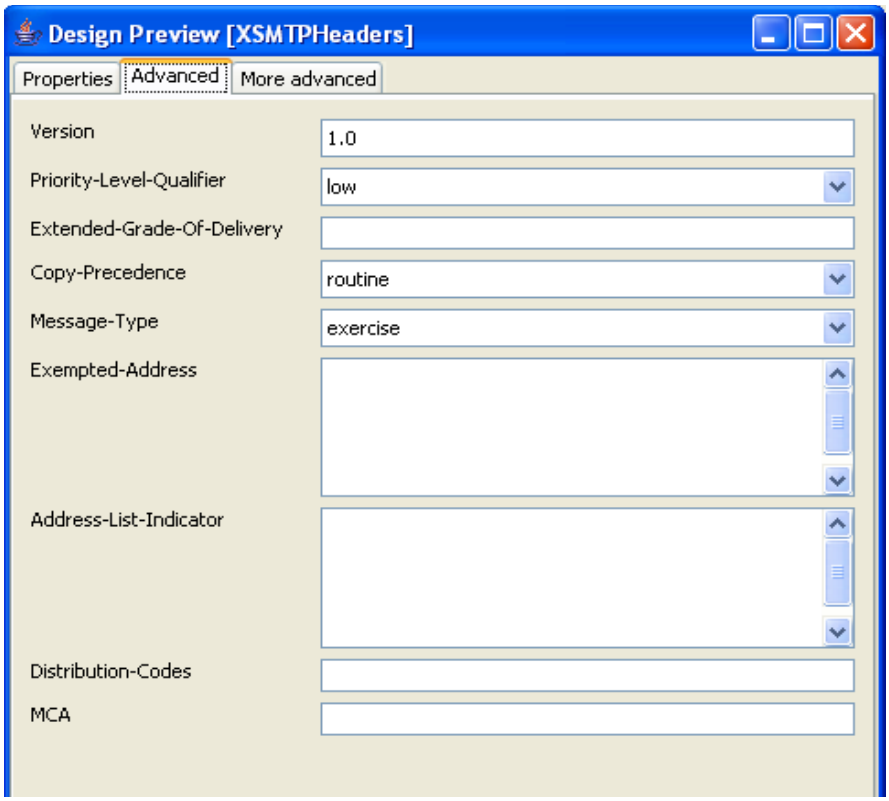
Les valeurs associées à l'entête *Security-Classification* sont stockées en fichier de configuration RDF.




Les valeurs associées à l'entête *Security-Classification* sont stockées en fichier de configuration RDF.

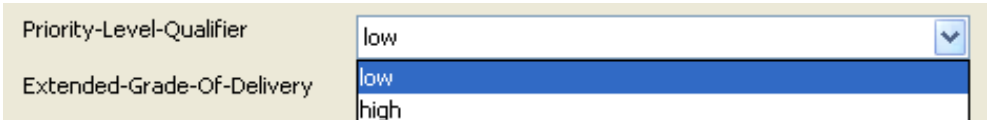


#### 6.1.3.4.1.6 Partie Optionnelle

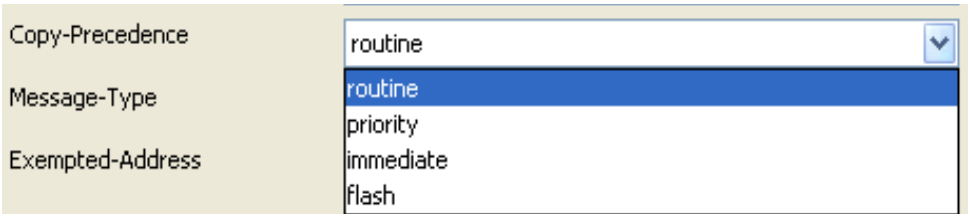



#### 6.1.3.4.1.7 Détails

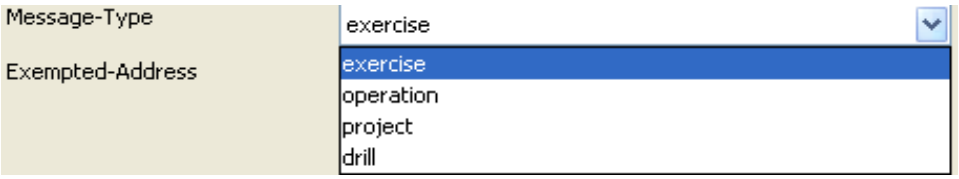
Les valeurs associées à l'entête *Priority-Level-Qualifier* sont stockées en fichier de configuration RDF.



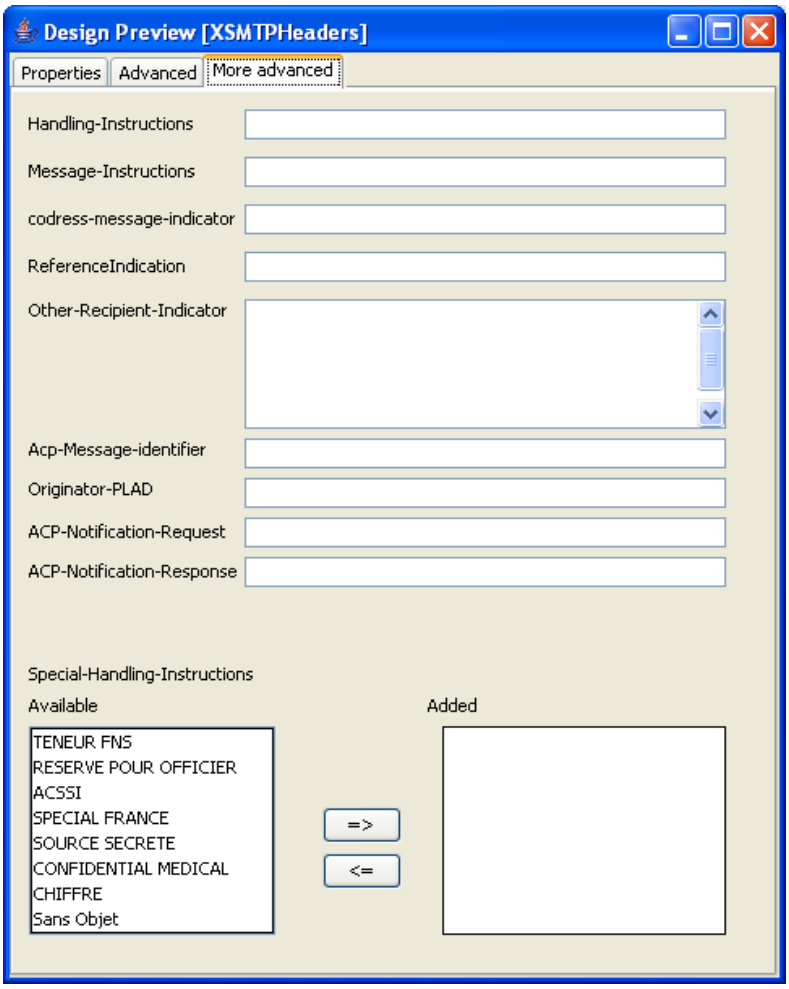

Les valeurs associées à l'entête *Copy-Precedence* sont stockées en fichier de configuration RDF.



Les valeurs associées à l'entête *Exempted-Address* sont stockés en fichier de configuration RDF.




#### 6.1.3.4.1.8 Partie optionnelle suite




Les valeurs associées à l'entête *Special-Handling-Instructions* sont stockées en fichier de configuration RDF.

#### 6.1.3.4.2 Affichage des entêtes dans la vue réception des messages :

Une table liste tous les messages contenus dans un dossier de réception.

Le but est d'ajouter une colonne par entête de la norme XSMTP. Ces colonnes pourront être ajoutées à la vue ou retirer grâce au système de gestion de colonne déjà inclus dans Thunderbird. Certains entête sont obligatoires (Voir Spécifications DGA), de ce fait les colonnes associées ne pourront pas être enlevées de la table.

D'autre part, à la lecture du message, l'utilisateur doit pouvoir accéder aux détails des champs XSMTP en lecture seule (même interface graphique qu'à l'envoi).


Messages
Outils
?
Répondre
Rép. à tous
Transférer
Étiquette
Supprimer
Indésirable
Imprimer
Stop
Sujet ou expéditeur
Sujet
Expéditeur
Date
Priorité
X-P772-Version
X-P772-Priority

## 6.2 Carnet d'adresses

### 6.2.1 Vérification des destinataires avec des annuaires multiples

#### 6.2.1.1 Principe général

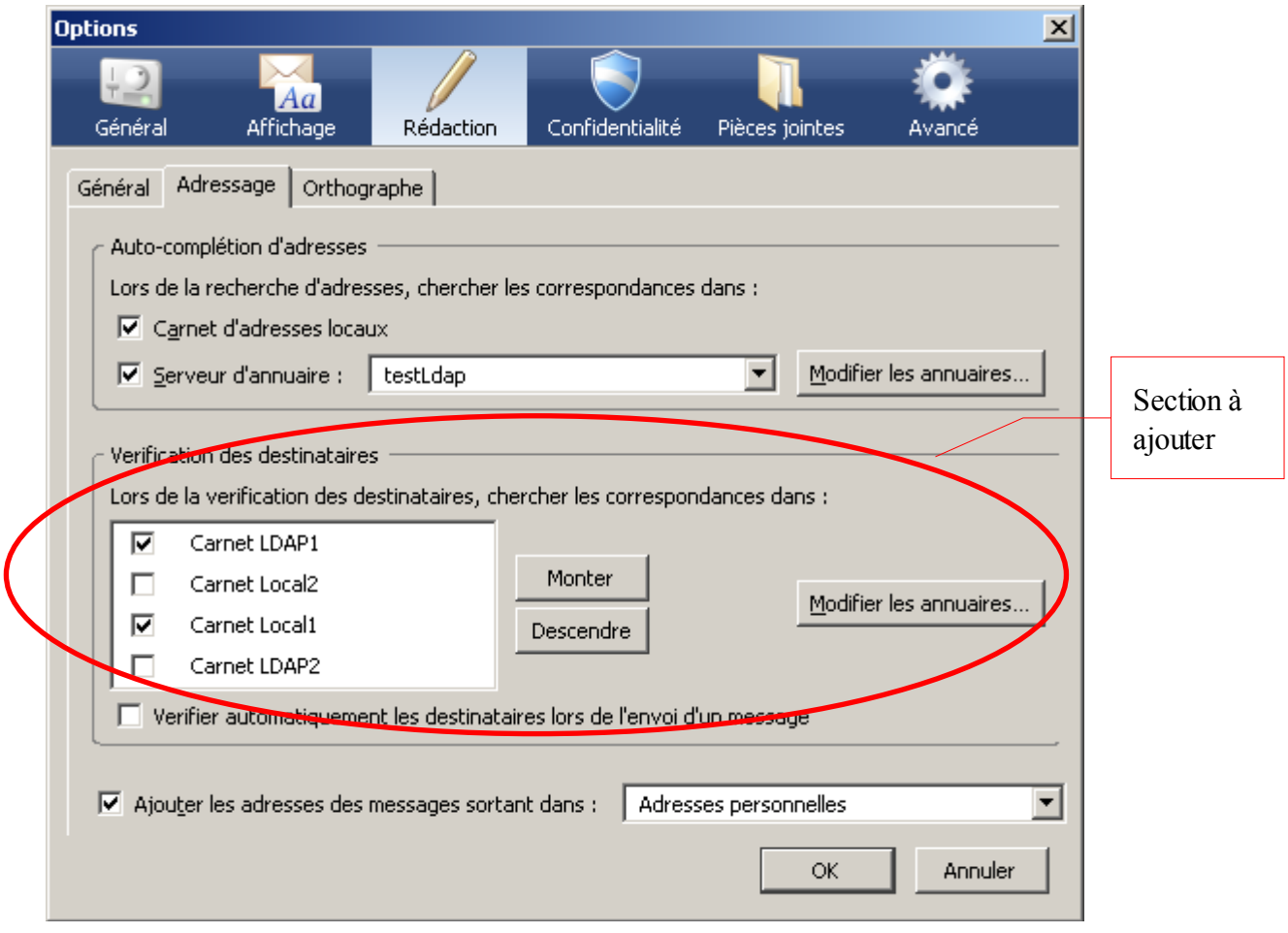
La fonctionnalité permettant de vérifier les destinataires avec des annuaires multiples sera implémentée sous la forme d'une extension Thunderbird dédiée. Le nom de cette extension sera check_recipients et ce préfixe sera repris le plus souvent possible, notamment pour éviter les collisions avec les objets standards : par exemple dans le nommage des propriétés utilisateurs, des variables globales etc ...

#### 6.2.1.2 Définition des annuaires et de leur ordre pour la vérification

Afin de définir les annuaires et leur ordonnancement pour la vérification, il est nécessaire d'ajouter une fonctionnalité de paramétrage dans l'écran dédié aux options utilisateurs.

Cet écran se trouve dans le menu Outils >> Options.

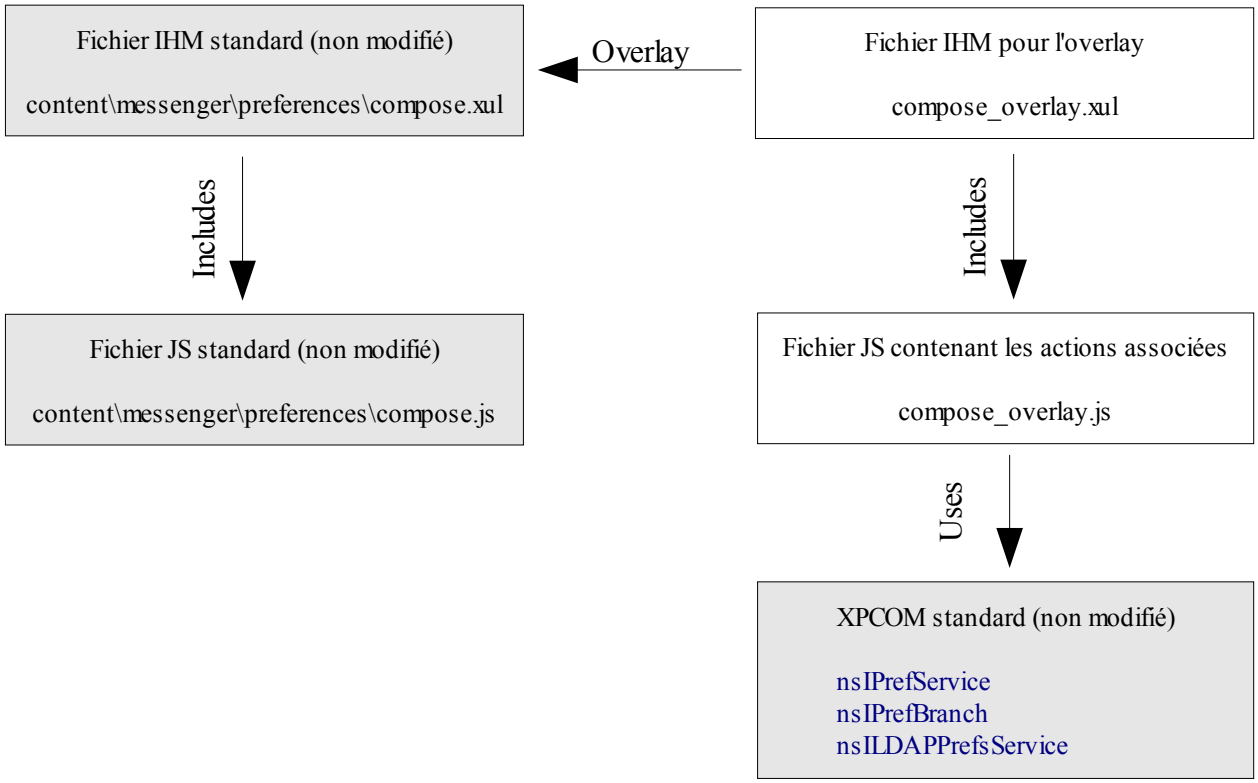
La partie « Rédaction », onglet « Adressage » semble la plus adaptée pour ajouter ce paramétrage.



La définition de cet écran et les actions associées sont définies dans l'archive chrome\messenger.jar, par les fichiers :

- content\messenger\preferences\compose.xul
- content\messenger\preferences\compose.js

La technique de l'overlay sera utilisée pour surcharger le comportement de ces fichiers suivant le diagramme suivant :




L'appel aux objets XPCOM permettra de charger et de sauvegarder ces préférences, en utilisant notamment les interfaces : nsIPrefService et nsIPrefBranch.

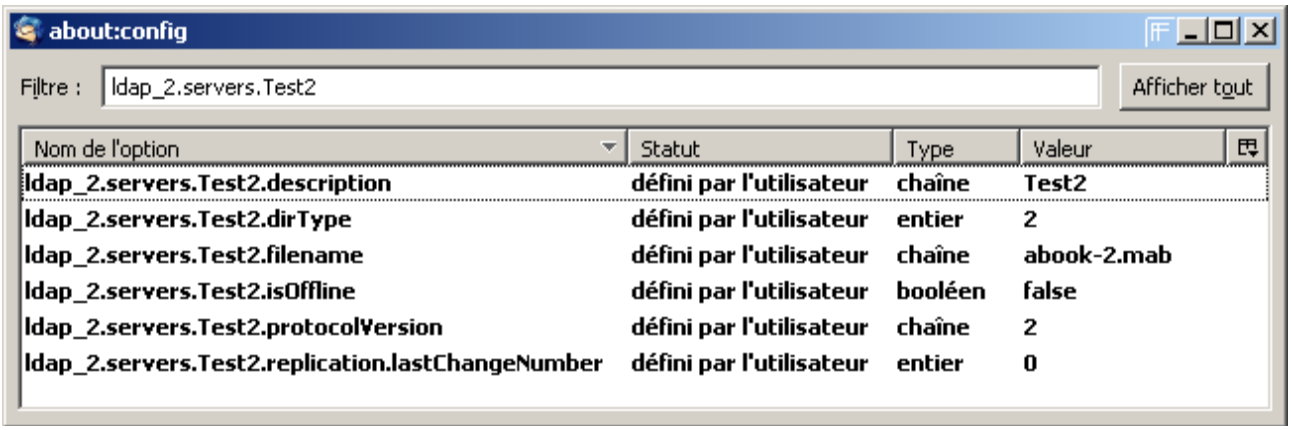
La documentation suivante détaille les mécanismes de chargement et de sauvegarde des préférences: http://www.xulplanet.com/tutorials/xulqa/q_prefs.html

L'interface XPCOM nsILDAPPrefsService sera également utile, car elle permet de récupérer la liste des annuaires LDAP définis par l'utilisateur par la méthode suivante :

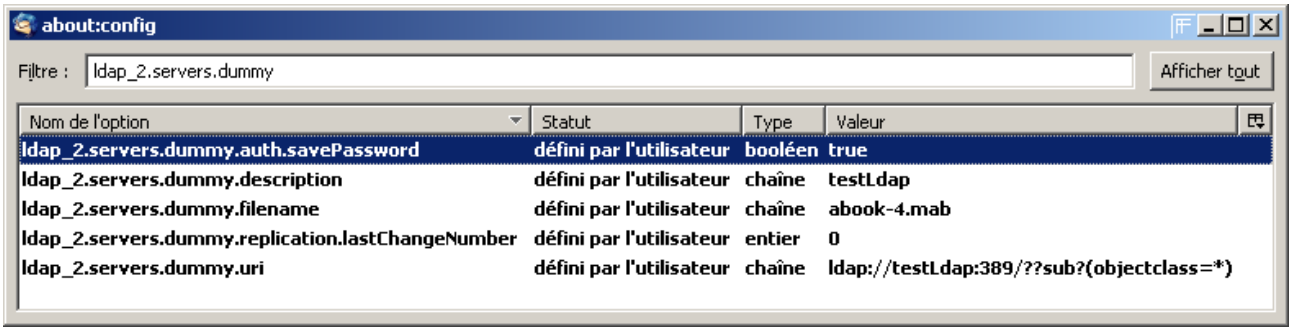
```
void getServerList(nsIPrefBranch prefBranch, out PRUint32 count, out arrayof
char* childArray)
```

Les préférences standards Thunderbird concernant les annuaires sont celles ayant comme préfixe : ldap_2.servers. Ceci est utile afin de faire le lien avec les préférences de cette extension.

Par exemple, les préférences pour un annuaire Local nommé « Test2 » sont :




Les préférences pour un annuaire LDAP nommé «dummy » sont :




Les préférences utilisées pour stocker la liste ordonnée des annuaires seront les suivantes :

| Nom de la préférence | Type | Description |
|---|---|---|
| check_recipients.directory.X.position | integer | Permet de connaître la position de cet annuaire dans la liste |
| check_recipients.directory.X.enable | boolean | Permet de savoir si cet annuaire est utilisé pour la vérification |
| check_recipients.on_mail_send.enable | boolean | Activation / Désactivation de la vérification automatique des destinataires lors de l'envoi d'un message |

où X est la clé correspondant à l'annuaire définit par l'utilisateur, ce qui permettra de faire le lien avec les préférences standards ldap_2.servers et ainsi de récupérer les informations utiles.

### 6.2.1.3 Suppression d'un annuaire

Il est nécessaire de gérer la suppression d'un annuaire par l'utilisateur. En effet, cette suppression doit engendrer la mise à jour des préférences liées à la vérification des destinataires.

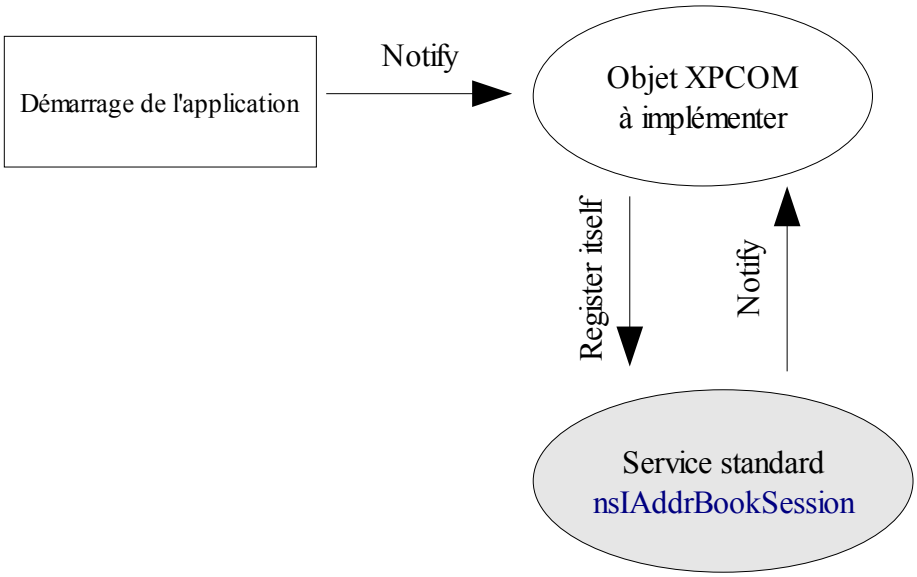
Cette gestion doit se faire au niveau des objets XPCOM.

Le service nsIAddrBookSession permet d'enregister un listener qui sera notifié lors d'événements concernant les carnets d'adresses par la méthode :

```
void addAddressBookListener(nsIAbListener listener, abListenerNotifyFlagValue
notifyFlags)
```

Parmis les événements qui génèrent une notification, on trouve notamment celui indiquant la suppression du carnet d'adresse.

Il est donc nécessaire d'implémenter un objet XPCOM qui lors du lancement de l'application enregistrera ce listener sur les carnets d'adresses.




Ce composant XPCOM à implémenter sera donc lui même un listener sur l'événement de démarrage de l'application. Ce composant doit donc s'enregistrer en tant que listener de démarrage lors du mécanisme « XPCOM Registration ».

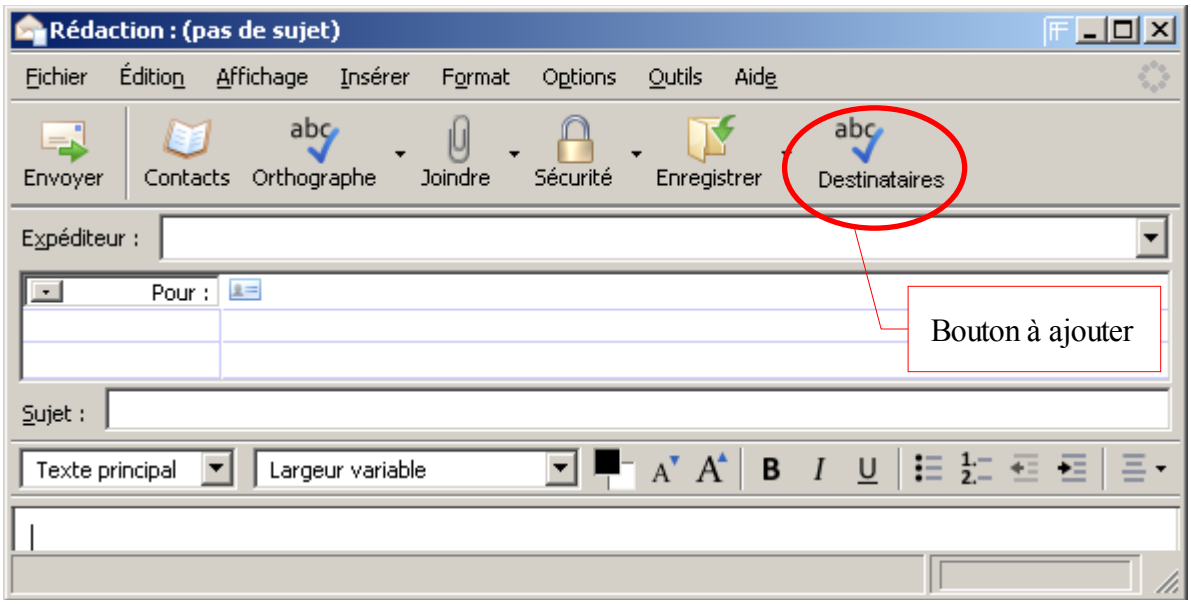
Un exemple complet de code pour implémenter ce mécanisme est disponible ici :

http://www.mozilla.org/projects/xpcom/book/cxc/html/weblock.html

### 6.2.1.4 Vérification des destinataires

L'écran d'envoi d'un message sera complété par une fonctionnalité permettant de vérifier les destinataires. Cette vérification pourra être utilisée avec des annuaires (locaux ou LDAP) multiples.

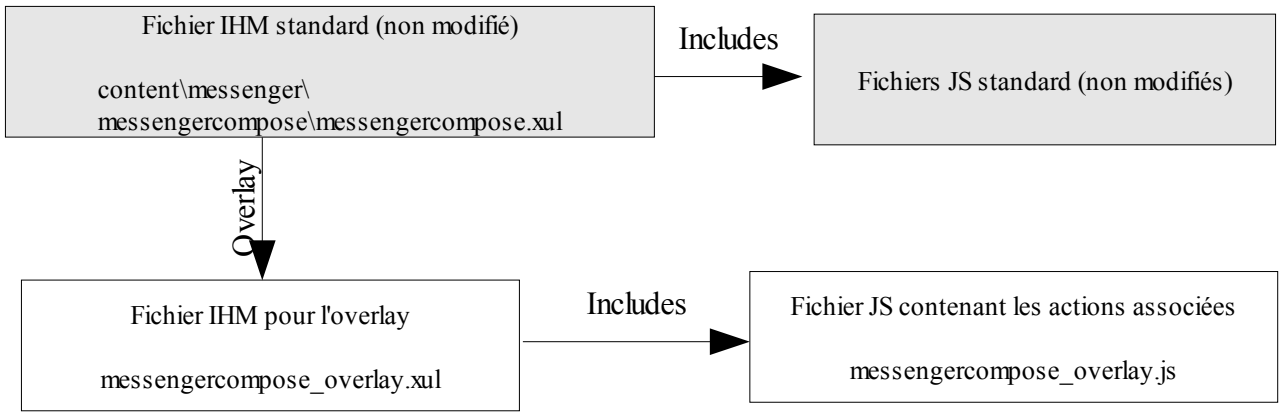
Cette fonctionnalité sera accessible par un bouton d'action, ainsi que par le raccourci clavier « Ctrl + K ». De plus, selon les préférences définies par l'utilisateur, cette vérification pourra être automatiquement exécutée avant l'envoi d'un message.



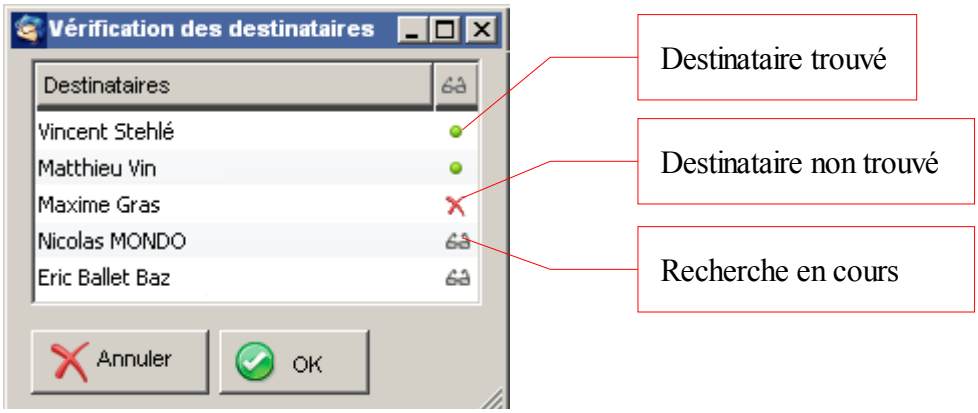

La définition de cet écran et les actions associées sont définies dans l'archive chrome\messenger.jar, par les fichiers :

- content\messenger\messengercompose\messengercompose.xul
- de nombreux fichiers JS ...

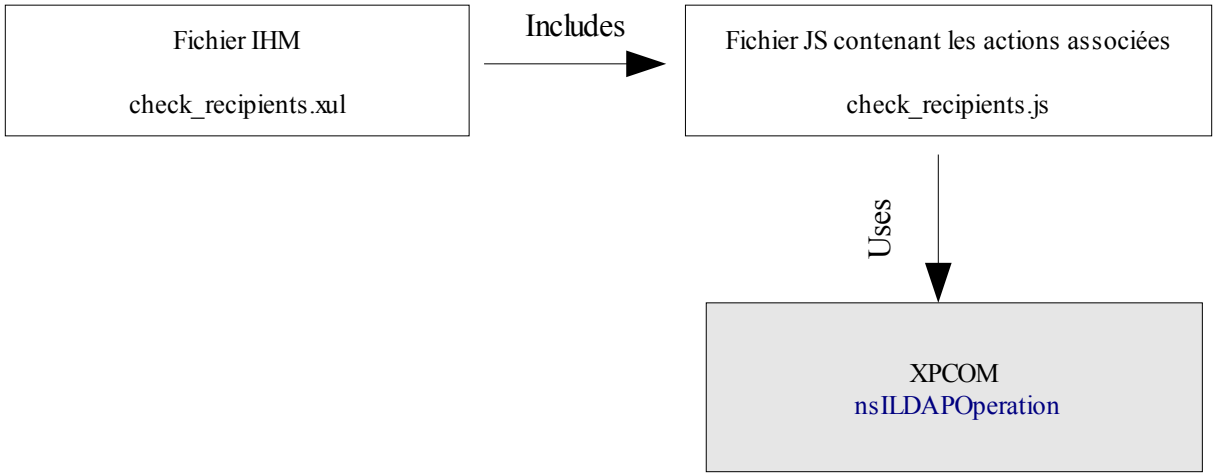
La technique de l'overlay sera utilisée pour surcharger le comportement de ces fichiers suivant le diagramme suivant :



Le lancement de la vérification provoquera l'affichage d'une fenêtre du type :




Cette nouvelle fenêtre devra être implémentée par un fichier XUL et un fichier JavaScript spécifique.



### 6.2.2 Auto complétion de l'adresse des contacts lors de l'écriture d'un mail

Cette fonctionnalité est disponible est tant que patch dans Bugzilla

https://bugzilla.mozilla.org/show_bug.cgi?id=125188

Il est prévu de l'intégrer au source de Thunderbird, puis de recompiler cette nouvelle version.

## *6.3 Envoi de Mail*

### 6.3.1 Gestion des formats distants à partir de l'annuaire

#### *6.3.1.1 Principe général*

La fonctionnalité permettant d'utiliser le format préféré d'un destinataire LDAP sera implémentée sous la forme d'une extension Thunderbird dédiée. Le nom de cette extension sera send_format_ldap et ce préfixe sera repris le plus souvent possible, notamment pour éviter les collisions avec les objets standards : par exemple dans le nommage des propriétés utilisateurs, des variables globales etc ...

Par défaut, lors de l'envoi d'un mail, la recherche du format accepté par un destinataire est effectuée uniquement sur les carnets d'adresses locaux. La valeur définie au niveau du LDAP n'est pas prise en compte.

Une anomalie a été ouverte chez Mozilla à ce sujet :

https://bugzilla.mozilla.org/show_bug.cgi?id=375833

Par défaut, l'attribut LDAP utilisé par Thunderbird pour déterminer si un contact supporte le format HTML est un attribut booléen nommé soit « mozillaUseHtmlMail », soit « xmozillausehtmlmail ».

Cette valeur par défaut est paramétrable via la propriété standard :

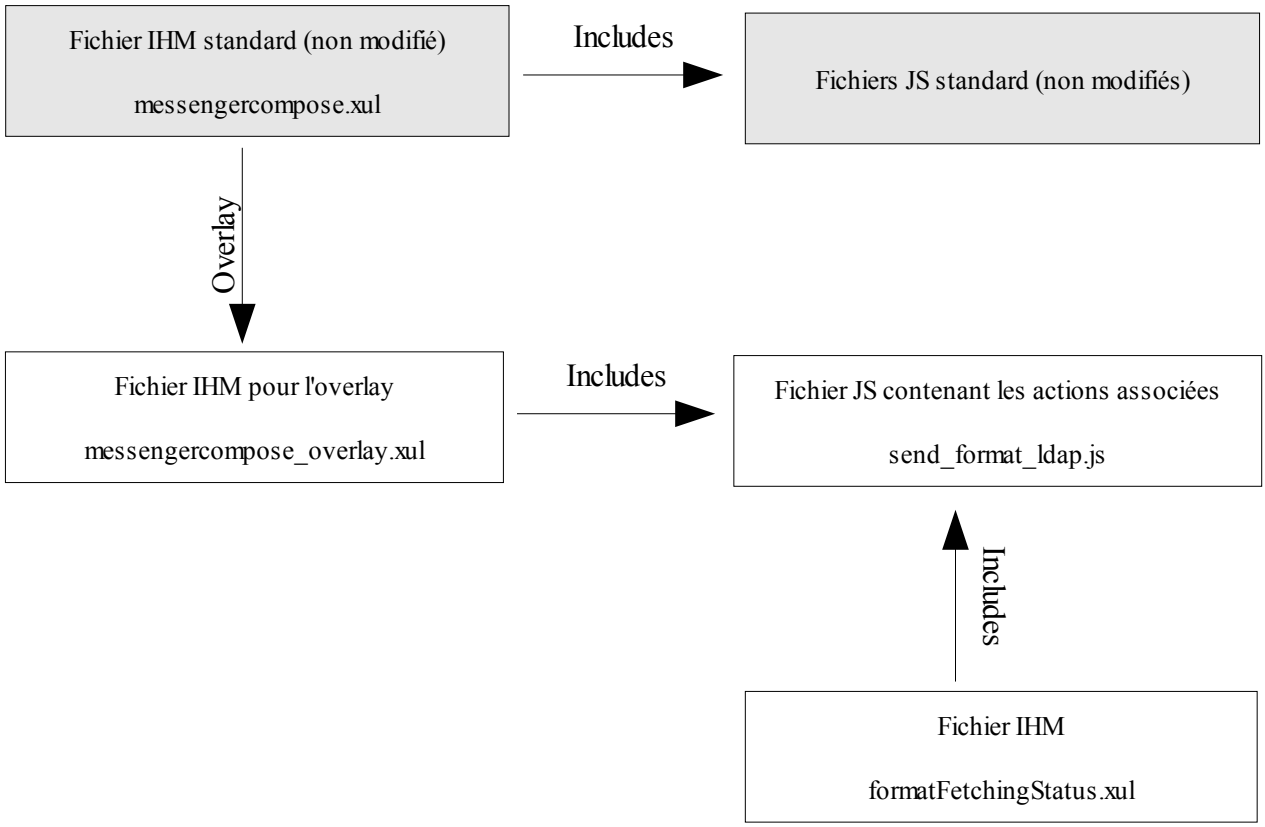
ldap_2.servers.default.attrmap.PreferMailFormat

Cette propriété peut être multivaluée.

Le paramétrage de cette valeur sera disponible dans les options de cette extension. La valeur de l'attribut LDAP doit être « TRUE » ou « FALSE » (Attention à la casse).

### 6.3.1.2 Implémentation

La technique de l'overlay sera utilisée pour surcharger le comportement de ces fichiers suivant le diagramme suivant :




Le fichier « formatFetchingStatus.xul » implémentera une fenêtre montrant la progression de la recherche.

Un moyen simple d'ajouter la recherche en utilisant le carnet d'adresse LDAP est de modifier la librairie MsgComposeCommands.js au niveau de la fonction :

```
function DetermineHTMLAction(convertible)
```

En standard, cette fonction fait appel au service nsIMsgCompose pour déterminer le format supporté des destinataires via la méthode CheckAndPopulateRecipients. Cette fonction effectue la recherche uniquement sur les carnets d'adresses locaux, et retourne la liste des destinataires ne supportant pas le HTML.

Il faut compléter cette fonction pour traiter la liste des destinaires retournés. Pour chaque destinataire, il faut effectuer une recherche LDAP et le retirer de la liste s'il est défini comme supportant le HTML.

Pour l'interrogation d'un annuaire LDAP, l'API XPCOM fournit une interface nsILDAPOperation et en particulier la méthode :

```
void searchExt ( AUTF8String baseDn , PRInt32 scope , AUTF8String filter ,
PRUint32 attrCount , arrayof char* attributes , PRIntervalTime timeOut ,
PRInt32 sizeLimit )
```

La propriété définissant l'attribut LDAP booléen à interroger est une propriété Thunderbird standard: ldap_2.servers.default.attrmap.PreferMailFormat

Cet exemple simple ne traite pas de la gestion des données retournées. Un exemple complet de récupération de données depuis un annuaire LDAP est disponible dans la librairie certFetchingStatus.js. L'interrogation du LDAP est contenue dans la méthode kickOffSearch() qui est un bon point d'entrée pour comprendre ce code.

## 6.3.2 Limite de la taille des messages en fonction de la priorité

### *6.3.2.1 Principe général*

Dans le cas d'un message envoyé avec les extensions XSMTP, un contrôle de la taille d'un message peut être effectué en fonction de la priorité du message. L'activation ou non de ce contrôle doit pouvoir être configuré par une entrée dans les configuration de l'utilisateur.

### *6.3.2.2 Configuration*

La priorité est représenté par l'entête X-P772-Copy-Precedence. Des limites en kilo octets sont définis pour chaque niveau de priorité par défaut dans le fichier de configuration de l'utilisateur.

Les quatre niveaux de priorités sont : Flash, Immediat, Urgent, Routine.

Les clefs associées en configuration de Thunderbird sont :

- xsmtp.size.flash
- xsmtp.size.immediat
- xsmtp.size.urgent
- xsmtp.size.routine

Par défaut :

- xsmtp.size.flash = 10
- xsmtp.size.immediat = 50
- xsmtp.size.urgent = 1000
- xsmtp.size.routine = 10000

L'activation du contrôle :

- xsmtp.size.check.enable boolean

Service à utilisé pour lire et modifier les préférences :

```
var prefs = Components.classes["@mozilla.org/preferences-service;1"].
                getService(Components.interfaces.nsIPrefBranch);
```

Ajouter des préférences : fichier à inclure (user.js)

```
user_pref("mypreference", value);
```

#### *6.3.2.3 Contrôle de la taille du message lors de son envoi*

Thunderbird propose une gestion d'écouteurs que l'on peut enregistrer au près de lui pour effectuer des traitements lors d'évènements.

La méthode suivante permet d'associer une fonction écouteur qui sera lancée lors de l'envoi du mail.

```
addEventListener('compose-send-message', listenerFunction, true);
```

Ensuite il faut récupérer le message courant de type **nsIMsgCompose**, puis grâce à lui récupérer les champs du message avec l'attribut **compFields** de type **nsIMsgCompFields.**

Le type **nsIMsgCompFields** contient l'attribut *body* et la méthode **SplitRecipients()** permettant de lister les pièces jointes.

Faire la somme des tailles du *body* et des pièces jointes et le comparer à la taille limite en fonction de la priorité courante. Si la taille est dépassée, il faut le signaler à l'utilisateur avec une fenêtre de type dialogue et bloquer le processus d'envoi.

## 6.4 API

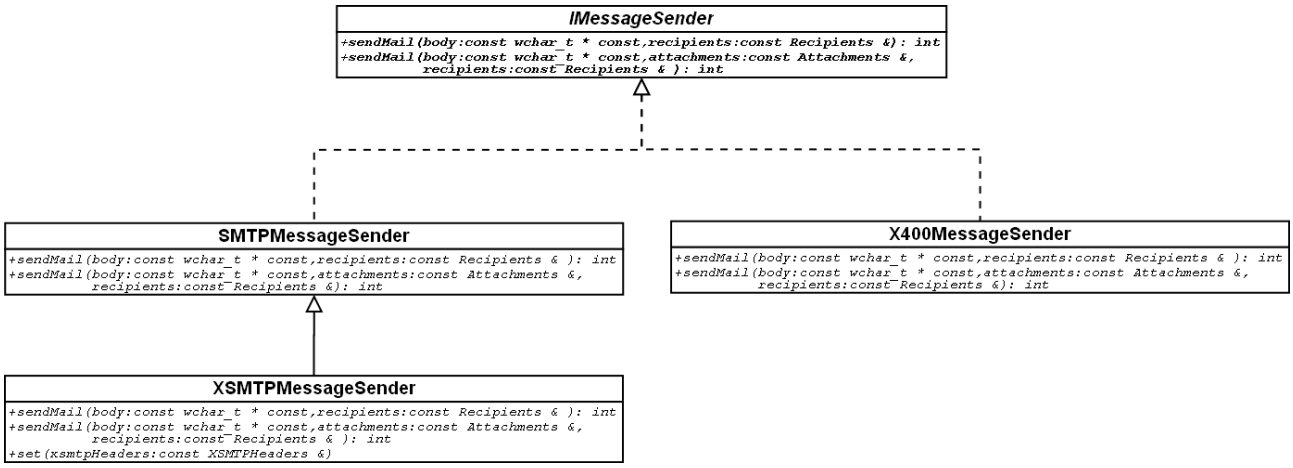
### 6.4.1 API Milimail

L'API (Application Program Interface) MiliMail permet d'offrir toutes les possibilités d'envoi de messages pouvant être intégrés dans une application externe. L'API est écrit en C++ utilisant des types standard pour permettre une intégration optimal.

#### 6.4.1.1 Accès au fonctionnalités d'envoi

Chaque protocole est représenté par une implémentation de l'interface `IMessageSender`. Le protocole SMTP est représenté par la classe `SMTPMessageSender`, sa variante munie d'une méthode pour renseigner les entêtes XSMTP est représentée par `XSMTPMessageSender`.

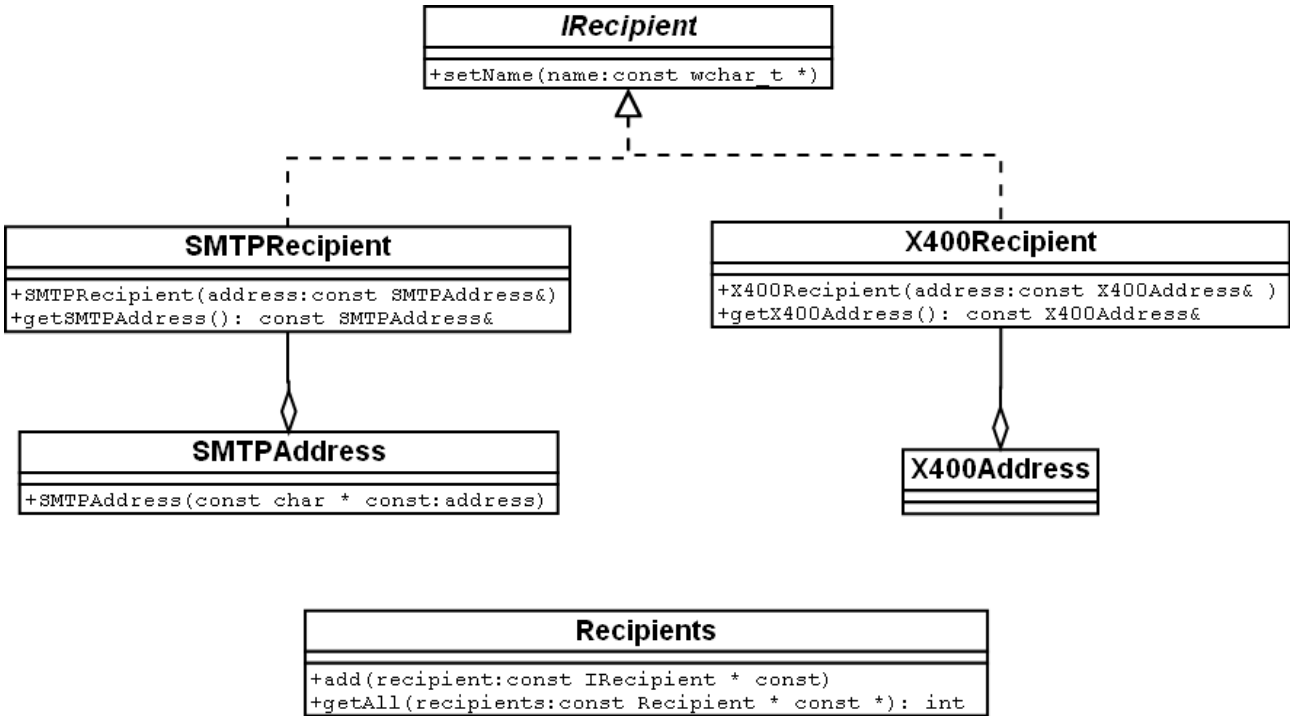
L'envoi de message protocole X4OO est lui représenté par la classe `X400MessageSender`.




#### 6.4.1.2 Classes en rapport avec les Destinataires

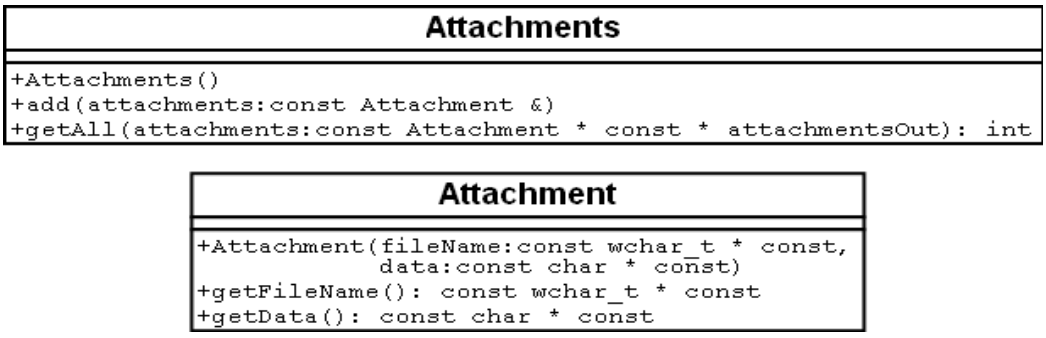
Chaque implémentation de l'interface `IMessageSender` utilise une implémentation de l'interface `IRecipient`. `SMTPRecipient` représente un destinataire complet pour le protocole SMTP muni d'une adresse instance de la classe SMTPAddress. `X400Recipient` représente un destinataire complet pour le protocole X400.

Les méthodes associées liées à la classe `X400Recipient` seront détaillées ultérieurement.




### 6.4.1.3 Classes pour les fichiers attachés

Chaque implémentation de l'interface `IMessageSender` utilise la classe `Attachements` qui représente les documents attachés au message. Cette classe contient plusieurs instances de la classe `Attachement` représentant un document attaché.



### 6.4.1.4 Classe pour les entêtes XSMTP

Une instance de la classe `XSMTPMessageSender` peut être renseigné des entêtes XSMTP.

| XSMTPHeaders |
| --- |
| +setVersion(version:const char * const)<br>+setPriorityLevelQualifier(levelQualifier:XSMTPPriorityLevelQualifierEnum)<br>+setExtendedGradeOfDelivery(extendedGradeOfDelivery:const char * const)<br>+setPrimaryPrecedence(primaryPrecedence:XSMTPPrimaryPrecedenceEnum)<br>+setCopyPrecedence(copyPrecedence:XSMTPCopyPrecedenceEnum)<br>+setMessageType(messageTypeLabel:XSMTPMessageTypeEnum,<br>copyPrecedenceValue:const char * const=0)<br>+setExemptedAddress(exemptedAddress:const SMTPRecipient * const,<br>addressesSizeLength:const int)<br>+setDistibutionCodes(distributionCodes:const char * const * ,<br>distributionCodesNumber:const int)<br>+setMCA(mca:const char * const * ,mcaNumber:const int)<br>+setHandlingInstructions(handlingInstructions:const char * const * ,<br>handlingInstructionsNumber:const int)<br>+setOtherRecipientIndicator(otherRecipientIndicator:const char * const * ,<br>otherRecipientIndicatorNumber:const int)<br>+setAdressListIndicator(adressListIndicator:const char * const * ,<br>adressListIndicatorNumber:const int)<br>+setOriginatorPLAD(originatorPLAD:const char * const)<br>+setAcpNotificationRequest(acpNotificationRequest:const char * const)<br>+setAcpNotificationResponse(acpNotificationResponse:const char * const)<br>+setSecurityClassification(securityClassification:XSMTPSecurityClassificationEnum)<br>+setSpecialHandlingInstructions(specialHandlingInstructions:XSMTPSpecialHandlingInstructionsEnum * const,<br>instructionsNumber:const int) |

### 6.4.1.5 Énumération utiles pour la classe XSMTPHeaders

La classe `XSMTPHeaders` contient des énumérations pour certaines définitions des entêtes.


### 6.4.2 Lancement Thunderbird conditionnel

Il est possible d'ouvrir la fenêtre de composition d'un message avec des valeurs pré-remplies pour les champs principaux depuis la ligne de commande :

```
thunderbird -compose
"mailto:somebody@somewhere?cc=address@provider&subject=hi&body=something"
```

Ceci est détaillé ici :

http://kb.mozillazine.org/Command_line_arguments_%28Thunderbird%29

## 6.5 Notifications

Deux type de notification doivent être géré par MiliMail :

- MDN Message Disposition Notification
- DSN Delivery Status Notification.

### 6.5.1 MDN

#### 6.5.1.1 Principe général

Le MDN ou (Message Disposition Notification) est un processus qui permet à l'émetteur d'un message de demander au destinataire un accusé. La spécification de la MDN est la RFC 3798.

L'accusé renvoyé par le destinataire à l'émetteur peut être de plusieurs formes :

- un accusé de Lecture, le destinataire accuse la lecture du message en cliquant sur une fenêtre de type *Dialog*. Par contre, il n'y aucune garantie que le destinataire a effectivement lu ou compris le contenu du message.
- un accusé de Suppression, le destinataire supprime le message. Le destinataire peut très bien ne pas avoir lu le message.

Un MDN est demandé grâce à l'inclusion d'un entête *Disposition-Notification-To* dans le message (RFC 3798 2.). La syntaxe de l'entête est la suivante :

```
mdn-request-header = "Disposition-Notification-To" ":"  mailbox *("," mailbox)
```

Explications : Le mot clef de demande de MDN est suivi d'une ou plusieurs adresses mail correspondantes au destinataires du message.

Le MDN renvoyé par le destinataire à l'émetteur originale est un message MIME suivant le convention suivante. Pour plus d'informations, se référer à la (RFC 3798 3.).

```
disposition-notification-content = [ reporting-ua-field CRLF ]
    [ mdn-gateway-field CRLF ]
    [ original-recipient-field CRLF ]
    final-recipient-field CRLF
    [ original-message-id-field CRLF ]
    disposition-field CRLF
    *( failure-field CRLF )
    *( error-field CRLF )
```

```
    *( warning-field CRLF )
    *( extension-field CRLF )

disposition-field =
          "Disposition" ":" disposition-mode ";"
          disposition-type
          [ "/" disposition-modifier
          *( "," disposition-modifier ) ]

   disposition-mode = action-mode "/" sending-mode

   action-mode = "manual-action" / "automatic-action"

   sending-mode = "MDN-sent-manually" / "MDN-sent-automatically"

   disposition-type = "displayed"
           / "deleted"

   disposition-modifier = "error"
          / disposition-modifier-extension

   disposition-modifier-extension = atom
```

Les points importants sont contenus dans l'entête :

- *disposition-type,* qui peut valoir *displayed* (si le message a été affiché) ou *deleted* (si le message a été supprimé).
- s*ending-mod*e qui peut valoir *MDN-sent-manually* (dans le cas ou le MDN est envoyé manuellement par l'utilisateur) ou *MDN-sent-automatically* (dans le cas ou le MDN est envoyé automatiquement par l'utilisateur, préférence par défaut).

#### *6.5.1.2 Existant Thunderbird*

Thunderbird implémente une partie de la spécification MDN.

En émission, le client propose trois niveaux de configuration :

#### 6.5.1.2.1 La configuration générale du logiciel

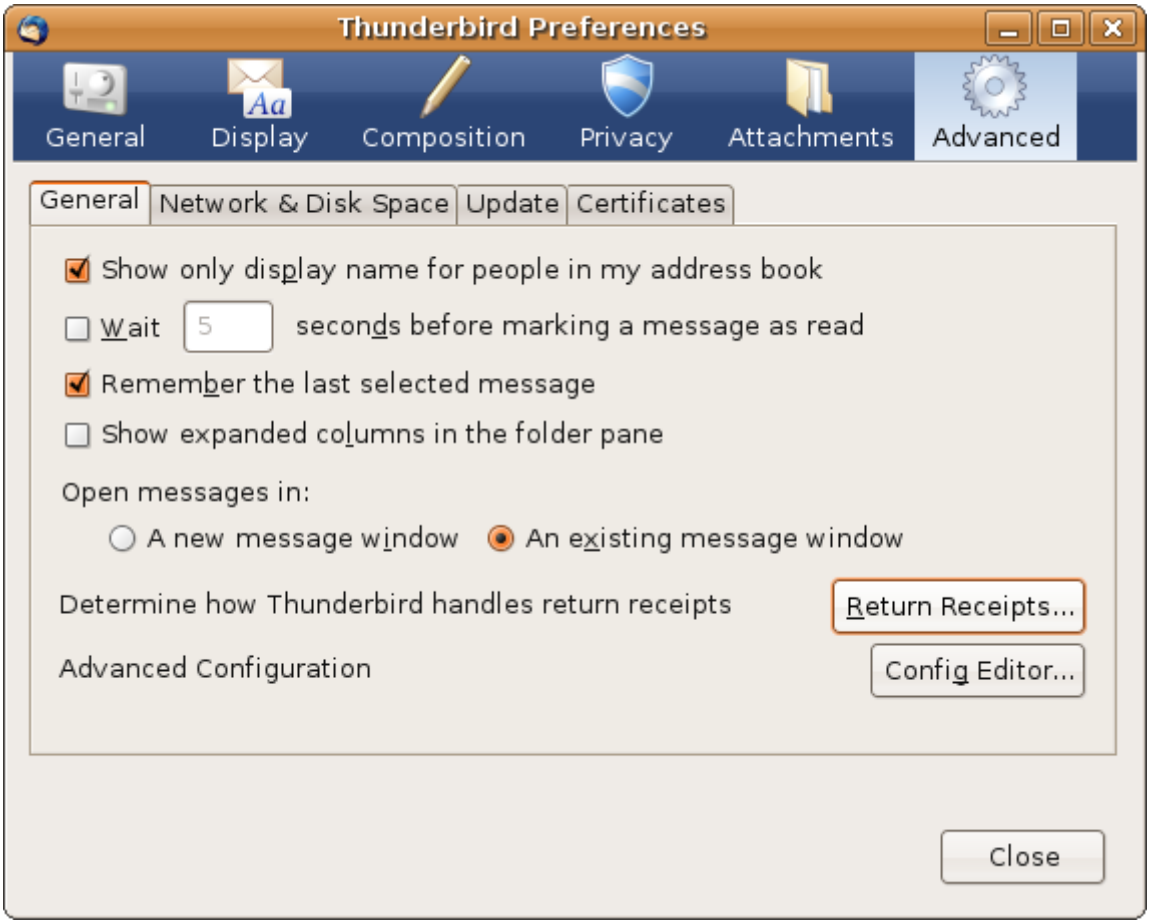



*Illustration 2: Edit -> Preferences -> Advanced (Préférences générales)*

#### 6.5.1.2.2 La configuration des MDN au niveau des comptes de messagerie

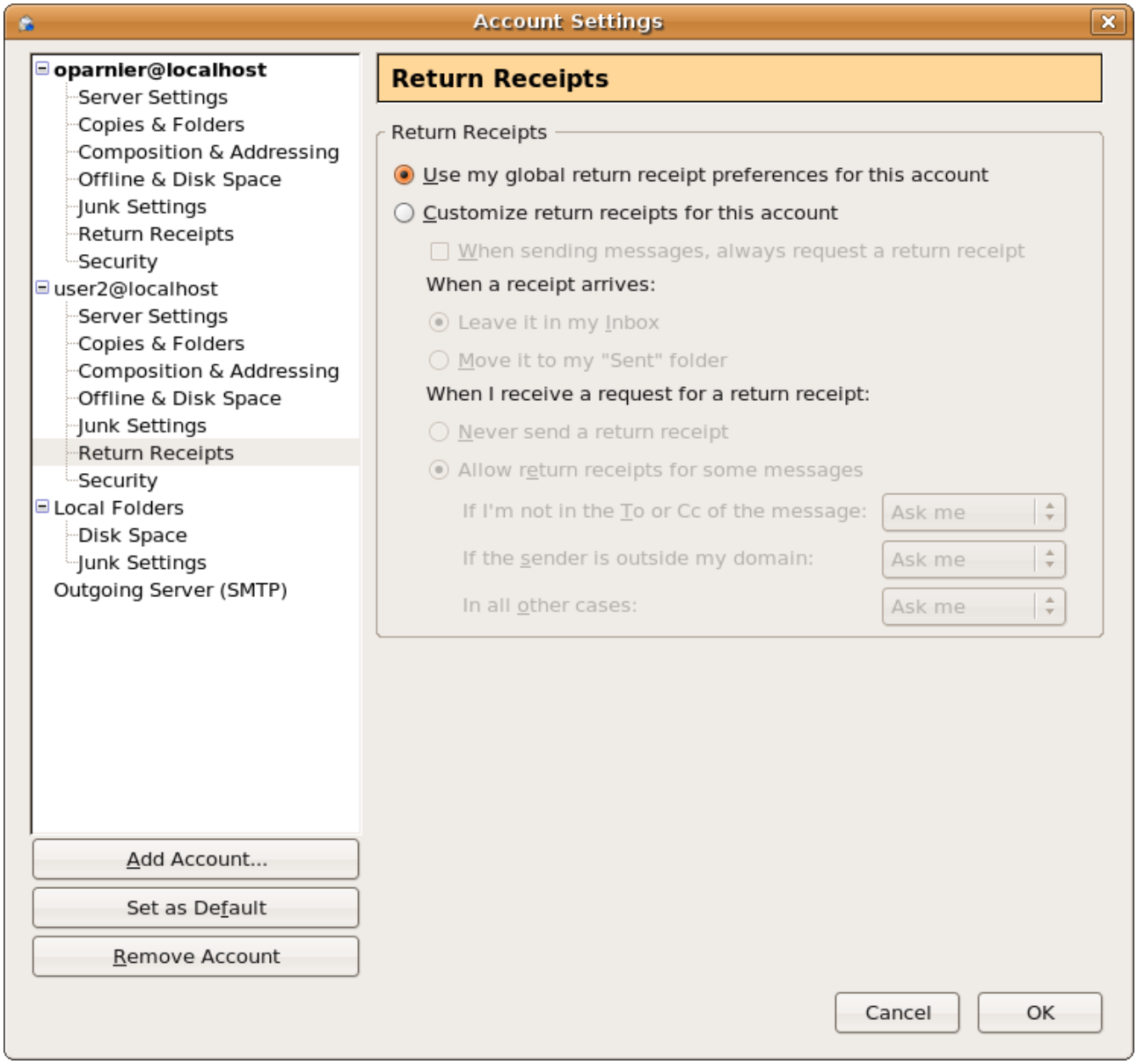



*Illustration 3: Paramétrage MDN par compte*

#### 6.5.1.2.3 L'activation ou non de la demande de MDN au niveau de l'édition de la demande

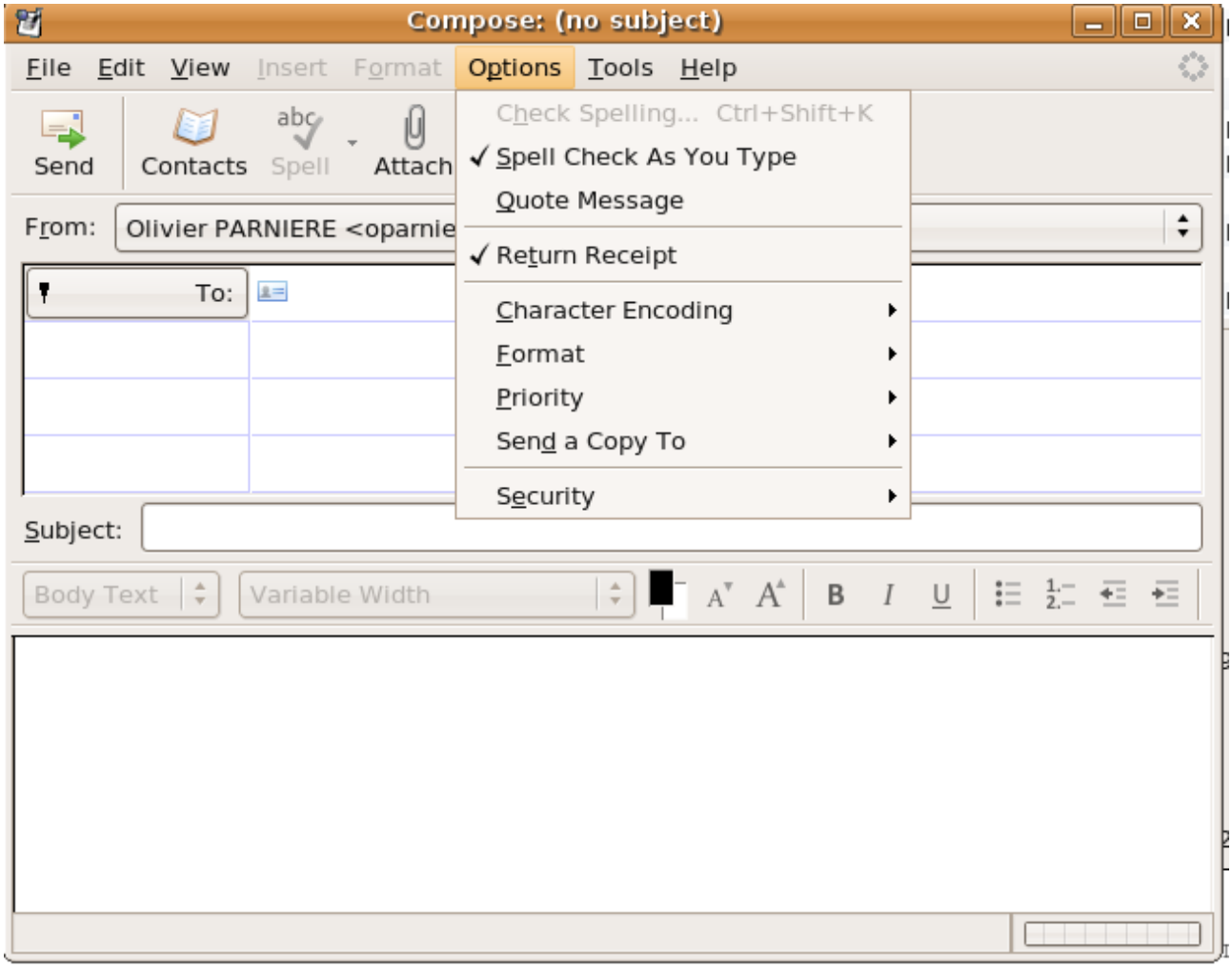



*Illustration 4: Activation de la demande de MDN*

#### 6.5.1.2.4 Traitement du MDN en réception

Lorsque le destinataire clique dans la liste des messages pour lire un message, si celui ci contient l'entête référant au MDN alors une fenêtre s'ouvre :

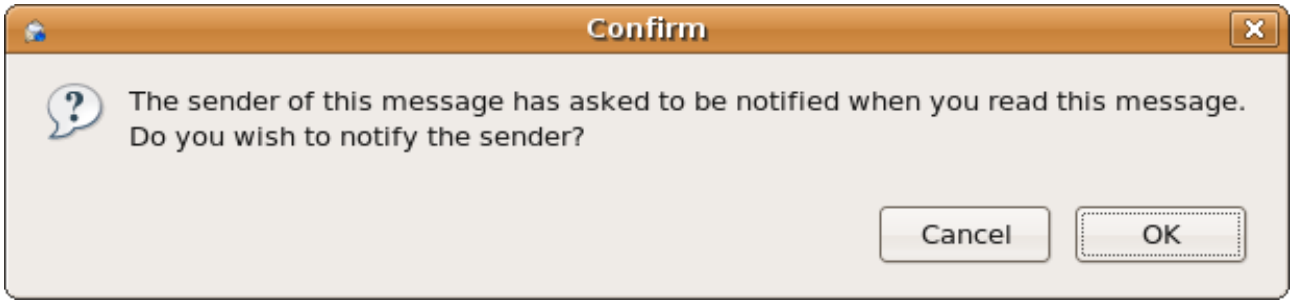



*Illustration 5: Fenêtre de confirmation d'envoi de notification en lecture*

### *6.5.1.3 Partie non implémentée*

La partie que Thunderbird n'implémente pas est l'envoi de notification quand un message (munie d'une demande MDN) est supprimé. (RFC 3798 3.2.6.2)

### *6.5.1.4 Implémentation de la notification de suppression de message*

#### 6.5.1.4.1 Algorithme

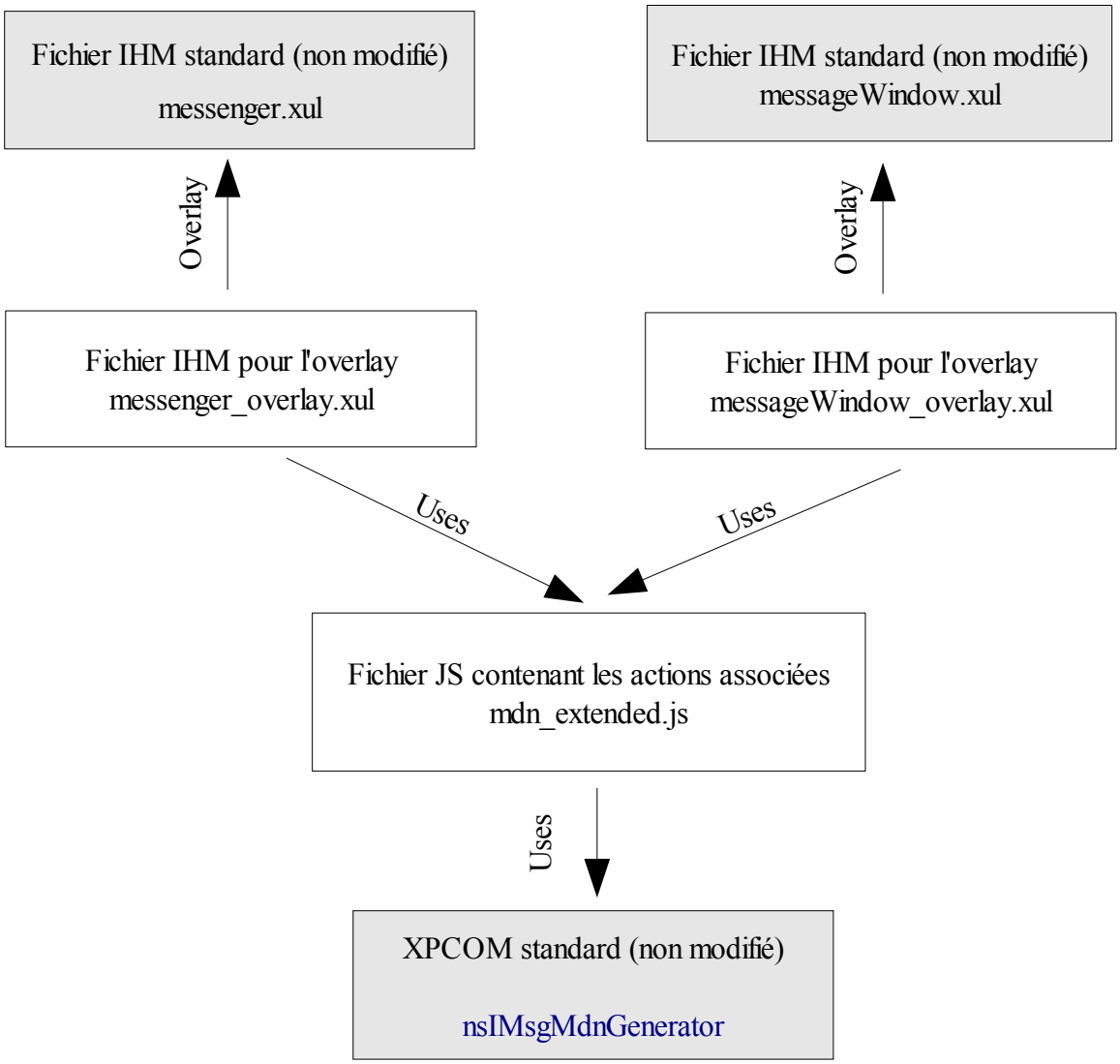
- Pour un message sélectionné dans la liste des messages
  - intercepter l'événement de suppression
  - si le message contient une demande MDN
    - Lancer une PopUp de confirmation d'envoi de MDN
      - « The sender of this message has asked to be notified when you delete this message . \n Do you widh to notify the sender? »
      - Actions disponibles (Ok/Cancel) avec focus sur Ok
    - envoyer un message de notification de suppression
    - Ajouter un état de traitement au message pour éviter l'envoi multiple de MDN

#### 6.5.1.4.2 Architecture

##### 6.5.1.4.2.1 Création de l'extension en utilisant le système des écouteurs (listeners)

La gestion des MDN lors de la suppression d'un message contenant une demande de MDN peut être implémentée sous la forme d'une extension contenant

- un fichier messenger_overlay.xul effectuant un overlay sur le fichier de base messenger.xul.
- un fichier messageWindow_overlay.xul effectuant un overlay sur le fichier de base messageWindow.xul.
- Ces deux fichiers XUL incluent un fichier JS mdn_extended.js qui ajoute un controleur à la liste « top.controllers » afin de traiter les cas cmd_delete, cmd_shiftDelete et button_delete.



#### 6.5.1.4.2.2 Code de la génération de la notification MDN en suppression

Code :

Récupérer le message sélectionné, les headers du messages et les headers MIME :

```
var msgURI = GetLoadedMessage();
var msgHdr = messenger.msgHdrFromURI(msgURI);
mimeHdr = aUrl.mimeHeaders;
```

Tester si le message contient un MDN :

```
var DNTHeader = mimeHdr.extractHeader("Disposition-Notification-To", false);
var oldDNTHeader = mimeHdr.extractHeader("Return-Receipt-To", false);
if (!DNTHeader && !oldDNTHeader)
    return;
```

Instantiation un Générateur d'un MDN (XPCOM) et exécution:

```
var mdnGenerator = Components.classes["@mozilla.org/messenger-
mdn/generator;1"].
createInstance(Components.interfaces.nsIMsgMdnGenerator);

mdnGenerator.process(MDN_DISPOSE_TYPE_DELETED, msgWindow, msgFolder,
msgHdr.messageKey, mimeHdr, false);
```

avec MSN_DISPOSE_TYPE_DELETED étant un long constant valant 3.

Si le message sélectionné est dans le répertoire « Trash » il ne faut pas générer d'accusé de suppression.

## 6.5.2 DSN

### *6.5.2.1 Principe général*

Le *Delivery Status Notification* (DSN) permet à l'émetteur d'un message de s'assurer que tous les MTA (Mail Transfert Agent) relayant le message au destinataire ont bien délivré le message à leurs destinataires respectifs et donc que le dernier MTA a délivrer le message au destinataire.

Dans le cas où le MTA suivant le MTA courant ne supporte plus DSN, alors le MTA courant génère une notification à l'émetteur.

La demande de DSN se situe au niveau enveloppe.

### *6.5.2.2 Implémentation*

#### 6.5.2.2.1 Analyse de l'extension DSN à SMTP

Lors de l'interrogation du MTA par le client Mail :

Le client émet : EHLO

Si le MTA répond : 250-DSN comme extension SMTP supporté alors le MTA supporte DSN

Alors le client peut utiliser DSN.

le client peut lancer la procédure d'envoi de message :

Tout d'abord il indique son identité d'emission

| MAIL FROM:<monadresse@localhost> RET=FULL ENVID=NS40112696JT SIZE=335 |
|---|

Les mots clefs relatifs à DSN sont RET et ENVID

- RET peut valoir FULL ou HDRS. Le but de ce champs est de savoir si la notification doit contenir tout le contenu du message (FULL) ou juste ses entêtes (HDRS).
- ENVID (optionnel) correspond à l'identifiant de l'enveloppe (enveloppe identifier). L'idée de ce champs et de permettre à l'émetteur d'identifier pour quel message est destiné un retour de DSN.

Ensuite l'émetteur indique pour chaque adresse de destinataire :

| RCPT TO:<M le Destinataire> NOTIFY=SUCCESS,FAILURE ORCPT=rfc822;dest@localhost |
|---|

Les mots clefs relatifs à DSN sont :

- NOTIFY (RFC 1891 5.1 ) peut valoir NEVER, SUCESS, FAILURE, DELAY.
  - NEVER ne peut être combiné avec les autres valeurs. Cette valeur indique aux MTA de ne jamais renvoyer de notification.

- SUCCESS ou FAILURE indique aux MTA de renvoyer respectivement une notification de succès ou d'échec de délivrance.
- DELAY indique aux MTA de renvoyer une notification si le message a été retardé lors de sont parcours et qu'aucun état de de succès ou d'échec peut être déterminé.
- les paramètres SUCESS / FAILURE / DELAY peuvent être combinés.

- ORCPT indique le destinataire original du message (utile en cas de Forward). le mot clef est suivi du type de l'adresse suivie elle-même de l'adresse du destinataire.

#### 6.5.2.2.2 Mise en pratique dans Thunderbird

##### 6.5.2.2.2.1 Code existant datant de Netscape Communicator

Quelque existant se trouve dans la Classe mailnews/compose/src/nsSmtpProtocol.cpp.

Par contre aucune gestion de l'interface graphique est implémentée, le code n'est pas compilable et tout la norme DSN n'est pas intégrée.

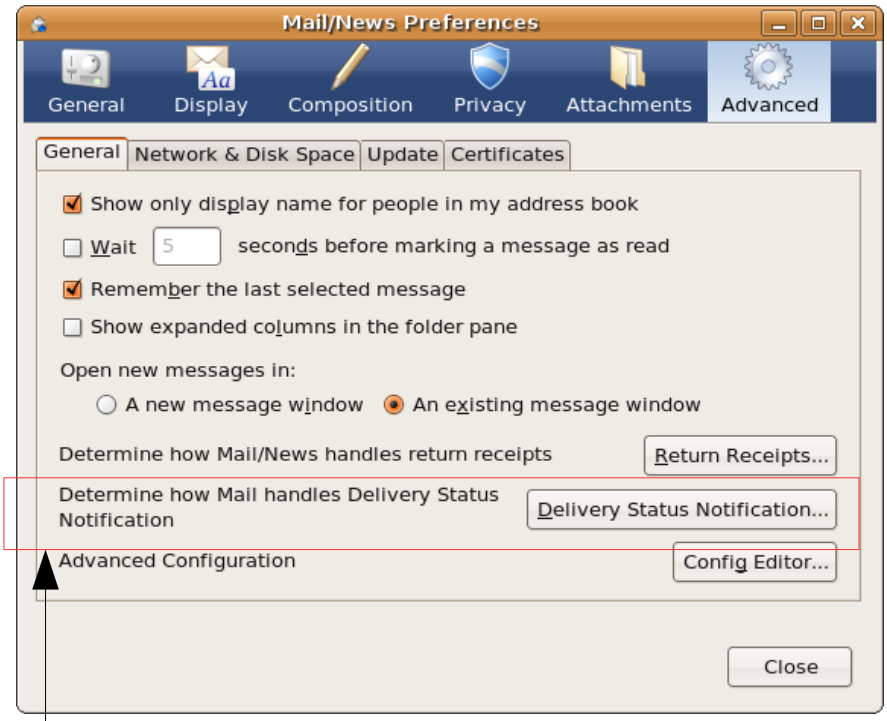
##### 6.5.2.2.2.2 Architecture de l'implémentation de DSN

**Gestion de l'interface graphique : intégration de DSN**

**Préférence générale**

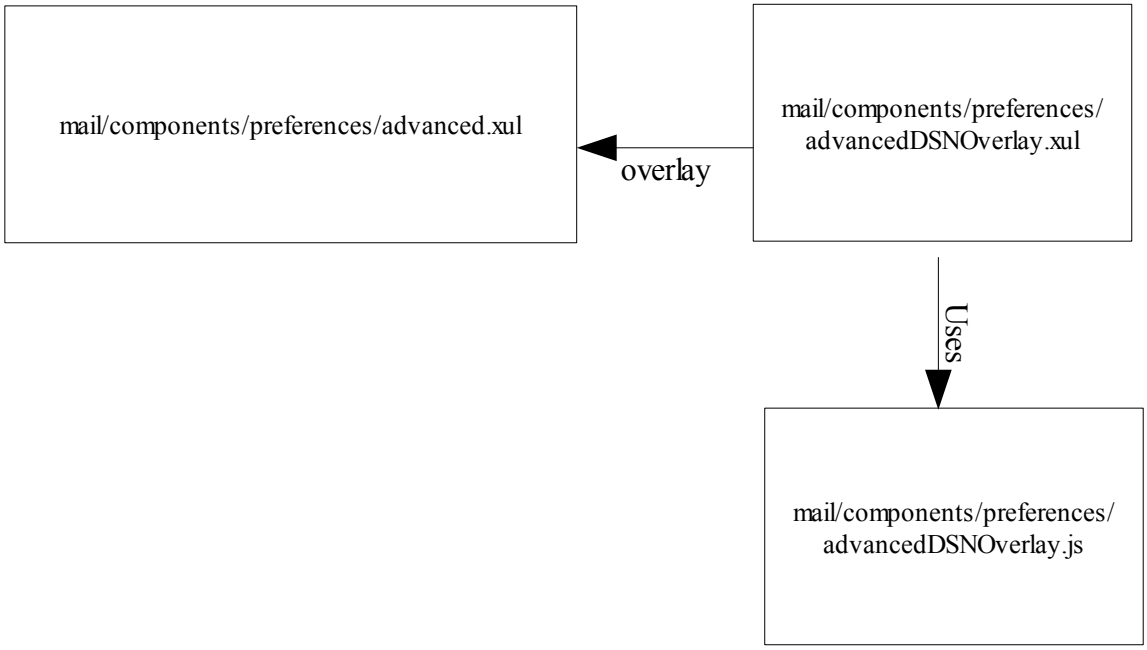
Création d'une entrée dans l'onglet général du menu préférences générales avancées. (Edit/Preferences).

**Partie XUL**

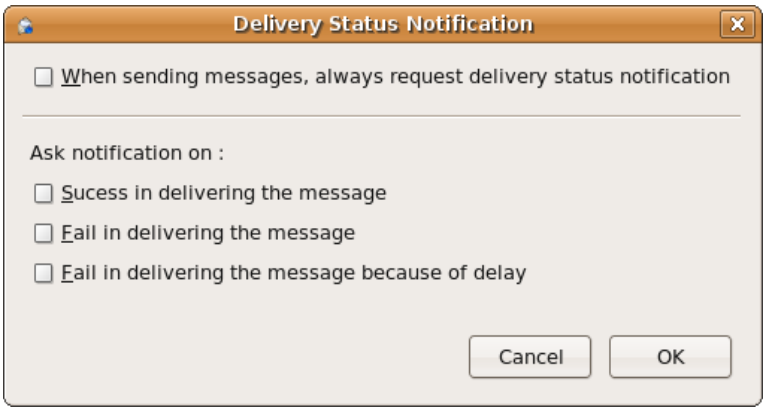



Description et bouton d'accès à la fenêtre de paramétrages du DSN

Pour intégrer, cette modification de la fenêtre de préférence, l'extension doit contenir un Overlay qui ajoute le bouton « Delivery Status Notification » au panel existant (mail/components/preferences/advanced.xul).




Quand l'utilisateur clique sur le bouton «Delivery Status Notification», la fenêtre de paramétrages du DSN apparaît.



Cette fenêtre est à implémenter dans l'extension sous la forme d'un fichier xul nommé *DSNGeneralSettingOverlay.xul* accompagné du fichier javascript *DSNGeneralSettingOverlay.js.*

Détails :

- La première case liée à l'envoi de notification automatique force la demande de notification pour chaque envoi de message.
- La case (Success), la case (fail), ainsi que le case (delay) déterminent le type de demande de notification. Ces trois cases peuvent être cochées en même temps

Contraintes :

- si la première case est cochée, alors au moins une des trois autres cases spécifiant le type de notification doit être coché aussi. En effet une demande de notification automatique pour chaque message doit être obligatoirement accompagnée d'un type.

**Javascript**

Dans le fichier à implémenter *mail/components/preferences/advancedDSNOverlay.js*

- Ajouter un écouteur qui capture l'événement de click que le bouton « Delivery Status Notification» et implémenter l'affichage de la fenêtre de paramétrage « Delivery Status Notification ».

Dans le fichier à implémenter *DSNGeneralSettingOverlay.js,* enregistrer les différents états des cases dans les préférences de l'utilisateur.

- case qui active l'envoi automatique de notification, enregistrer Vrai en booléen dans le champs mail.dsn.request_dsn_on si la case est activé, faux sinon.
- case qui active l'envoi de notification si le message à été délivré avec Succès, enregistrer Vrai en booléen mail.dsn.request_dsn_onsucess si la case est activé, faux sinon
- case qui active l'envoi de notification si le message n'a pas été délivré, enregistrer Vrai en booléen mail.dsn.request_dsn_onfail si la case est activé, faux sinon
- case qui active l'envoi de notification si le message n'a pas été délivré suite à un retard de traitement, enregistrer Vrai en booléen mail.dsn.request_dsn_ondelay si la case est activé, faux sinon

Exemple de code pour la préférence mail.dsn.request_dsn_on :

- Créer un élément XUL <preferences/> listant la préférence à intégrer :

```
<preferences>
  <preference id="mail.dsn.request_dsn_on"
     name="mail.dsn.request_return_dsn_on" type="bool"/>
</preferences>
```

- Déclarer la case  et sa définition :

```
      <checkbox id="alwaysRequest" label="When sending messages, always request
delivery status notification"
       preference="mail.dsn.request_dsn_on"
       accesskey="W"/>
```

**Préférence relative à un compte de messagerie**

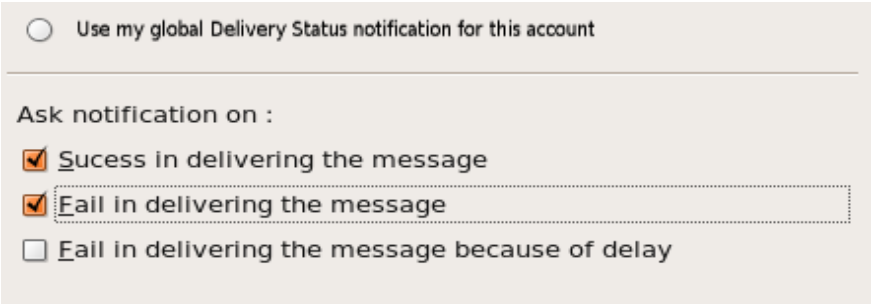
Ajouter une entrée de confirmation pour chaque compte de messagerie.

Accès du menu : *Edit -> AccountSettings*

**Partie Xul**

Ajouter une entrée nommée « Delivery Notification » pour chaque compte de l'arbre en dessous de l'entrée nommée « Return Receipts ».

Panel de configuration DSN pour chaque compte :




Contrôles à effectuer :

- si le radio boutton « use my global ... » est coché alors les préférences générales prennent le pas sur la configuration relative au compte. Donc les autres champs doivent être grisés.
- Gérer la persistance des préférences pour chaque compte en configuration en prenant comme prefix mail.server.%server% (prendre exemple sur am-mdn.xul)

#### Préférence du message

Ajout d'une entrée « Request Delivery Notification » dans le menu « Option » de la fenêtre de composition de message. L'option cliquée enclenche la demande de notification de délivrance pour le message courant. Son état (coché / non coché) est conditionné par les préférences du compte courant.

L'utilisateur peut donc cocher ou décocher suivant le cas pour demander ou non une notification.

#### Partie XUL

L'extension du menu doit être implémenté grâce au système d'overlay.

Le fichier XUL *messengercomposeOverlayDSN.xul* doit implémenter l'ajout de l'entrée « Request Delivery Notification » dans l'élément menu dont l'id est *optionsMenu*, la gestion des événements et du code métier doit être implémentée dans le fichier messengercomposeOverlayDSN.xul.

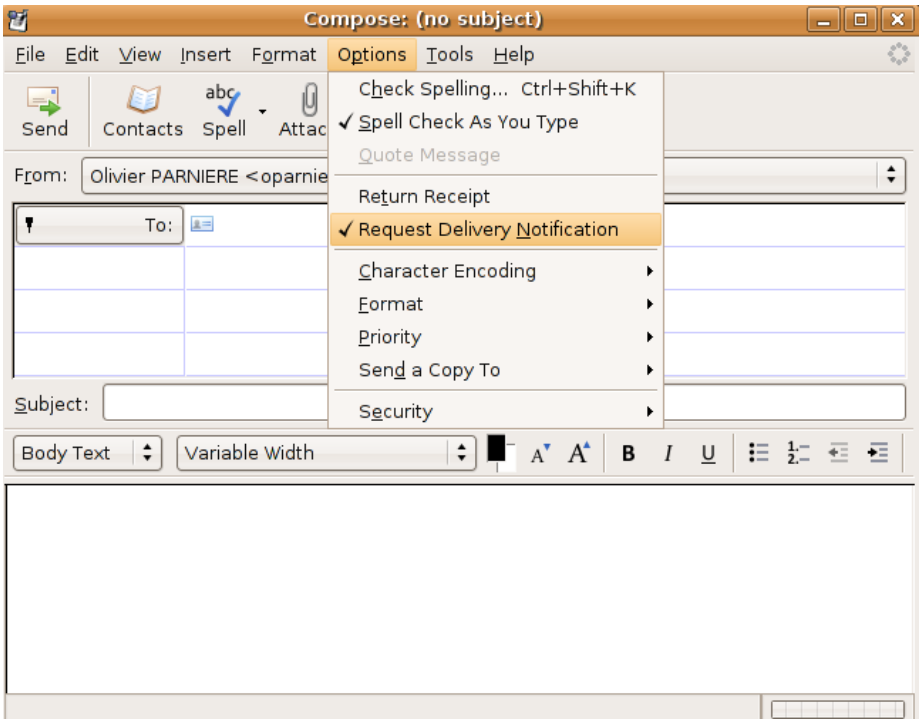
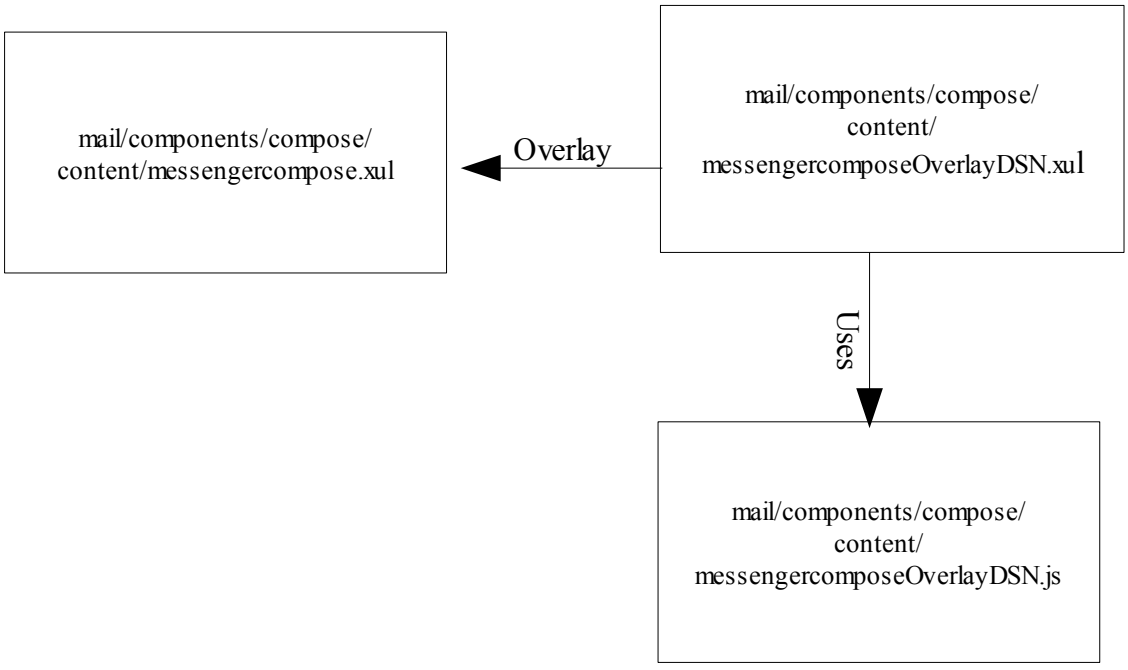



*Illustration 6: Ajout de l'entrée lié au DSN*



### Javascript

Dans *messengercomposeOverlayDSN.js,* il faut gérer l'événement du clique sur l'élément « Request Delivery Notification » pour que la fenêtre d'options du DSN s'ouvre. De plus, il faut ajouter le chargement des préférences générales liés aux DSN dans l'objet *nsIMsgIdentity.*

Attention *nsIMsgIdentity* ne contient pas de base les champs permettant d'enregistrer les options DSN par défaut de l'utilisateur courant.Ci – après seront détaillées les modifications à y apporter.

Dans *dsnOptions.js,* il faut gérer l'enregistrement des options choisies pour la composition du message courant dans l'objet *nsICompFields* contenu dans l'objet *nsIMsgCompose* courant.

Attention *nsICompFields* ne contient de base les champs permettant d'enregistrer les options DSN relative au message en cours d'envoi, ci-après seront détaillés les ajouts à implémenter.

### Modification de couche XPCOM

#### Modification de la l'interface nsIMsgIdentity.idl et nsIMsgCompFields.idl

#### Modification de l'XPIDL

- Ajouter : attribute boolean deliveryNotificationRequested
- Ajouter : attribute boolean deliveryNotificationOnSuccess
- Ajouter : attribute boolean deliveryNotificationOnFailure
- Ajouter : attribute boolean deliveryNotificationOnDelay

#### Génération des classes C++ XPCOM

Régénérer les classes correspondantes avec les même Id que le composant original grâce à XPIDL. Réintégrer les sources original.

#### Implémentation du composant nsIMsgIdentity.idl et nsIMsgCompFields.idl

Modifier les sources en ajoutant le support des setters/getters relatifs aux attributs ajouter à l'IDL.

#### Modification de l'implémentation de l'interface nsIMsgSend.idl

Ce composant XPCOM est responsable de l'envoi du message vers le service SMTP *nsISmtpService*.

Il faut modifier la classe implémentante ndMsgSend.cpp pour ajouter la copie des nouveaux attributs DSN de *nsIMsgIdentity* et *nsIMsgCompFields* dans les attributs locaux de la classe.

De plus, il faut ajouter l'objet local représentant `nsIMsgCompFields` dans la signature de la méthode d'appel :

```
rv = smtpService->SendMailMessage(aFileSpec, buf, mUserIdentity,
                                  mSmtpPassword.get(), uriListener,
                                  msgStatus,
                                  callbacks, nsnull,

        getter_AddRefs(mRunningRequest));
```

en

```
rv = smtpService->SendMailMessage(aFileSpec, buf, mUserIdentity, mCompFields
                                  mSmtpPassword.get(), uriListener,
                                  msgStatus,

                                  callbacks, nsnull,

        getter_AddRefs(mRunningRequest));
```

**Modification du composant nsISMTPService.idl**

changer la signature de la méthode :

```
void SendMailMessage(in nsIFileSpec aFilePath, in string aRecipients,
                     in nsIMsgIdentity aSenderIdentity,
                     in string aPassword,
                     in nsIUrlListener aUrlListener,
                     in nsIMsgStatusFeedback aStatusListener,
                     in nsIInterfaceRequestor aNotificationCallbacks,
                     out nsIURI aURL,
                     out nsIRequest  aRequest);
```

en

```
void SendMailMessage(in nsIFileSpec aFilePath, in string aRecipients,
                     in nsIMsgIdentity aSenderIdentity,
                     in nsIMsgCompFields aMessagePreferences,
                     in string aPassword,
                     in nsIUrlListener aUrlListener,
                     in nsIMsgStatusFeedback aStatusListener,
                     in nsIInterfaceRequestor aNotificationCallbacks,
                     out nsIURI aURL,
                     out nsIRequest  aRequest);
```

Générer les fichiers d'implémentation grâce à XPIDL, inclure l'ancien code ainsi que le code de la méthode `SendMailMessage`(), et y est intégrer la gestion de l'objet *aMessagePrefence*.

Modifier la méthode appelée dans `SendMailMessage()` :

```
rv = NS_MsgBuildSmtpUrl(aFilePath, smtpHostName, smtpPort, smtpUserName,
                        aRecipients, aSenderIdentity, aUrlListener,
                       aStatusFeedback,
                        aNotificationCallbacks, &urlToRun);
```

en

```
rv = NS_MsgBuildSmtpUrl(aFilePath, smtpHostName, smtpPort, smtpUserName,
                        aRecipients, aSenderIdentity,
                      aMessagePreferences,aUrlListener,
                       aStatusFeedback,
                        aNotificationCallbacks, &urlToRun);
```

et la modifier pour quelle génère un objet *nsIURI* contenant les informations *aMessagePreference* pour qu'on puisse passer ces informations au service *nsISMTPProtocol*

#### Modification de l'implémentation de l'interface nsISMTPProtocol.idl

Ce composant gère le protocole SMTP, et c'est à ce niveau qu'il faut ajouter les éléments spécifiques DSN à l'implémentation du protocole SMTP.

Récupérer les préférences DSN du message, grâce à l'objet nsIURI passé en paramètres de la méthode `nsSmtpProtocol::LoadUrl()`

`Dans la méthode nsSmtpProtocol::SendMailResponse():`

- Tester si le serveur SMTP supporte le DSN
  - `TestFlag(SMTP_EHLO_DSN_ENABLED)`
- Si c'est le cas et que la préférence *deliveryNotificationRequested* est à Vrai, forger la demande SMTP en fonction des préférence DSN comme suit :

```
buffer = "RCPT TO:<";

buffer += m_addresses;
buffer += "> NOTIFY=TYPETOREPLACE ORCPT=rfc822;";
buffer += encodedAddress;
buffer += CRLF;
```

avec TYPETOREPLACE valant SUCCESS et/ou FAILURE et/ou DELAY en fonction de la valeur booléenne des préférences *deliveryNotificationOnSuccess*, d*eliveryNotificationOnFailure, deliveryNotificationOnDelay* courantes.

Dans la méthode `nsSmtpProtocol::SendHeloResponse()`

- tester si le serveur SMTP supporte le DSN
  - `TestFlag(SMTP_EHLO_DSN_ENABLED)`

- et que l'objet préférence *deliveryNotificationRequested* du message courant contient une demande DSN.

  Alors générer :

```
buffer = "MAIL FROM:<";
buffer += fullAddress;
buffer += ">";
buffer += " RET=FULL ENVID=ENVELOPID";
```

  avec ENVELOPID valant la valeur de l'header Message-ID

## 6.5.3 Signatures des notifications

### 6.5.3.1 Principe général

Cette fonctionnalité correspond à l'implémentation de la partie « Signed Receipt » de la RFC 2634: http://www.ietf.org/rfc/rfc2634.txt. Cette RFC doit servir de référence pour le détail de cette implémentation.

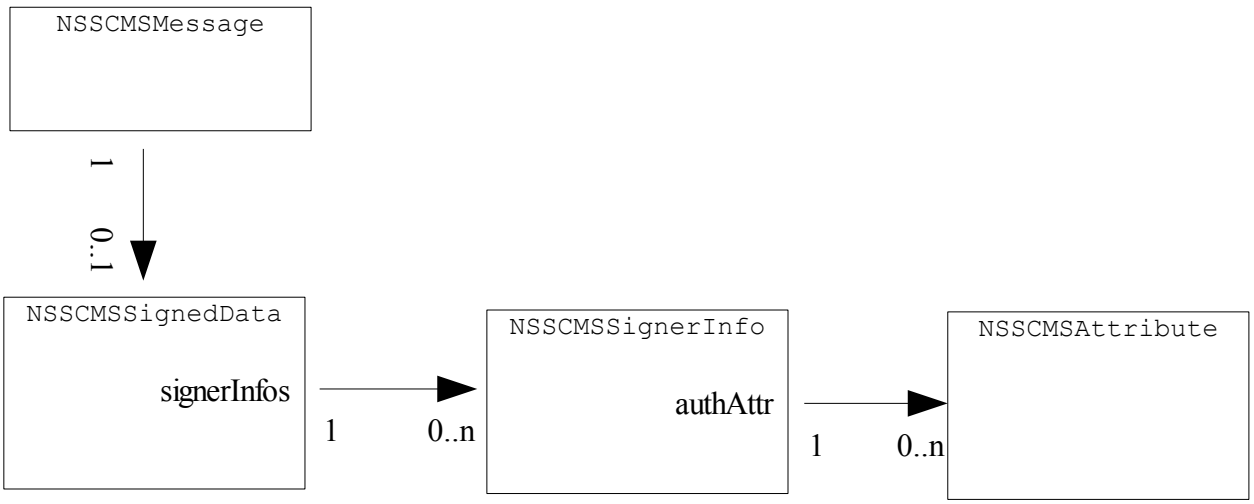
En résumé, l'objectif de cette partie de la RFC est de fournir la possibilité lors de l'envoi d'un mail de demander un accusé de réception signé, et lors de la réception d'une telle demande de générer l'accusé de réception signé.

Cette fonctionnalité fait partie des travaux en cours de la fondation Mozilla : http://www.mozilla.org/projects/security/pki/nss/smime/. Il est donc prévu de l'intégrer directement au code source de Thunderbird sans passer par une extension. De plus, ceci facilitera l'ajout de cette fonctionnalité car il est nécessaire d'intervenir au coeur du produit Thunderbird. Ce code sera ensuite proposé à Mozilla pour validation et intégration, il est donc primordial de respecter les conventions Mozilla lors du développement de cette partie.

### 6.5.3.2 Architecture

L'implémentation de cette RFC consiste principalement à ajouter des attributs dans les éléments de signature d'un message. Un mail avec demande d'accusé de réception signé doit forcément être un mail signé. Lors de la réception, ces mêmes attributs sont lus pour déterminer s'il est nécessaire de signer l'accusé de réception.

Lors de l'envoi d'un mail, l'API Thunderbird standard construit la structure de données suivante :




Les attributs à ajouter se situent au niveau de la structure « `NSSCMSSignerInfo` » et sont de type « `NSSCMSAttribute` ».

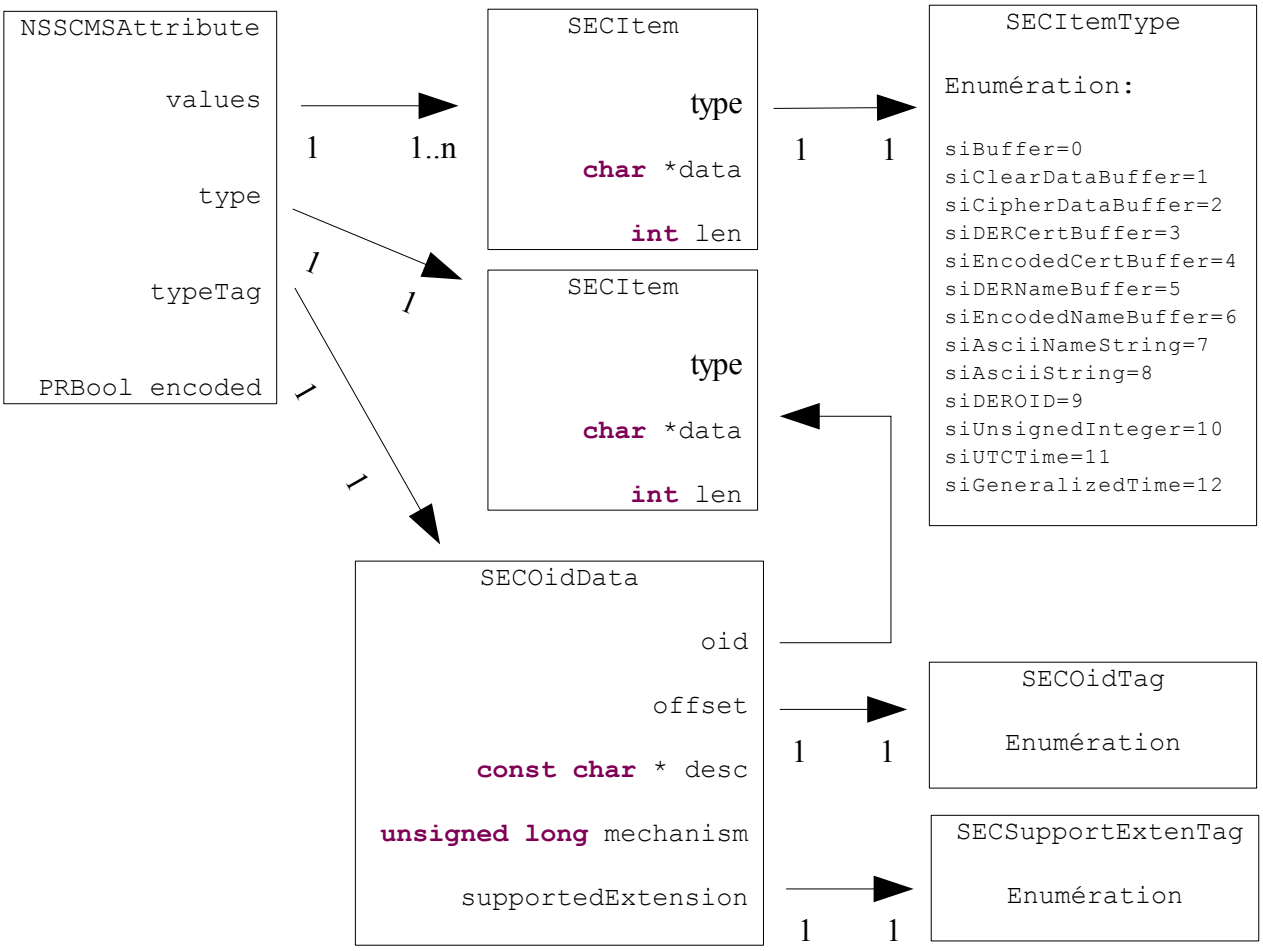
Cette structure est définie par les fichiers :

- cmst.h
- cmssiginfo.c
- cmssigdata.c

Pour ajouter un attribut, il faut utiliser la méthode :

```
SECStatus NSS_CMSSignerInfo_AddAuthAttr(NSSCMSSignerInfo *signerinfo,
NSSCMSAttribute *attr);
```

Une structure «  NSSCMSAttribute » est définie de la façon suivante :




L'attribut à ajouter un est « receiptRequest » dont la structure est définie par la RFC 2634.

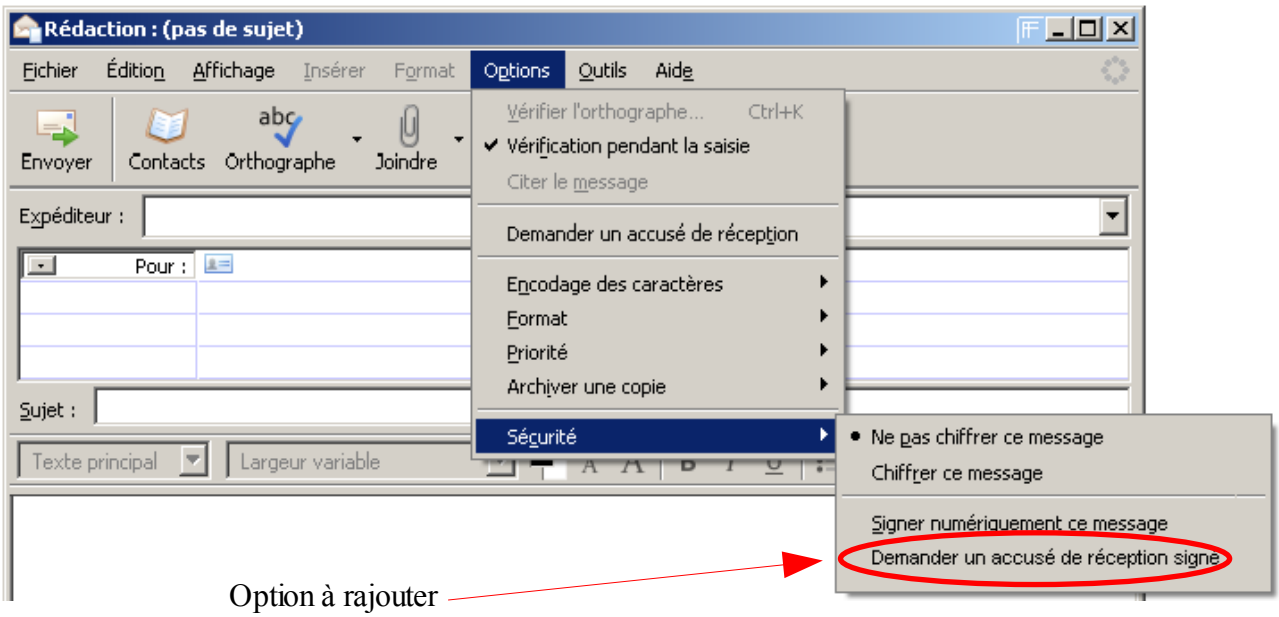
### *6.5.3.3 Envoi d'un message avec demande d'accusé de réception signé*

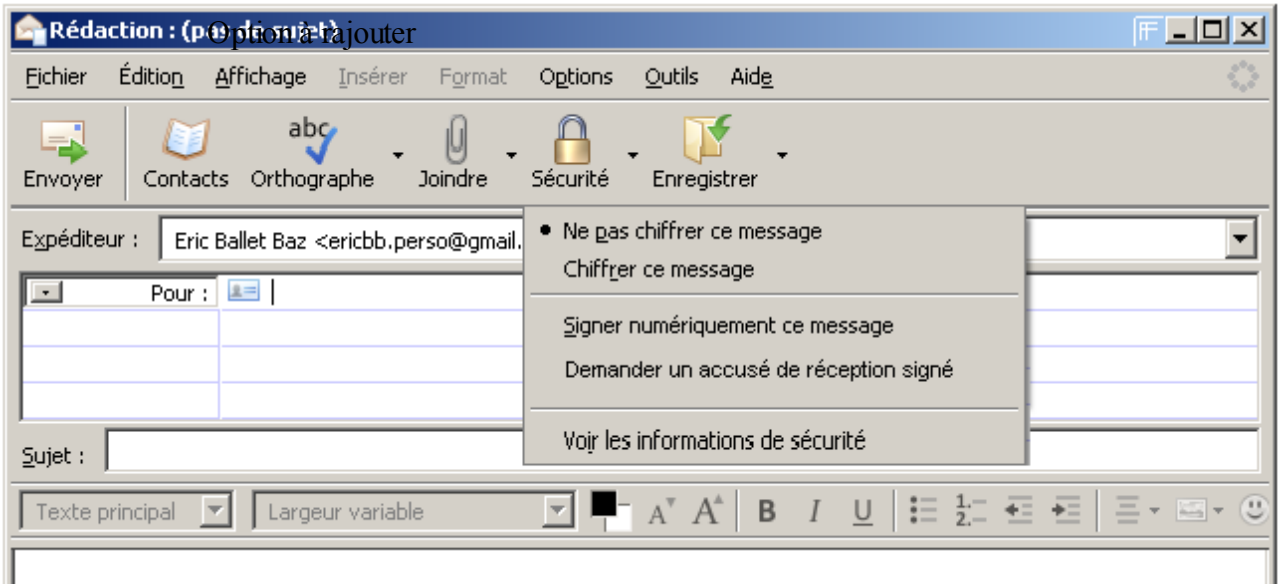
#### 6.5.3.3.1 Implémentation IHM

Pour envoyer un message avec une demande d'accusé de réception signé, il est obligatoire de signer le message. La notion d'accusé de réception signé est donc fortement liée aux fonctionnalités de sécurité, c'est pourquoi il est préférable de l'implémenter à ce niveau tant au niveau de l'IHM que du back-end.

Au niveau de l'IHM, la gestion de la signature est actuellement implémentée par l'utilisation d'un flag « signMessage », stocké dans un objet nsIMsgSMIMECompFields. L'ensemble du code gérant le positionnement de ce flag et l'interaction avec l'IHM est contenu dans le fichier JS « msgCompSMIMEOverlay.js ».

L'objet nsIMsgSMIMECompFields est ensuite fourni au service XPCOM d'envoi de message, qui se charge de la signature en fonction de ce flag. Il est donc nécessaire de compléter cette classe pour pouvoir stocker la demande d'accusé signé. Il est également nécessaire de compléter l'IHM afin que l'utilisateur puisse choisir cette option. Pour cela, les menus liés à la sécurité seront complétés comme démontré dans les deux captures d'écran suivantes:






L'activation par l'utilisateur de l'option accusé signé ne doit être possible que si l'option « signer numériquement ce message est activée ».

Il n'est pas prévu de gérer un paramétrage par défaut, car ceci complexifie les combinaisons à vérifier.

#### 6.5.3.3.2 Implémentation XPCOM

Afin de minimiser les impacts de cet ajout de code, l'essentiel de l'implémentation de cette fonctionnalité se situera dans le fichier « nsMsgComposeSecure.cpp » au niveau de la classe « nsMsgComposeSecure ».

Cette implémentation récupérera la demande d'accusé signé depuis l'objet nsIMsgSMIMECompFields fourni par l'IHM.

Ensuite, il faut compléter la méthode « MimeFinishMultipartSigned » pour remplir la structure de données représentant les informations de signature après création de celle-ci par le code standard.

### *6.5.3.4 Réception d'un message avec demande d'accusé de réception signé*

## 6.6 Sécurité

### 6.6.1 Triple enveloppe

#### 6.6.1.1 Principe général

Cette fonctionnalité correspond à l'implémentation de la partie « Triple Wrapping » de la RFC 2634: http://www.ietf.org/rfc/rfc2634.txt. Cette RFC doit servir de référence pour le détail de cette implémentation.

En résumé, l'objectif de cette partie de la RFC est de fournir la possibilité lors de l'envoi d'un mail d'utiliser une triple enveloppe. Un message utilisant une triple enveloppe est un message qui a été signé, puis chiffré, puis signé.

Pour avoir plus d'informations, sur les notions d'enveloppe, de signature et de chiffrement, vous pouvez vous référer aux documents suivants :

http://fr.wikipedia.org/wiki/Multipurpose_Internet_Mail_Extensions

http://tools.ietf.org/html/rfc1847

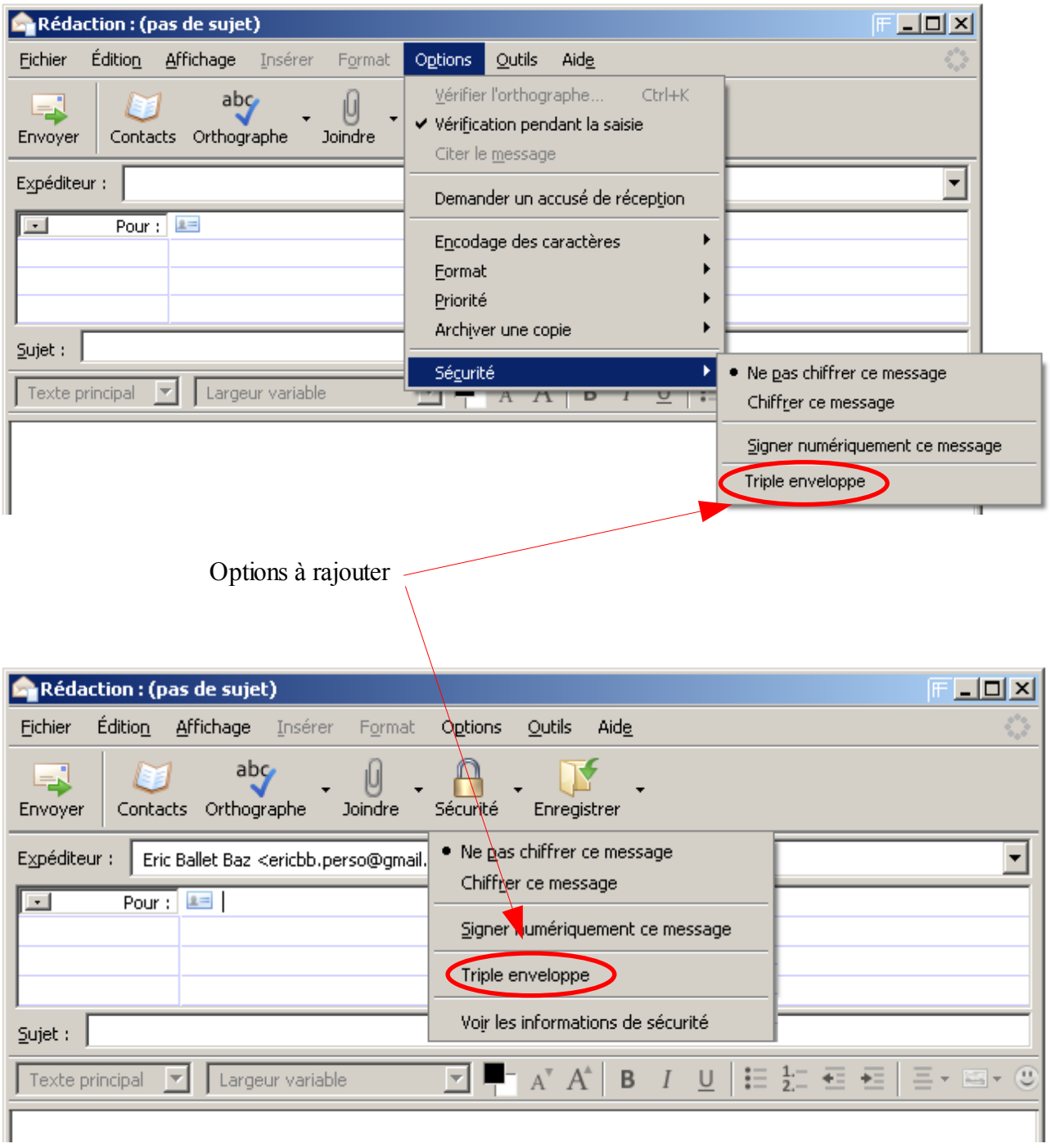
Cette fonctionnalité fait partie des travaux en cours de la fondation Mozilla : http://www.mozilla.org/projects/security/pki/nss/smime/. Il est donc prévu de l'intégrer directement au code source de Thunderbird sans passer par une extension. De plus, ceci facilitera l'ajout de cette fonctionnalité car il est nécessaire d'invernir au coeur du produit Thunderbird. Ce code sera ensuite proposé à Mozilla pour validation et intégration, il est donc primordial de respecter les conventions Mozilla lors du développement de cette partie.

#### 6.6.1.2 Envoi d'un message utilisant le mode triple enveloppe

##### 6.6.1.2.1 Implémentation IHM

Au niveau de l'IHM, la gestion de la signature et du chiffrement est actuellement implémentée par l'utilisation de deux flags « signMessage » et « requireEncryptMessage », stocké dans un objet nsIMsgSMIMECompFields. L'ensemble du code gérant le positionnement de ces flags et l'interaction avec l'IHM est contenu dans le fichier JS « msgCompSMIMEOverlay.js ».

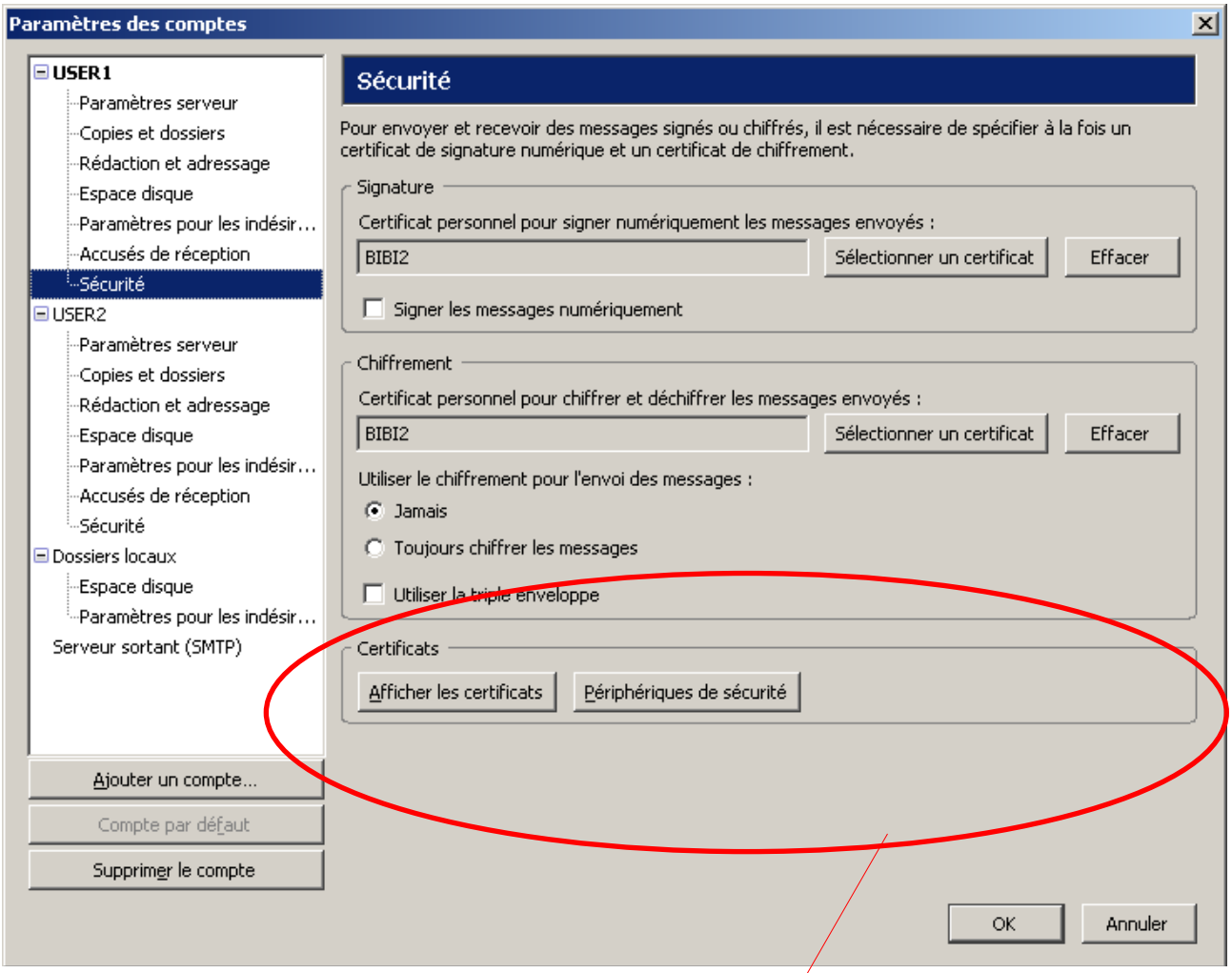
L'objet nsIMsgSMIMECompFields est ensuite fourni au service XPCOM d'envoi de message, qui se charge de la signature et du chiffrement en fonction de ces deux flags. Il est donc nécessaire de compléter cette classe pour pouvoir stocker la demande de triple enveloppe. Il est également nécessaire de compléter l'IHM afin que l'utilisateur puisse choisir ce mode de sécurité. Pour cela, les menus liés à la sécurité seront complétés comme démontré dans les deux captures d'écran suivantes:




L'activation par l'utilisateur de la triple enveloppe force les sélections « Chiffrer ce message » et « Signer numériquement ce message » puis désactive ces menus.

Il est également nécessaire de compléter l'écran de paramètrage des comptes, afin que l'utilisateur puisse définir s'il souhaite utiliser systématiquement la triple enveloppe lors de l'envoi de message.

De plus, la sélection du mode triple enveloppe n'est possible que si un certificat est défini pour la signature et le chiffrement.




Paramétrage du comportement par défaut concernant la triple enveloppe

#### 6.6.1.2.2 Implémentation XPCOM

L'essentiel de l'implémentation de cette fonctionnalité pour la partie envoi d'un message se situera dans le fichier « nsMsgComposeSecure.cpp » au niveau de la classe « nsMsgComposeSecure ».

Tout d'abord, il est nécessaire d'ajouter le cas triple enveloppe au niveau de la routine existante permettant de déterminer le type de signature / chiffrement d'un message :

```
nsresult nsMsgComposeSecure::ExtractEncryptionState(nsIMsgIdentity * aIdentity,
nsIMsgCompFields * aComposeFields, PRBool * aSignMessage, PRBool * aEncrypt)
```

Cette fonction sera donc modifiée pour ajouter un type de retour :

`« PRBool * aSignEncryptSign »`

Cette fonction se base sur l'objet nsIMsgSMIMECompFields fourni par l'IHM.

Ensuite, il faut compléter la méthode « BeginCryptoEncapsulation » pour gérer ce nouveau type de retour et donc le mode triple enveloppe. Cette implémentation doit se baser sur les routines existantes permettant de chiffrer ou signer un message.

### *6.6.1.3 Réception d'un message utilisant le mode triple enveloppe*

En mode réception, l'implémentation actuelle est capable de gérer le mode triple enveloppe : la librairie de lecture du format MIME utilise un algorithme récursif et est donc capable de lire un contenu enveloppé plusieurs fois.

# 7 Version 1

## 7.1 Security Labels

## 7.2 Annuaire

### 7.2.1 Affichage du certificat pour un contact

#### 7.2.1.1 Principe général

La fonctionnalité permettant d'afficher le certificat d'un contact sera implémentée sous la forme d'une extension Thunderbird dédiée. Le nom de cette extension sera card_viewer_extended et ce préfixe sera repris le plus souvent possible, notamment pour éviter les collisions avec les objets standards : par exemple dans le nommage des propriétés utilisateurs, des variables globales etc ...

Nous devons distinguer deux types de fiche contact : la fiche d'un contact local, et la fiche d'un contact LDAP. En effet, le mode de récupération du certificat est différent pour chacun de ces deux types de contact.

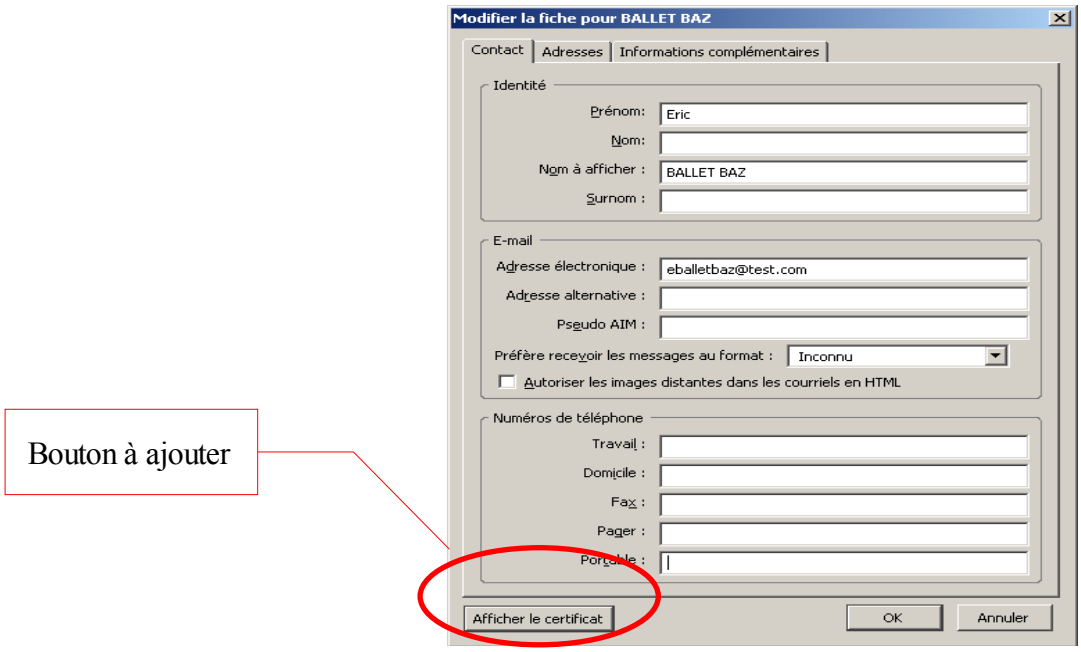
Dans tous les cas, la clé utilisée pour récupérer le certificat d'un contact sera son adresse mail.

Ces deux types de fiches sont affichées en utilisant le même fichier XUL. Celui ci permet de gérer différemment les 2 cas : par exemple champs éditables ou non ...

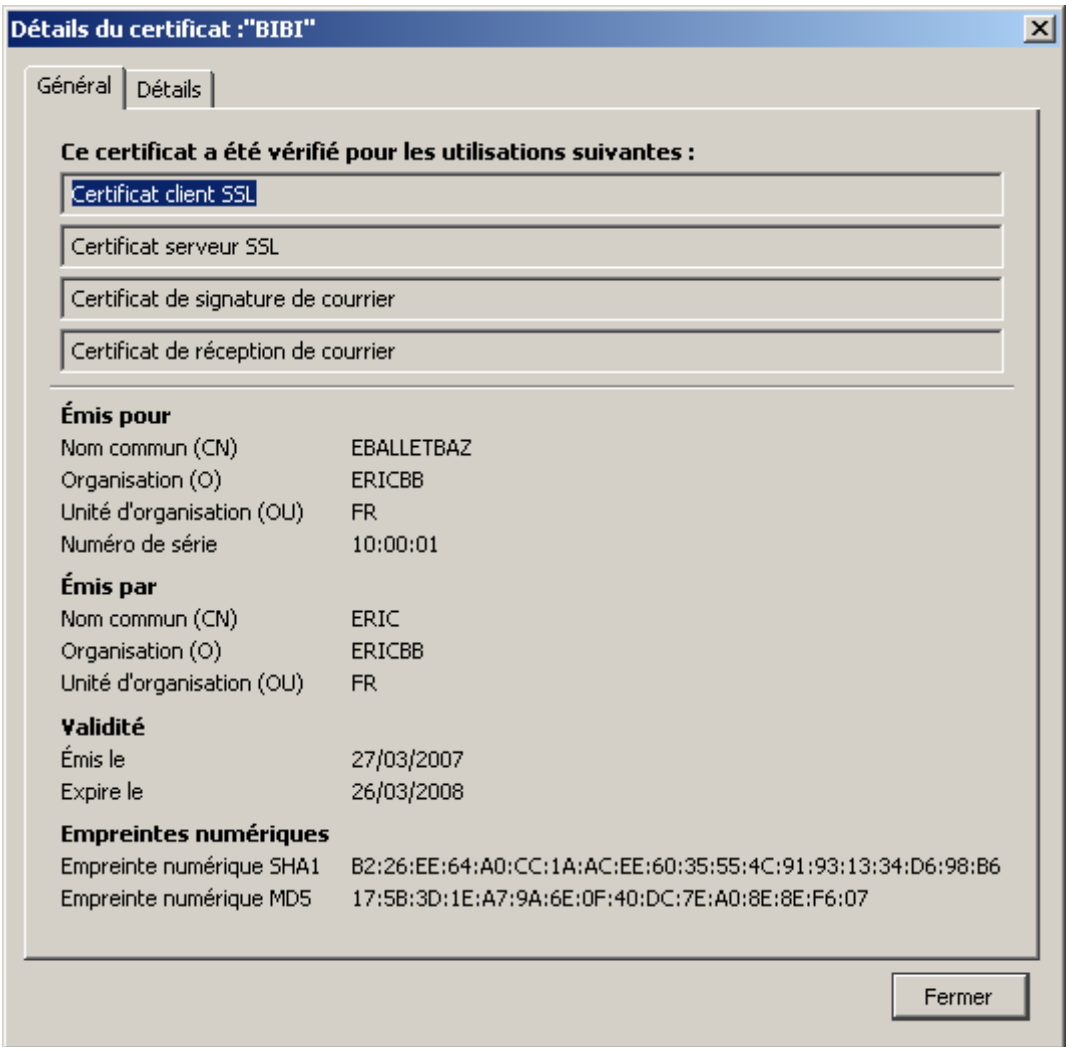
L'attribut LDAP interrogé pour récupérer un certificat est « usercertificate ». Cet attribut est une donnée binaire.

#### 7.2.1.2 Contact local

L'écran existant sera complété afin d'ajouter un bouton permettant l'affichage du certificat :



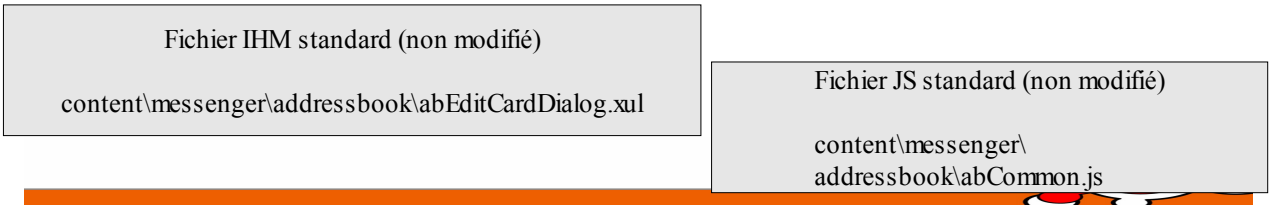

L'activation de ce bouton lancera la fenêtre standard Thunderbird de visualisation d'un certificat :

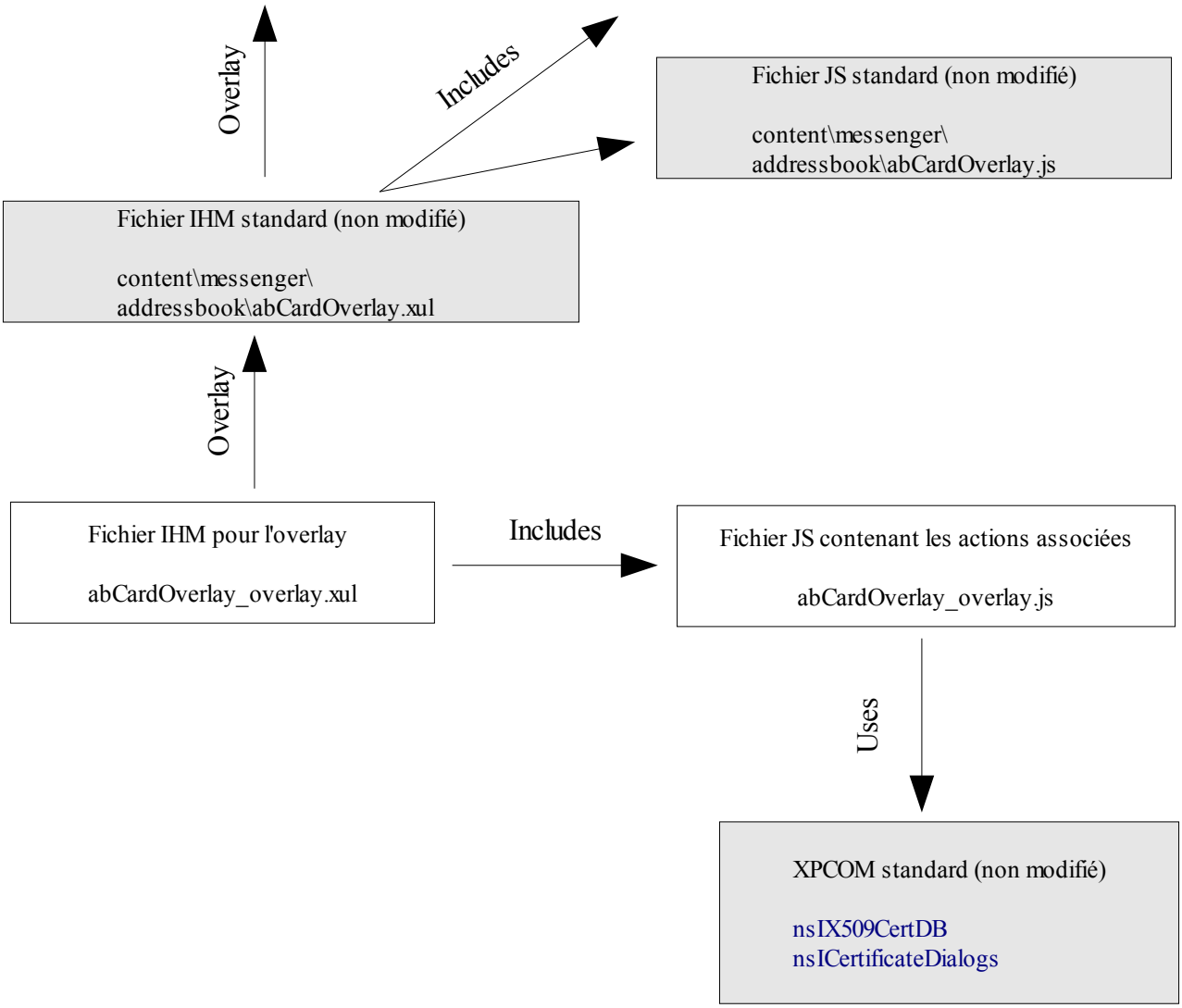



La définition de la fenêtre, pour laquelle ajouter le boutton d'action, et les actions associées sont définies dans l'archive chrome\messenger.jar, par les fichiers :

- content\messenger\addressbook\abEditCardDialog.xul
- content\messenger\addressbook\abCardOverlay.xul (fichier overlay standard appliqué sur le fichier précédent)
- content\messenger\addressbook\abCommon.js
- content\messenger\addressbook\abCardOverlay.js

La technique de l'overlay sera utilisée pour surcharger le comportement de ces fichiers suivant le diagramme suivant :






Le service nsIX509CertDB permet de récupérer un certificat en utilisant une adresse mail comme clé par la méthode :

```
nsIX509Cert findCertByEmailAddress(nsISupports token , char* emailAddress)
```

Cet objet certificat doit ensuite être fourni au service nsICertificateDialogs qui se chargera entièrement de son affichage dans une fenêtre dédiée. Le code Javascript à mettre en oeuvre pourra être repris depuis le fichier existant pippki.js :

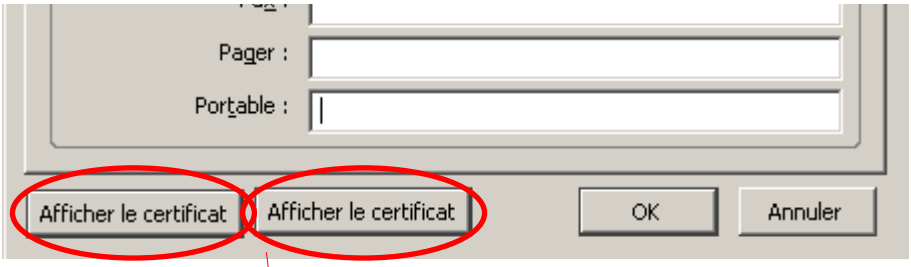
```
const nsICertificateDialogs = Components.interfaces.nsICertificateDialogs;
const nsCertificateDialogs = "@mozilla.org/nsCertificateDialogs;1"

function viewCertHelper(parent, cert) {
  if (!cert) {
    return;
  }

  var cd =
    Components.classes[nsCertificateDialogs].getService(nsICertificateDialogs);
  cd.viewCert(parent, cert);
}
```

Il est préférable de dupliquer ce code dans le fichier abCardOverlay_overlay.js dédié afin de s'affranchir des évolutions du fichier pippki.js.

#### *7.2.1.3 Contact LDAP*

Pour la gestion des contacts LDAP, les modifications à apporter au niveau des écrans ainsi que l'organisation des fichiers XUL et JS sont identiques à celles pour un contact local. En effet, seul le mécanisme de récupération d'un certificat différé.

Il est recommandé de dupliquer le bouton permettant de demander l'affichage d'un certificat et de gérer la visibilité de chacun des 2 boutons selon le type de fiche.




Certificat stocké en local

Certificat stocké dans LDAP

La visibilité des deux boutons est exclusive

Ainsi il sera facilement envisageable, si le besoin se faisait sentir, d'activer les 2 boutons pour un contact LDAP. En effet, on pourrait imaginer que pour la même adresse mail on souhaite afficher le certificat stocké en local et le certificat présent au niveau du LDAP.

Pour l'interrogation d'un annuaire LDAP, l'API XPCOM fournit une interface nsILDAPOperation

et en particulier la méthode :

```
void searchExt ( AUTF8String baseDn , PRInt32 scope , AUTF8String filter ,
PRUint32 attrCount , arrayof char* attributes , PRIntervalTime timeOut ,
PRInt32 sizeLimit )
```

L'attribut LDAP à interroger pour récupérer le certificat est « usercertificate ». Lors de l'interrogation, il est nécessaire d'utiliser la chaîne suivante « usercertificate;binary » pour spécifier que l'attribut est stocké sous forme binaire.

Cet exemple simple ne traite pas de la gestion des données retournées. Un exemple complet de récupération de certificat depuis un annuaire LDAP est disponible dans la librairie certFetchingStatus.js. L'interrogation du LDAP est contenue dans la méthode kickOffSearch() qui est un bon point d'entrée pour comprendre ce code.

## 7.3 Langage

Deux extensions existent pour gérer cette fonctionnalité :

Locale Switcher :

https://addons.mozilla.org/fr/thunderbird/addon/356

Quick Locale Switcher :

https://addons.mozilla.org/fr/firefox/addon/1333

## 7.4 Gestion de la priorité au niveau de l'enveloppe

## 7.5 X400

### 7.5.1 P3

### 7.5.2 P7

## 7.6 Version 2

### 7.6.1 LDAP: import des CRL (Certification Revocation List)

### 7.6.2 Format : P772

#### 7.6.2.1 Affichage

#### 7.6.2.2 Écriture

##### 7.6.2.3 BER PER

## 7.7 Version 3

##### 7.7.1.1 CRL over FTP

## 7.8 Version 4