ACTIMAGE

## Synchronisation des Agendas


# Spécifications Techniques Générales

**Ref : ADAE_Synchro_STG**
**Version 0.5**
**Date : 22 Mars 2005**

## Descriptif du document

| | |
|---|---|
| **Document :** | Synchronisation des Agendas<br>Spécifications Techniques Générales |
| **Référence du document :** | ADAE_Synchro_STG |
| **Etat :** | A valider |
| **Auteur :** | GASSMANN Benjamin, Chef de projet<br>IOVAN Rodica, Ingénieur logiciel<br>SEILER Julien, Ingénieur logiciel |
| **Examinateur :** | Julien MALIQUE, ADAE |
| **Diffusion :** | Comité de pilotage |

## Diffusion du document

| Nom | Organisme | Pour information | Pour action | Pour validation |
|---|---|---|---|---|
| Comité de pilotage | ADAE | | | X |
| Benjamin GASSMANN | ACTIMAGE | | X | |
| Guillaume DREYER | ACTIMAGE | X | | |
| Vincent GASS | ACTIMAGE | | X | |
| Christophe MEGEL | ACTIMAGE | X | | |
| Lydia CHERRIER | ACTIMAGE | X | | |
| Equipe Projet | ACTIMAGE | X | | |

## Révision(s) du document

| Date | Révision | Nature de la révision | Auteur |
|---|---|---|---|
| 16/12/2004 | 0.1 | Création du document | Benjamin GASSMANN |
| 15/01/2005 | 0.2 | Complétion de la section Serveur | Rodica IOVAN |
| 02/02/2005 | 0.3 | Ajout de la section Sunbird | Julien SEILER |
| 11/02/2005 | 0.4 | Ajout de la section Pocket PC | Julien SEILER |
| 22/03/2005 | 0.5 | Ajout de la section OpenGroupware | Julien SEILER |

# SOMMAIRE

# SECTION 1 - INTRODUCTION

## 1.1 - OBJET DU DOCUMENT

Ce document fait suite au document de spécifications fonctionnelles réalisé conjointement par ACTIMAGE et l'ADAE ; il a pour objectif de définir d'un point de vue technique le projet « Synchronisation des Agendas ». En particulier, ce document détaillera l'architecture et l'implémentation de chacun des éléments de la solution.

## 1.2 - DOCUMENTS DE RÉFÉRENCE

| Titre du document | Référence du document |
|---|---|
| Plan Assurance Qualité | ADAE_Synchro_PAQ |
| Spécifications fonctionnelles générales | ADAE_Synchro_SFG |

## 1.3 - GLOSSAIRE

### 1.3.1 - Framework

Outil de structuration orienté objet. Bibliothèque de classes spécialisées orientées objet. Un framework fournit aux programmeurs un canevas de travail et des objets spécialisés (accès aux bases, objets métiers, etc.).

### 1.3.2 - Format ANT

ANT, à l'instar du "Makefile" en C, permet d'automatiser des tâches comme la compilation, génération de bibliothèques (fichiers Jar), la génération de la documentation, etc.

### 1.3.3 - SGBD

Système de Gestion de Base de Données. Par exemple, PostgreSQL.

### 1.3.4 - IHM

**I**nterface **H**omme – **M**achine, ensemble de dispositifs matériels et logiciels permettant à un utilisateur de communiquer avec un système informatique.

### 1.3.5 - API

« Application Programming Interface ». Interface de programmation d'applications, contenant un ensemble de fonctions courantes de bas niveau, bien documentées, permettant de programmer des applications de « Haut Niveau » .

### 1.3.6 - XPCOM

XPCOM (Cross (X) Platform Component Object Module) est un framework permettant de séparer un projet d'application monolithique en éléments modulaires de

plus petite taille. Ces pièces, appelées « composant » sont re-assemblées au moment de l'exécution de l'application.

### 1.3.7 - XUL

Le XUL (XML User Interface Language) est un langage balisé permettant de créer des interfaces utilisateur riche et dynamique. Ce langage est intégré à Gecko qui est le moteur de présentation graphique utilisé par le projet Mozilla (navigateur et autre application).
Les interface graphique décrites en XUL sont généralement associé à des fonctions de gestion des commandes (par exemple, « clique sur un bouton ») écrites en Javascript.

# SECTION 2 - GENERALITES SUR LA SYNCHRONISATION

Tous les appareils mobiles (Pocket PC, PC portables, téléphones) engendrent un besoin de synchroniser leurs données avec un serveur où l'information serait stockée et centralisée.
Cette possibilité d'accéder et de mettre à jour « à la volée » ces diverses sources d'information est un facteur clé de la pérennité de ces réseaux d'appareils mobiles.

La synchronisation, ce process qui consiste à rendre deux sources de données identiques, introduit des concepts techniques clés, qu'il convient de brièvement clarifier :

- Manipulation d'identifiant (ID Handling)
- Détection de changement
- Commandes de modification
- Synchronisation lente et rapide (« Slow et fast synchronisation)

## 2.1 - MANIPULATION D'IDENTIFIANT

La manipulation d'identifiants de données est un point important dans le domaine de la synchronisation.

Traditionnellement, au sein des systèmes d'informations, un élément de données (ou enregistrement) possède une propriété permettant de l'identifier de façon non ambiguë. Par exemple, une commande est caractérisée par un numéro ou une référence unique.
Pour une application d'entreprise, les données sont stockées sur une base de données centralisée, ce qui garantit la validité de cet identifiant. L'enregistrement correspondant est connu de tout le système grâce à cette propriété, alors nommée « identifiant unique global » (Global Unique ID, GUID).

Au niveau d'un système de synchronisation, ce modèle peut être remis en question, puisque chaque device utilise bien souvent son propre identifiant, dans la mesure ou chaque appareil est implémenté sans souci de synchronisation.

Afin de faire correspondre les enregistrements des clients et du serveur, il est donc indispensable d'effectuer une mise en correspondance (« mapping ») pour garantir la cohérence des données. Le serveur attribue un GUID à chaque enregistrement, et établie une table de correspondance avec les identifiants de chaque client (LUID, « Local Unique Identifiant » pour identifiant unique local).

## 2.2 - DETECTION DE CHANGEMENT

Une détection de changement au niveau d'un ensemble d'enregistrements (en l'occurrence les rendez-vous dans un agenda) s'effectue à deux niveaux distincts :

- Au niveau de l'ensemble des enregistrements : recherche d'enregistrements nouveaux ou supprimés

- Au niveau de chaque enregistrement : comparaison entre la date de dernière synchronisation et la date de dernière modification (« last update ») de l'enregistrement en question.

## 2.3 - COMMANDES DE MODIFICATION

La façon dont les modifications seront échangées entre les clients et le serveur est un point clé du système de synchronisation ; les opérations nécessaires et suffisantes que doivent implémenter le protocole de synchronisation sont au nombre de trois :

- L'ajout d'enregistrement
- La suppression d'enregistrement
- La modification d'enregistrement.

## 2.4 - SYNCHRONISATION « LENTE » ET « RAPIDE »

La synchronisation peut se faire selon deux modes distincts, le mode dit « lent » et celui dit « rapide ».

Le second n'implique que les modifications ayant eu lieu depuis la dernière synchronisation réussie entre deux devices ; bien sûr, ce type de traitement est optimisé, et c'est celui qui a lieu dans la majorité des cas.

Cependant, il arrive dans certains cas que la synchronisation dite « rapide » ne soit pas possible, par exemple :

- Lors de la première synchronisation d'un device.
- Après l'échec d'une synchronisation rapide
- Après la réinitialisation d'un appareil mobile
- etc.

Alors, une synchronisation dite « lente » sera effectuée. Le device envoie au serveur la totalité de ses enregistrements, et ce dernier lui retournera les commandes de modifications nécessaires à la mise à jour de sa base d'enregistrements locale.

Indépendamment des deux modes principaux (« lent » et « rapide ») il existe plusieurs une autre classification des modes de synchronisation :

- Client à serveur : le serveur modifie sa base d'enregistrement à partir de celle des clients, ne retourne aucune commande de modification
- Serveur à client : les clients mettent à jour leur base d'enregistrements à partir de celle du serveur, mais n'envoient aucune commande de modification
- « Two-way » : les client et le serveur s'échange leurs commandes de modifications, afin de synchroniser ensemble leurs bases

Dans le cadre de ce projet, seulement la troisième technique sera utilisée.

# SECTION 3 - LE PROTOCOLE DE SYNCHRONISATION

Avec la quantité grandissante d'appareils mobiles et la dispersion des données qui en résulte, la nécessité d'une standardisation dans le domaine de la synchronisation est devenue évidente.

C'est alors que la norme SyncML est apparue (Synchronization Markup Language) ; à présent, les travaux autour de ce standard se font sous tutelle de l'OMA (Open Mobile Alliance).

Le standard SyncML présente les caractéristiques suivantes :

- Initiative d'origine industrielle visant à développer et promouvoir un protocole simple et commun de synchronisation de données applicable à l'échelle industrielle
- Spécification pour un framework de synchronisation de données
- Format basé sur XML
- Protocole transportant des commandes de modification (cf. §2).

SyncML définit comment établir, procéder et terminer une session de synchronisation de données. Il précise également comment échanger des modifications de données et les commandes à utiliser.
Par contre, il ne définit pas comment détecter et résoudre les conflits. Cette tâche revient aux développeurs / concepteurs d'applications compatibles SyncML.

SyncML permet de synchroniser presque n'importe quel type de données, par exemple :

- Format de données personnelles (vCard, vCalendar, vTodo, etc.)
- Emails, network news
- Données relationnelles
- Documents XML et HTML
- Données binaires

Pour faciliter l'adoption de ce standard, l'initiative SyncML fournit :

- Une spécification d'architecture
- Deux spécifications de protocole (SyncML representation protocol et SyncML synchronization protocol)
- Connexion avec les protocoles de transport standard (HTTP, WSP, OBEX)
- Un prototype ouvert d'implémentation.

# SECTION 4 - SERVEUR DE SYNCHRONISATION

## 4.1 - SYNC4JSERVER

La solution proposée se base sur la solution Sync4j, et notamment le produit Sync4jServer. Ce produit fournit un serveur SyncML open source (licence GPL) stable, implémenté en JAVA, bien architecturé et surtout extensible par des modules additionnels à développer.

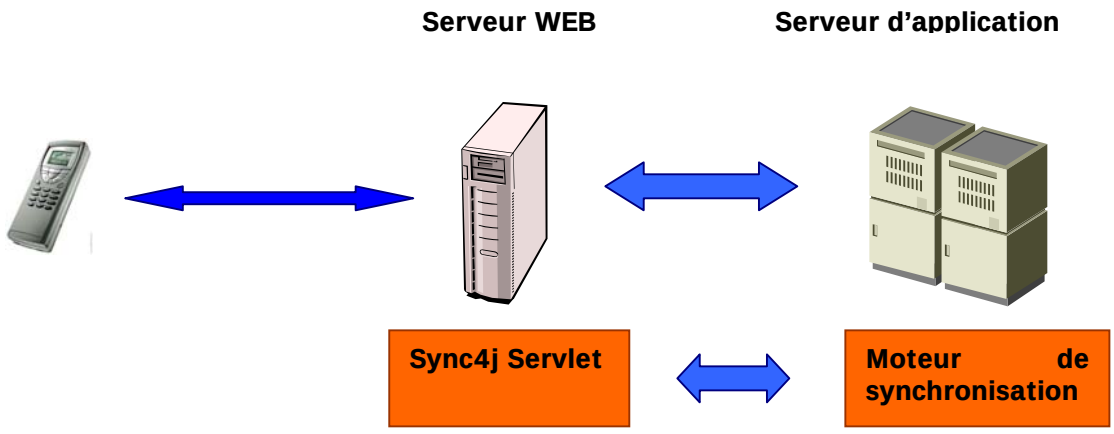
### 4.1.1 - Modules de base

Voici la liste des modules de base du serveur Sync4j, qui permettent une synchronisation conformément à la norme SyncML :

- Le module « moteur Sync4j », extensible par des modules additionnels
- Le module « Transport Layer », qui reçoit et envoie du flux contenant des messages SyncML (par exemple via le protocole http)
- Le module « SyncML », qui encode et décode les messages SyncML conformément aux spécifications
- Le module « protocole », qui implémente le protocole de synchronisation SyncML, en décrivant comment les messages doivent s'enchaîner pour représenter une session de synchronisation complète
- Le module « services », fournissant diverses fonctionnalités transversales comme l'authentification, la sécurité, la configuration, le traçage, etc.
- Les « SyncSources », permettant au serveur Sync4j de s'intégrer avec des données extérieures.

### 4.1.2 - Architecture du système

L'architecture système de ce serveur de synchronisation est représenté sur la figure suivante :

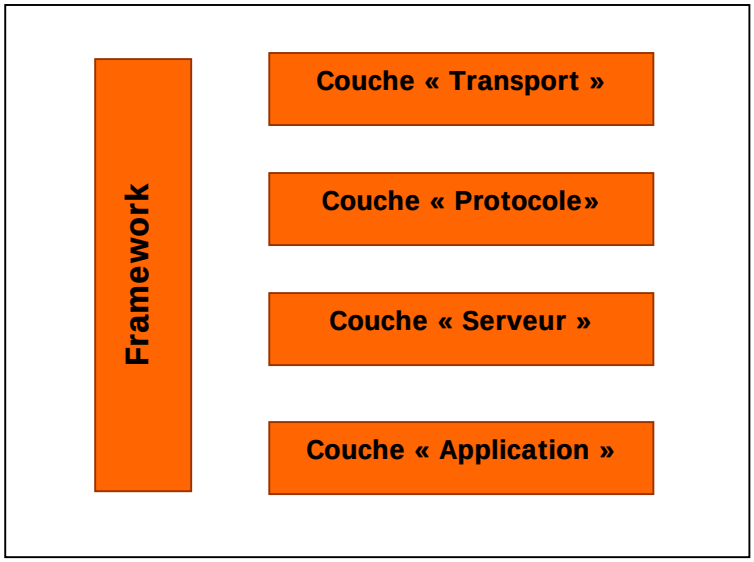


Ce schéma met en évidence la séparation entre le couche « transport » et la couche métier du système.
La partie « transport » est implémentée grâce à un servlet JAVA, et fonctionne via un conteneur web compatible J2EE, tandis que la partie métier est incluse dans un EJB (*Enterprise Java Bean*) déployé via un conteneur d'EJB (serveur d'application).

Dans la majorité des cas, ces couches sont réunies sur le même système (et la même machine).

### 4.1.3 - Vue d'ensemble de l'architecture du serveur

Le schéma ci-dessous illustre l'architecture en couches (layers) du serveur Sync4jServer :




Par couche logicielle, on entend un ensemble de fonctionnalités bien définies et cloisonnées, accessible uniquement via une interface de communication.

Sur le diagramme précédent, on distingue les couches suivantes :

- Transport (qui gère la partie HTTP)
- Protocole (qui gère l'encodage/décodage SyncML)
- Serveur (qui gère la synchronisation)
- Application (qui gère les liens extérieurs).

La couche Transport constitue la porte d'entrée des messages en provenance des clients. La version actuelle du SyncServer de Sync4j implémente le protocole de transport HTTP. Cependant, l'architecture du système permet l'intégration d'autres protocoles de transport.

La couche Protocole gère la manipulation des messages SyncML, encapsulé dans le flux HTTP. A l'instar de la couche transport, on peut imaginer l'implémentation d'un autre protocole de synchronisation au niveau de cette couche.

La couche Serveur est chargé de la synchronisation à proprement parler ; le moteur de synchronisation, implémenté en JAVA, s'enveloppe facilement dans une application J2EE (Java 2 Enterprise Edition) grâce à un EJB déployable sur un serveur d'application compatible J2EE.

La couche Application décrit la façon dont le serveur de synchronisation interagit avec le monde extérieur. Aussi, cette couche implémente un framework dans le but de satisfaire tout besoin applicatif spécifique.

Le framework fournit des services et abstractions utilisés par les différentes couches. Les principales fournitures sont :

- Implémentation de la représentation et du protocole SyncML de base
- Configuration
- Traçage applicatif (« *logging* »)
- Outils de synchronisation de données
- Sécurité
- Outils standards.

Ces services sont implémentés respectivement dans les paquetages java suivants :

- *sync4.framework.core*
- *sync4.framework.config*
- *sync4.framework.logging*
- *sync4.framework.engine*
- *sync4.framework.security*
- *sync4.framework.server*

## 4.2 - CONCEPTS SYNC4J

### 4.2.1 - Sync4j Module

Le concept de « module » pour l'application Sync4jServer représente le moyen d'étendre les fonctionnalités de celle-ci ; un module contient les éléments nécessaires à l'activité et au déploiement de la (ou les) nouvelle(s) fonctionnalité(s) implémentée(s).

Ces éléments sont :

- Les classes implémentant la partie métier
- Le(s) script(s) d'initialisation
- Le(s) fichier(s) de configuration
- Le(s) script(s) SQL si nécessaire.

Le tout est packagé en un seul et unique fichier, structurellement comparable à une archive JAVA (fichier *.jar*), ayant pour extension *.s4j* (Sync4j).

La structure générique d'une telle archive est la suivante :

- lib
  - *nom_du_module.jar*
  - *librairie_dépendante_1.jar*
  - *librairie_dépendante_2.jar*
  - ...
- config
  - *config.properties*
  - *MySyncSource .xml*
  - ...
- exclude
  - *manifest.mf*

- install
  - *install.xml*
- sql
  - postgresql
    - *create_schema.ddl*
    - *drop_schema.ddl*
    - *init_schema.sql*

Le répertoire « lib » contient les classes métiers de l'extension, packagée sous forme d'archive java, plus éventuellement des librairies addtionnelles (*librairie_dépendate_X.jar*).

Le répertoire « config » contient un fichier de configuration générale du module (*config.properties*) ainsi que d'éventuels fichiers de configuration pour les classes métiers.

Le répertoire « install » qui contient un script au format Ant ; ce script est appelé lors de l'installation du module.

Le répertoire « sql » contient, classés par SGBD, les scripts de modification du schéma de la base de données, et d'initialisation, si le module nécessite de telles interventions.

Enfin, le répertoire « exclude » contient les fichiers nécessaires lors de la procédure d'installation, mais qui ne seront pas inclues dans l'archive finale (fichier *.ear*).

#### 4.2.2 - SyncConnector

Un SyncConnector est une extension permettant la synchronisation de Sync4jServer avec une source de données extérieure. La version standard de Sync4j intègre un SyncConnector pour « FileSytem », c'est-à-dire que la source de données standard est le système de fichier de la machine hébergeant Sync4jServer.

#### 4.2.3 - SyncSourceType

Un SyncSourceType sert au typage des SyncSources.

#### 4.2.4 - SyncSource

Une SyncSource constitue une unité minimale de synchronisation ; elle représente une entité que le client pourra synchroniser. Elle est identifiée uniquement au sein du serveur par un paramètre nommé « SourceURI », qui constitue la clé d'accès que devra utiliser le client pour y accéder.

### 4.3 - DEVELOPPEMENT AUTOUR DE SYNC4J DANS LE CADRE DU PROJET

#### 4.3.1 - Authentification

L'authentification des utilisateurs sur le serveur **sync4j** est basée sur l'architecture **JAAS** *(Java Authentication and Authorization Service)* fournie avec la version JDK > 1.4. L'utilisation du module **JAAS** sera utilisée et précisée dans le fichier *sync4j.properties* (configuration générale de l'application) :

*security.officer=sync4j/server/security/JAASOfficer.xml*

Ainsi la classe **JAASOfficer** du framework (*sync4j.framework.security.JAASOfficer*) va déléguer l'authentification au module **JAAS** par l'intermédiaire d'un objet *LoginContext*.

Cet objet interroge les fichiers de configuration (dans notre cas *serverlogin.config*) sur la technologie qui va être utilisée.

Krb5LoginModule est basé sur le protocole **Kerberos**, le protocole de sécurité principal pour les authentifications réalisées dans un domaine Windows 2000. Il permet aux utilisateurs d'accéder aux ressources réseau à partir de la même ouverture de session. Pour confirmer son identité, le client va demander au service Kerberos du réseau un ticket qui lui permettra de confirmer son identité. Une fois le ticket reçu, l'utilisateur pourra accéder au service réseau voulu.

Le service Kerberos s'appelle le KDC (Key Distribution Center) et se trouve dans chaque contrôleur de domaine, qui stocke toutes les informations relatives aux comptes.

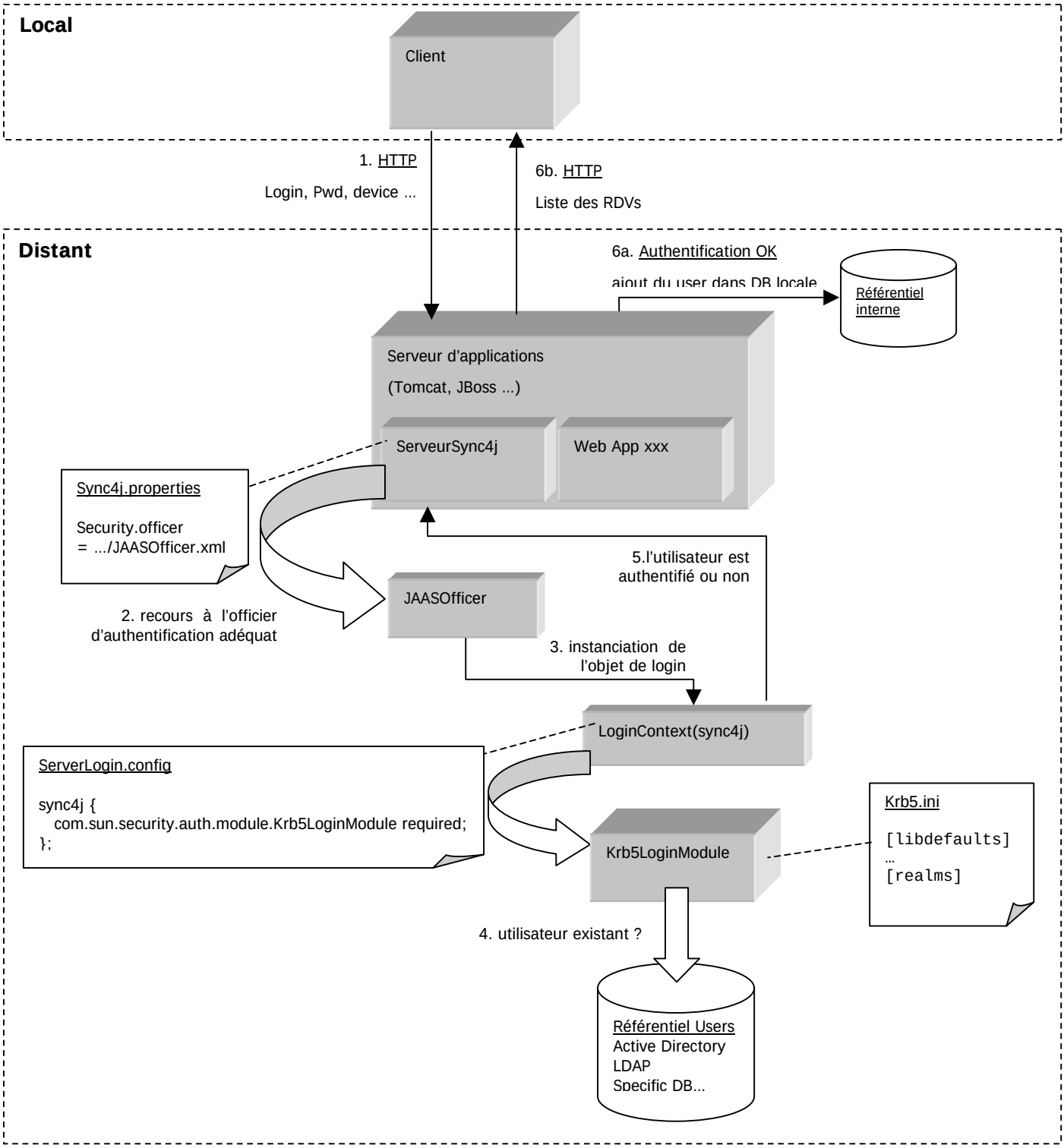
Le ticket émis au client par un KDC s'appelle un TGT (Ticket Granting Ticket). Le TGT permet au système client d'accéder au TGS (Ticket Granting Service) se trouvant dans le contrôleur de domaine. Ce TGS permettra l'émission d'un ticket de service au client. C'est ce ticket de service que le client présentera au service réseau demandé.

Dans le cadre de l'intégration pour l'ADAE le module **Krb5LoginModule** offre le grand avantage de pouvoir utiliser l'Active Directory de l'entreprise comme unique référentiel utilisateur. Cela permettra aux utilisateurs d'avoir accès au service de synchronisation avec leur mot de passe réseau habituel et non avec un nouveau mot de passe dédié à la synchronisation (autrement dit on évite ainsi la réplication ou la création d'une nouvelle base de données stockant les logins/passwords au niveau du serveur).

Néanmoins, il faudra tout de même conserver une base locale au serveur qui contient les associations *device/utilisateurs/principals*, afin de référencer la liste des rendez-vous correspondant.

Cette réplication pourra se faire soit « à la volée » lors de la première synchronisation d'un user/*device*, soit en intégrant dans la console d'administration une fonctionnalité « import utilisateurs » depuis l'Active Directory et d'association « à la main » de ces utilisateurs aux *devices* existants.

Processus d'authentification :



### 4.3.2 - Le module SGBD

La solution basique de Sync4J serveur consistant à stocker les données sous forme de systèmes de fichiers (fournie par la classe *sync4j.foundation.engine.source.FileSystemSyncSource*) n'intègre pas le scope du projet.

Le travail de sur développement à effectuer va donc être l'implémentation du module (cf. paragraphe précédent) permettant le stockage de l'information (en l'occurrence des rendez-vous au format vCalendar) sur un SGBD, en particulier PostgreSQL.

Ce module sera nommé : ***sgbd -1.0.s4j***.

Les fonctionnalités principales offertes par ce module sont

- la création d'une source de synchronisation accédant une BD relationnelle
- création et initialisation de la base interne de l'application

#### a. Architecture métier du module SGBD

La partie métier du module sera contenue dans le package JAVA *sync4j.sgbd*.

Son arborescence sera la suivante :

- sync4j
  - sgbd
    - engine
      - source
        - SGBDSyncSource

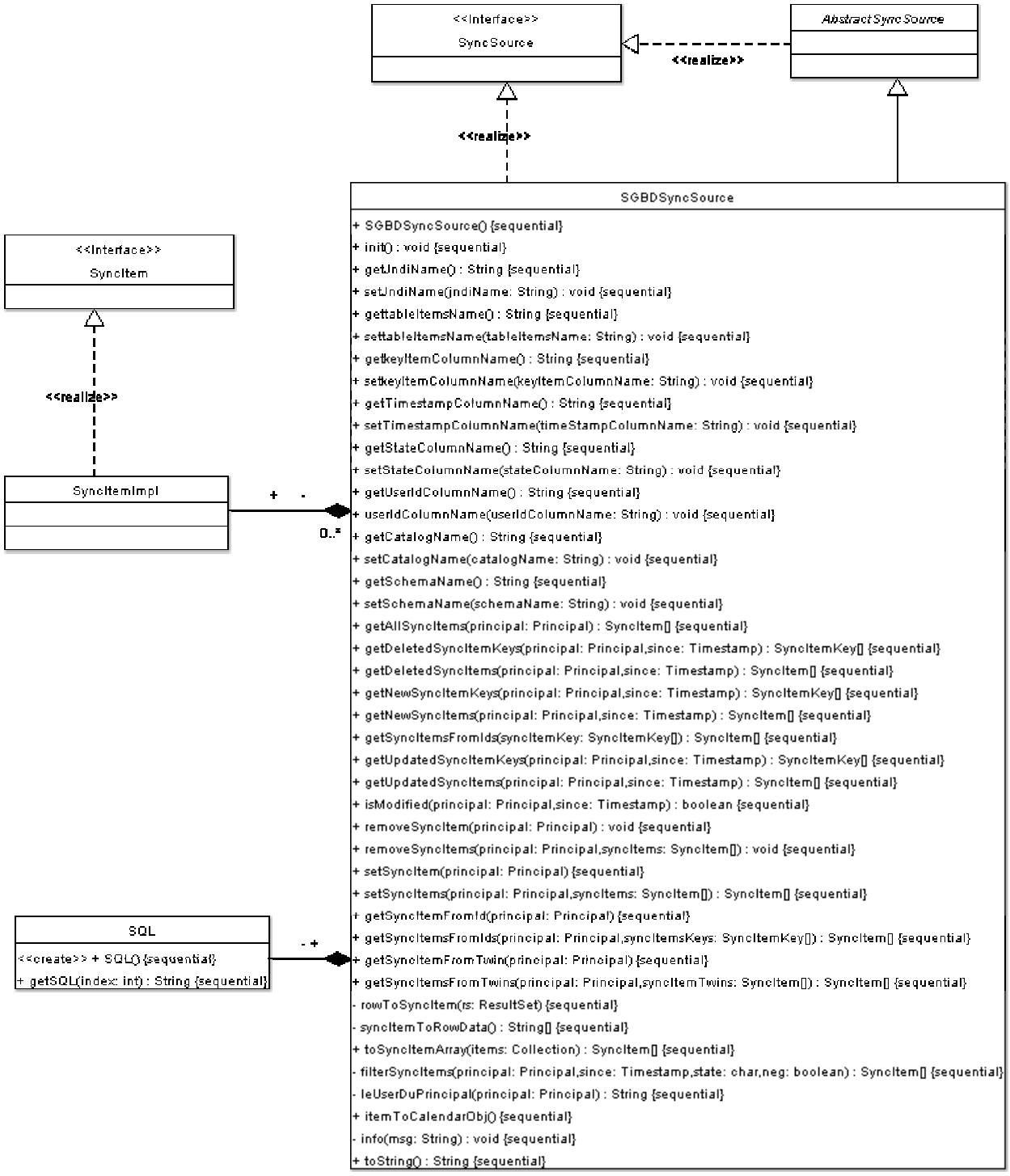
Les fonctionnalités de base de l'interface *SyncSource* sont implémentées par la classe *AbstractSyncSource*, donc la source SGBD va en hériter, en ajoutant les méthodes spécifiques au stockage des items à travers une base de données relationnelle.

Les principales méthodes de la classe *SGBDSyncSource :*

- *getUpdatedSyncItems* : cette méthode retourne une liste d'items modifiés (en l'occurrence de rendez-vous) pour un utilisateur donné, depuis une date donnée
- *getUpdatedSyncItemKeys* : cette méthode retourne les clés (identifiant unique) d'items modifiés pour un utilisateur donné, depuis une date donnée
- *getNewSyncItems* : cette méthode retourne une liste de nouveaux items pour un utilisateur donné, depuis une date donnée
- *getNewSyncItemKeys* : cette méthode retourne les clés (identifiant unique) de nouveaux items pour un utilisateur donné, depuis une date donnée
- *getDeletedSyncItems* : cette méthode retourne une liste d'items supprimés pour un utilisateur donné, depuis une date donnée
- *getNewSyncItemKeys* : cette méthode retourne les clés (identifiant unique) d'items supprimés pour un utilisateur donné, depuis une date donnée
- *getAllSyncItems* : cette méthode retourne la liste de tous les items pour un utilisateur donné, depuis une date donnée
- *setSyncItem* : ajoute ou modifie un item donné
- *setSyncItems* : ajoute ou modifie une liste d'items donnée
- *removeSyncItem* : supprime un item donné

- *removeSyncItems* : supprime une liste d'items donnée
- *getSyncItemFromTwin* : recherche le ou les item(s) représentant la même information qu'un item donné. Cette méthode est utilisée pour la détection de conflits et la synchronisation lente (les items qu'on peut considérer équivalents).

Ci-dessus un diagramme de classe représentant cette architecture orientée objet :



### b. Impact de l'ajout du module SGBD sur la base de données interne sync4j

Une nouvelle table **sync4j_items**, qui va rassembler les items résultants des synchronisations effectuées par les différents clients, va être intégrée dans la base interne du serveur.

| **sync4j_items** | |
|---|---|
| **ID_ITEM** | **VARCHAR(50)** |
| USER_ID | VARCHAR(92) |
| LAST_UPDATE | TIMESTAMP(14) |
| STATE | VARCHAR(40) |
| DATA_TYPE | VARCHAR(25) |
| DATA_ID | VARCHAR(25) |
| DATA | TEXT |

Certains champs ne décrivent pas le calendrier mais sont utilisés par le système :

- ➔ ID_ITEM représente la clé qui identifie un seul rdv
- ➔ UDER_ID représente le "propriétaire" du rdv (la personne qui peut accéder à ce rdv)
- ➔ Les champs **STATE** et **LAST_UPDATE** sont utilisés pour tracer les modification du rdv (informations récurrentes dans le processus de synchronisation)
- ➔ (DATA_TYPE, DATA_ID)( ?) a utiliser éventuellement pour faire référence a une autre table pour récupérer le contenu (ex : ('calendar', *<l'id_du_calendar>*), ('contact', *<l'id_du_contact>*) …)

La connexion à la base de données se fera par l'intermèdiaire des settings du projet (le contexte *jdbc* du serveur sync4j).

**La création et la configuration de la SGBDSyncSource**

Le serveur utilisera les données à synchroniser de la table *sync4j_items* si une demande d'un client pour cette source est faite (via son nom, *sourceURI* sur le client). Pour cela le serveur doit être « informé » de l'existence de cette nouvelle source. Le script *init.sql* va initialiser les tables

### 4.3.3 - La stratégie de synchronisation

La stratégie par défaut implémentée dans Sync4j prévoit une gestion des conflits stricts donnant la priorité au serveur.
Afin d'assouplir la gestion des conflits, une nouvelle stratégie doit être développée permettant de fusionner des items en conflits lorsque cela est possible ou de les dupliquer. Ces résolutions concernent essentiellement le cas de mises à jour concurrentes d'un même item par deux clients différents.
Prenons un exemple permettant d'illustrer simplement cette nouvelle stratégie :
On considère que deux agendas A et B sont proprement synchronisés et affiche chacun un unique rendez-vous [Réunion : Mercredi : Paris].
Sur l'agenda A, l'utilisateur modifie le rendez-vous en [Réunion : Mardi : Paris] et plus tard, sur l'agenda B fait la modification [Réunion : Mercredi : Strasbourg].
Lorsque ces deux agendas vont être synchronisés, un conflit va apparaître puisque deux modifications concurrentes ont été effectuées.
Puisque les modifications concernent des champs différents dans le rendez-vous, une fusion des items modifiés est possible et le serveur devra retourner l'item [Réunion : Mardi : Strasbourg].
Si en revanche ces modifications concurrentes avaient concerné des champs identiques, le rendez-vous aurait dû être dupliqué afin que chaque agenda présente les deux

versions. Cette solution permet d'avertir l'utilisateur de l'apparition d'un conflit ne pouvant pas être résolu.

La mise en place de cete stratégie nécessite la modification de la base de données du serveur afin qu'il puisse garder une trace des modifications effectuées par les différentes sources sur un même item et déterminer les possibilités de fusion ou duplication.

En effet, lorsque le serveur reçoit un item de mise à jour, il doit disposer d'un moyen lui permettant de déterminer quels sont les champs ayant été modifiés. Pour cela, il est donc nécessaire de conserver la version actuelle de chaque client pour chaque item.
Pour cela, nous devons ajouter un identifiant unique supplémentaire pour chaque version di'item dans la table sync4j_items et ajouter une nouvelle table sync4j_items_ref permettant d'associer une version d'un item à un ou plusieurs principals.

Reprenons l'exemple précédent en observant l'évolution de la base de données.

- Initialement chaque agenda A et B est proprement synchronisé et présente un seul rendez-vous

| ***sync4j_items*** | | | |
|---|---|---|---|
| **GUID** | **Version** | **Date** | **Data** |
| 1234 | 1 | T1 | [Réunion : Mercredi : Paris] |

| ***sync4j_items_ref*** | |
|---|---|
| Principal | Version |
| A | 1 |
| B | 1 |

- L'utilisateur modifie le rendez-vous en [Réunion : Mercredi : Paris] sur l'agenda A puis le synchronise. Ceci a pour effet de créer une nouvelle version de l'item. La version 1 doit être conservée car elle est encore la version de référence pour le principal B.

| ***sync4j_items*** | | | |
|---|---|---|---|
| **GUID** | **Version** | **Date** | **Data** |
| 1234 | 1 | T1 | [Réunion : Mercredi : Paris] |
| 1234 | 2 | T2 | [Réunion : Mardi : Paris] |

| ***sync4j_items_ref*** | |
|---|---|
| Principal | Version |
| A | 2 |
| B | 1 |

- Puis il modifie le rendez-vous en [Réunion : Mercredi : Strasbourg] sur l'agenda B et synchronise ce dernier. Au moment de la synchronisation, la mise à jour est comparée avec l'item de référence (version 1) et la fusion est automatiquement réalisée avec la dernière version présente sur le serveur (version 2). Plus aucun principal ne fait alors référence à la version 1. Elle est donc supprimée, et une nouvelle version de l'item est créée contenant les données fusionnées.

| ***sync4j_items*** | | | |
|---|---|---|---|
| **GUID** | **Version** | **Date** | **Data** |
| 1234 | 2 | T2 | [Réunion : Mardi : Paris] |
| 1234 | 3 | T3 | [Réunion : Mardi : Strasbourg] |

| ***sync4j_items_ref*** | |
|---|---|
| Principal | Version |

| | |
|---|---|
| A | 2 |
| B | 3 |

- Enfin, lorsque l'agenda A sera à nouveau synchronisé, il recevra automatiquement la dernière version de l'item, donc la version fusionnée (3). La version 2 n'étant plus utile, elle sera supprimée.

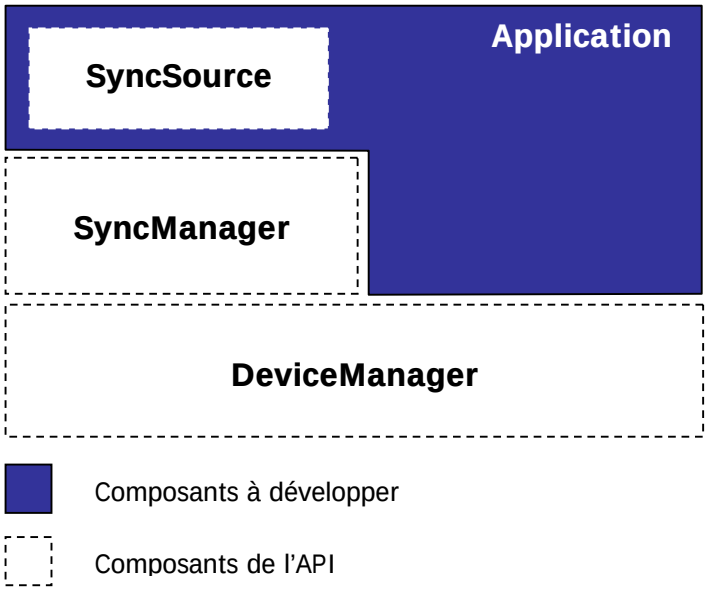
| ***sync4j_items*** | | | |
|---|---|---|---|
| **GUID** | **Version** | **Date** | **Data** |
| 1234 | 3 | T3 | [Réunion : Mardi : Strasbourg] |

| ***sync4j_items_ref*** | |
|---|---|
| Principal | Version |
| A | 3 |
| B | 3 |

La mise en place de cette méthode de résolution peut-être intégrée à la stratégie actuelle de Sync4j en redéfinissant la méthode de résolution du conflit de type UPDATE_UPDATE. Une méthode tryMerging() permettra pour chaque couple d'items en conflit de tenter la fusion. Si la fusion est impossible, l'item est simplement dupliqué.

# SECTION 5 - L'API SYNCCLIENT C++ DE SYNC4J

Le schéma ci-dessous décrit l'architecture générale d'une application basée sur cette API :




Cette architecture est composée de deux types d'éléments :

- les éléments spécifiques à l'application devant être développés
- les éléments fournis par l'API

La tâche principale de la brique métier s'insère entre l'objet SyncSource et l'application Sunbird. Il s'agit de fournir les méthodes qui permettront à l'application de pouvoir configurer un objet SyncSource puis d'initier une synchronisation par l'intermédiaire du SyncManager.

L'objet SyncSource joue le rôle d'interface, neutre de format, entre l'application et les fonctions de synchronisation disponible dans l'API. Un objet SyncSource peut-être décrit comme une collection d'objets SyncItem répartis en plusieurs catégories :

- éléments ajoutés
- éléments modifiés
- éléments supprimés

L'objet SyncItem peut contenir n'importe quel type de données sous la forme d'une chaîne de caractères. Le format de donnée choisi est spécifié par son type MIME. Il peut s'agir par exemple d'une description d'un rendez-vous au format vCalendar. Un objet SyncItem possède également une clé permettant de l'identifier de manière unique dans le contexte de l'application.
Le composant à développer devra donc permettre d'instancier un objet SyncSource et de le remplir en fonction des données modifiées dans Sunbird.

Parmi les éléments fournis par l'API on trouve le SyncManager et le DeviceManager.
Le SyncManager offre un niveau d'abstraction par rapport à la logique chargé de réaliser le protocole de synchronisation. Il fournit donc les méthodes permettant d'initialiser une synchronisation à partir des données collectées dans un objet de type SyncSource.

La couche DeviceManager rassemble les méthodes d'accès aux couches matérielles du système hôte de l'application. Il est donc spécifique à la plate-forme utilisée (Windows Mobile, Windows 2000 etc...).

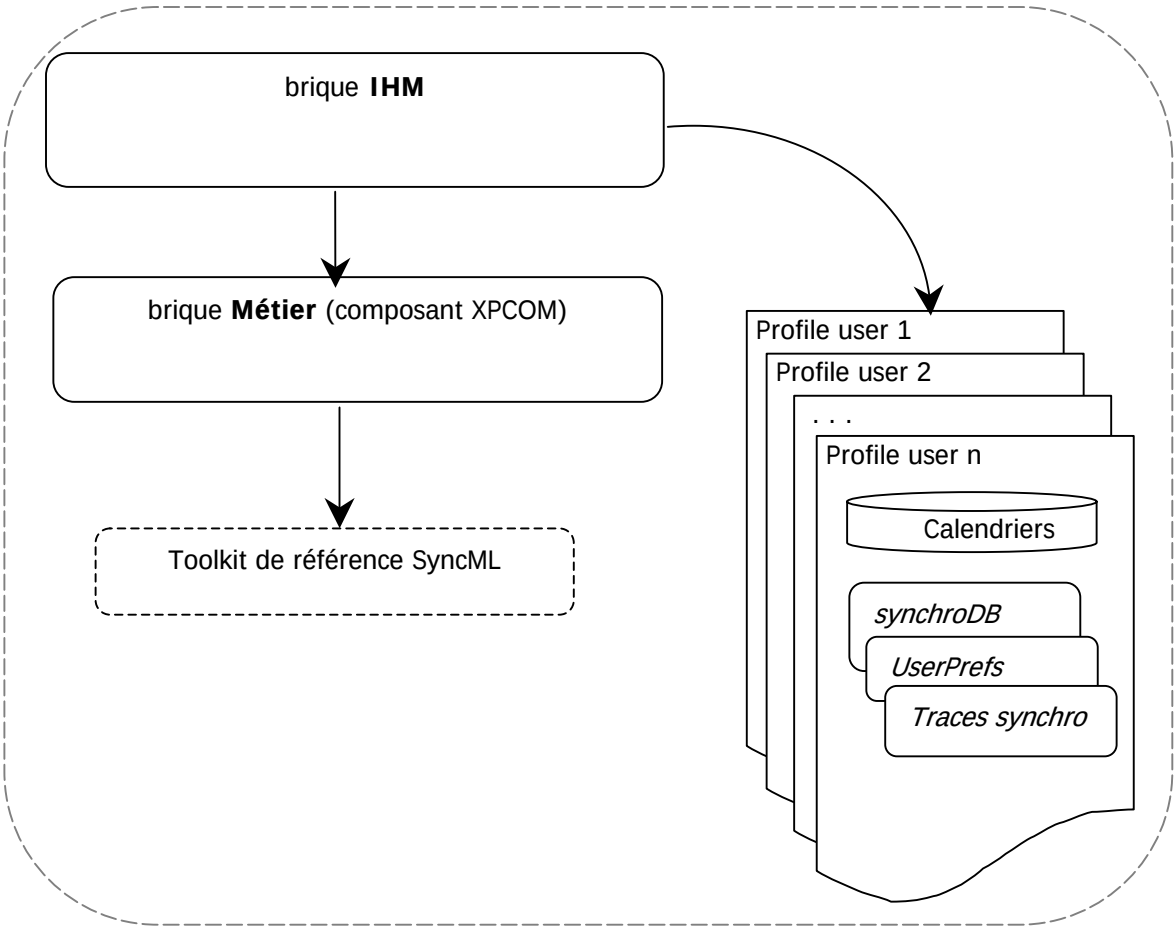
# SECTION 6 - CLIENT DE SYNCHRONISATION POUR SUNBIRD

## 6.1 - GENERALITES

Le client de synchronisation SyncML pour Sunbird sera principalement développé en C++ et JavaScript.
Il se présentera sous la forme d'un package à intégrer dans Sunbird (.xpi)
Ce package peut-être décomposé en deux briques fondamentales :

- **Métier** : module s'interposant entre le serveur SyncML et le calendrier Sunbird. Elle se présentera sous la forme d'un **composant XPCOM** (librairie C++) utilisant **le toolkit de référence** fourni par l'OMA (l'organisme initiateur du protocole SyncML)
- **IHM** : module permettant l'intégration du composant à Sunbird par l'intermédiaire des technologies XUL et Javascript.

L'IHM utilise les fonctions mises à disposition par la brique Métier et le calendrier Sunbird pour réaliser la synchronisation. Ses principaux rôles sont de fournir a l'application cliente c++ les RDVs de l'utilisateur courant récupérés dans la base locale (stockés par Sunbird dans un répertoire **Profile** spécifique a chaque utilisateur) et de la remettre à jour avec les modifications envoyées par le serveur de synchronisation.



## 6.2 - LA BRIQUE METIER

Le module Métier, développé en C++ à l'aide des outils de développement Mozilla (gecko-sdk), fournit les services de synchronisation des listes des éléments à synchroniser de l'IHM.

La tâche principale de la brique métier s'insère entre l'objet SyncSource et l'application Sunbird. Il s'agit de fournir les méthodes qui permettront à l'application de pouvoir configurer un objet SyncSource puis d'initialiser une synchronisation par l'intermédiaire du SyncManager.

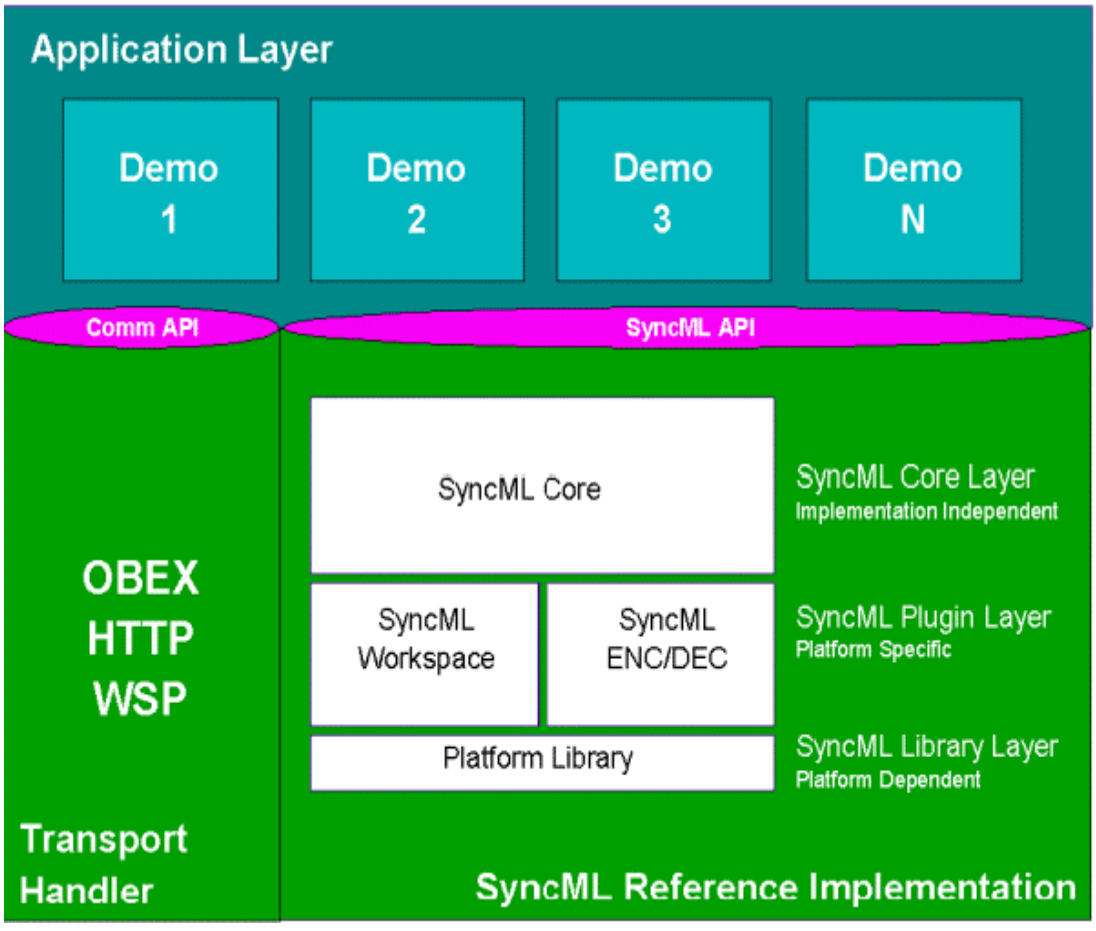
### 6.2.1 - Le Toolkit SyncML

Le toolkit SyncML est développé en C, portable sur les plateformes Win32, Palm OS et Linux. Il fournit principalement des fonctions qui gèrent les messages SyncML (encodage, décodage … ), l'échange de documents XML entre le client et le serveur de synchronisation et des librairies spécifiques à chaque plateforme (gestion de la mémoire, des chaînes des caractères etc.).

Le toolkit impose a l'application qui l'utilise, d'implémenter des fonctions de « callback », dont le rôle est de traiter les réponses du serveur. Le toolkit peut utiliser ensuite ces fonctions par l'intermédiaire des pointeurs des fonctions que le client lui transmette.

La construction et le traitement des messages SyncML sont réalisés au tour d'un « workspace » qui contient le document SyncML à chaque étape de la synchronisation.

L'architecture d'une application utilisant le toolkit :

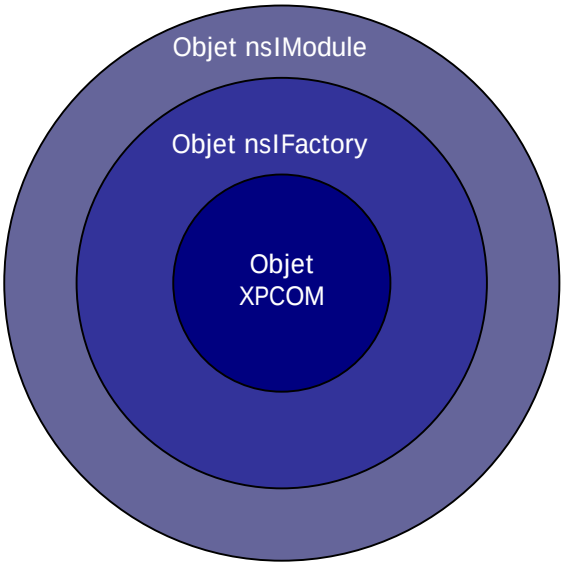


Typiquement une application utilisant le toolkit réalise une synchronisation en passant par les étapes suivantes :

- **Initialisation** : création d'une instance du toolkit et d'un workspace contenant les donnés échangées
- **Construction** des messages SyncML a envoyer au serveur et appel des fonctions du toolkit pour les rajouter au « workspace »
- **Echange** du contenu du « workspace » (précédemment construit) en utilisant la couche transport du toolkit
- **Traitement** des messages du serveur (fonctions de callback)
- **Fermeture** du processus de synchronisation (destruction instance, workspace).

### 6.2.2 - Le composant XPCOM

Un composant XPCOM est composé de trois parties principales organisées de la façon suivante :

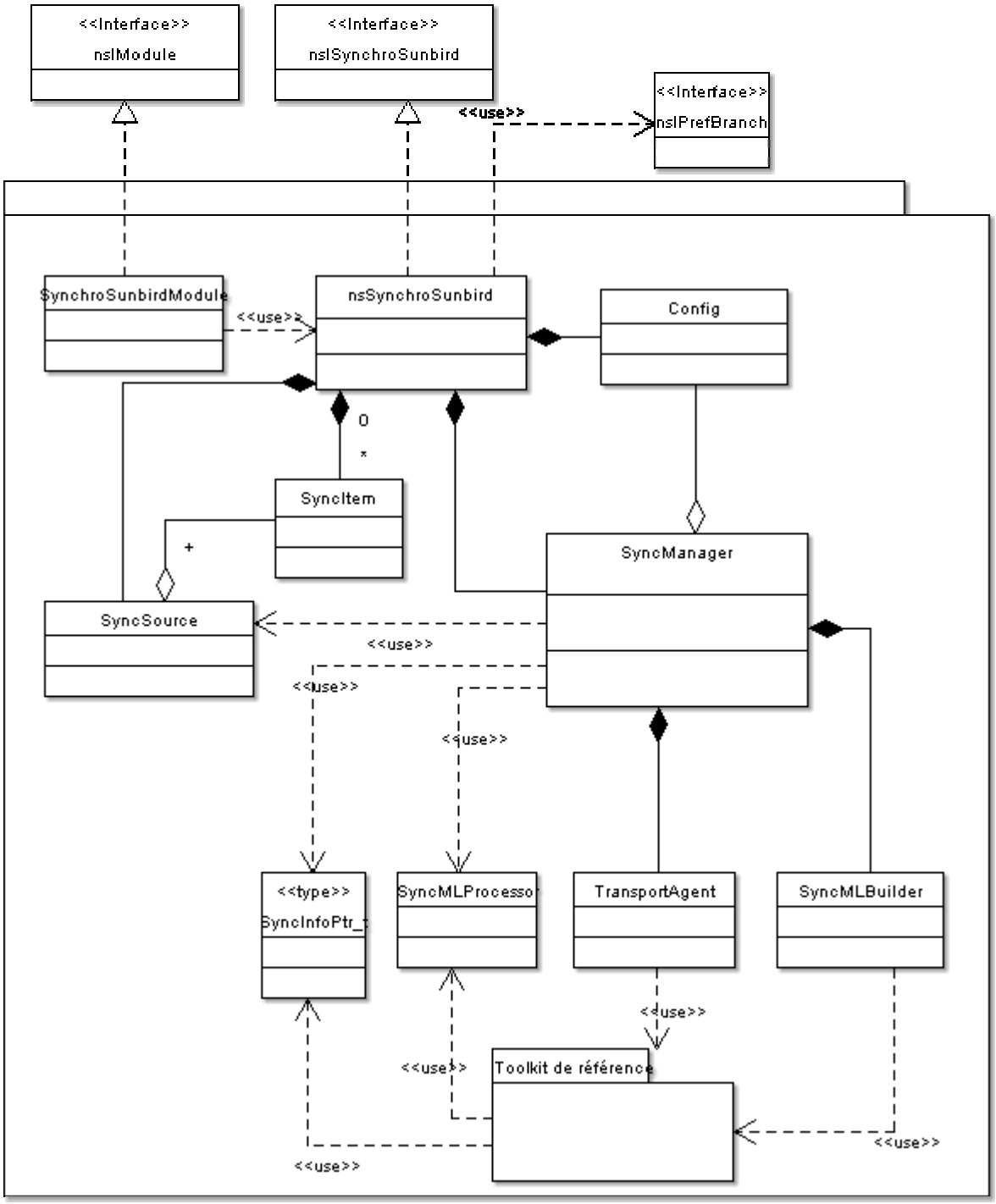



L'objet XPCOM décrit la logique métier du composant. On y retrouve donc les fonctions directement liés à la tâche de synchronisation. Les méthodes définies dans cet objet pourront être déclarées par l'intermédiaire d'une interface XPIDL afin d'être accessibles à l'ensemble des composants du système XPCOM et donc à Sunbird.

L'objet nsIFactory propose un premier niveau basique d'abstraction par rapport à l'objet XPCOM. Sa fonction principale est d'instancier une implémentation de l'objet XPCOM.

L'objet nsIModule contient les méthodes appelées lors de la déclaration ou la suppression du composant. Ces méthodes sont utilisées pour définir/supprimer les configurations de base du composant lors de son installation/désintallation.

Le composant sera donc organisé de la manière suivante :




L'interface ***nsISynchroSunbird*** permet de déclarer les méthodes de la classe nsSynchroSunbird qui seront offertes par le module. Cette interface est générée à partir d'un fichier au format **XPIDL** qui a une syntaxe proche du C++ et du Java.
De plus, les outils de développement XPCOM mettent à la disposition des développeurs des macros C++ facilitant le développement des classes SynchroSunbirdModule et nsSynchroSunbird.

Le composant possède un objet **SyncManager** qui fournit les méthodes permettant d'initialiser une synchronisation à partir des données collectées dans un objet de type SyncSource.

Un objet **SyncSource** peut-être décrit comme une collection d'objets **SyncItem** répartis en plusieurs catégories : éléments ajoutés, éléments modifiés, éléments supprimés. En fonction de l'étape de la synchronisation ces listes représentent les items du client ou les items synchronisés venus du serveur. De plus elles indiquent le mode de synchronisation (synchronisation lente, rapide …) et construisent le « mapping » des clés qui est envoyé au serveur dans la phase de finalisation de la synchronisation.

L'objet **SyncItem** peut contenir n'importe quel type de données sous la forme d'une chaîne de caractères. Le format de donnée choisi est spécifié par son type MIME. Il peut s'agir par exemple d'une description d'un rendez-vous au format vCalendar. Un objet SyncItem possède également une clé permettant de l'identifier de manière unique dans le contexte de l'application.

L'objet **Config** est créé par le composant a partir d'une interface ***nsIPrefBranch*** dans laquelle Sunbird expose les préférences utilisateur, donc aussi celle de synchronisation telles qu'elles sont remplies par l'utilisateur dans la boite des **options de synchronisation** dans la partie IHM.

La classe **SyncMLBuilder** regroupe les méthodes de construction des messages SyncML à envoyer au serveur.

Le **SyncMLProcessor** représenté par une classe statique implémente les fonctions de callback demandées par toolkit .

Le **TransportAgent** utilise les fonctions du toolkit implémentant le protocole de transport SyncML afin d'assurer la communication **http** entre le client et le serveur.

### 6.2.3 - A propos de la gestion des modifications du calendrier

D'une manière générale, la synchronisation consiste à rapporter au serveur de synchronisation les changements apparus dans le calendrier du client depuis la dernière synchronisation. Ces changements peuvent prendre plusieurs formes :

- ajout d'un nouvel élément
- modification d'un élément
- suppression d'un élément

L'objet SyncSource décrit précédemment doit contenir l'ensemble des modifications opérées sur le calendrier avant d'être synchronisé avec le serveur par l'intermédiaire du SyncManager.
Pour que ce traitement soit possible, il est nécessaire de tenir à jour un « registre » des modifications apportées sur le calendrier.

Après une synchronisation, les données de synchronisation sont sauvegardées dans un fichier dans le répertoire Profile de l'utilisateur courant. Lors d'une nouvelle synchronisation, l'objet SyncSource est construit à partir des différences observées entre ce fichier et les données du calendrier Sunbird.

## 6.3 - INTEGRATION AU NIVEAU DE L'IHM

Le composant XPCOM développé pour la synchronisation SyncML sera accompagné d'une interface utilisateur décrite au format XUL et Javascript. Cette interface sera alors intégrée dans Sunbird.
L'IHM sera composé de plusieurs fichiers XUL et Javascript décrivant l'interface utilisateur du module de synchronisation de la façon suivante :

- synchro.js : fonctions de commande de la synchronisation
- synchroSetting.xul : description de la boîte de configuration
- synchroSettings.js : fonctions associées à la boîte de configuration
- synchroOverlay.xul : description des éléments ajoutés dans le menu « Tools » et « toolbox » de Sunbird
- synchroProgress.xul : affichages du déroulement de la synchronisation
- synchroTraceViewer.xul : afficheur des informations sur les synchronisations réalisées en « «background »

## 6.4 - DISTRIBUTION DU COMPOSANT

L'application sera intégrée dans Sunbird comme une extension globale, présente sous la forme d'un fichier xpi.

Le fichier xpi est une archive zippée de l'ensemble des fichiers nécessaire au fonctionnement d'un composant. Cette archive contient les éléments suivants :

- un fichier manifest pour le package au format RDF
- le module XPCOM compilé (la librairie .dll pour Windows ou .so pour Linux et l'interface XPT)
- les librairies du toolkit
- les dialogues XUL
- les fichiers JavaScript contenant les commandes appelées à partir des interfaces XUL et les préférences par défaut de l'application
- un script d'installation au format Javascript (install.js)

Le script d'installation permet de mettre en place les différents éléments du module dans l'application cible (ici Sunbird). Pour cela, il s'appui sur l'API JavaScript XPInstall. Cette API propose toutes les fonctions permettant d'enregistrer le module et d'étendre l'interface utilisateur dans l'application.

# SECTION 7 - CLIENT DE SYNCHRONISATION POUR POCKET PC

## 7.1 - GENERALITES

Le client de synchronisation SyncML pour Pocket PC sera développé en C++.
Il se présentera sous la forme d'un fichier DLL (Dynamic Linked Library) qui contiendra l'ensemble des éléments nécessaire à la synchronisation. Celui-ci pourra être déployé sur l'ordinateur de poche par le biais d'une liaison ActiveSync préalablement établie avec un système Windows sur lequel sera exécuté le programme d'installation.
L'ensemble des éléments métier et graphique du module seront compilé dans le fichier DLL :

- Les fonctions métier permettant la synchronisation utiliseront principalement l'API SyncClient C++ de Sync4j pour établir la liaison SyncML ainisi que l'API Pocket Outlook Object Model (POOM) pour accéder aux données du calendrier Outlook.
- Les fonctions liés à l'IHM du module utiliseront les librairies standard de Windows CE.

Le module sera directement intégré à l'application Pocket Calendar (mode Calendrier de Pocket Outlook) et accessible depuis le menu « Outils ».

## 7.2 - INTEGRATION DU MODULE A POCKET OUTLOOK

L'architecture du logiciel Pocket Outlook permet d'ajouter facilement un module supplémentaire accessible depuis le menu « Outils » de l'application. Pour cela il suffit d'ajouter de nouvelles entrées dans la base de registre de Windows Mobile définissant le nom de l'item à afficher dans le menu ainsi que le chemin du fichier DLL du module.
Le DLL doit implémenter et exposer la méthode `CePimCommand` pour être exécuté depuis Pocket Outlook.

## 7.3 - DEVELOPPEMENT DES FONCTIONS METIERS DU MODULE

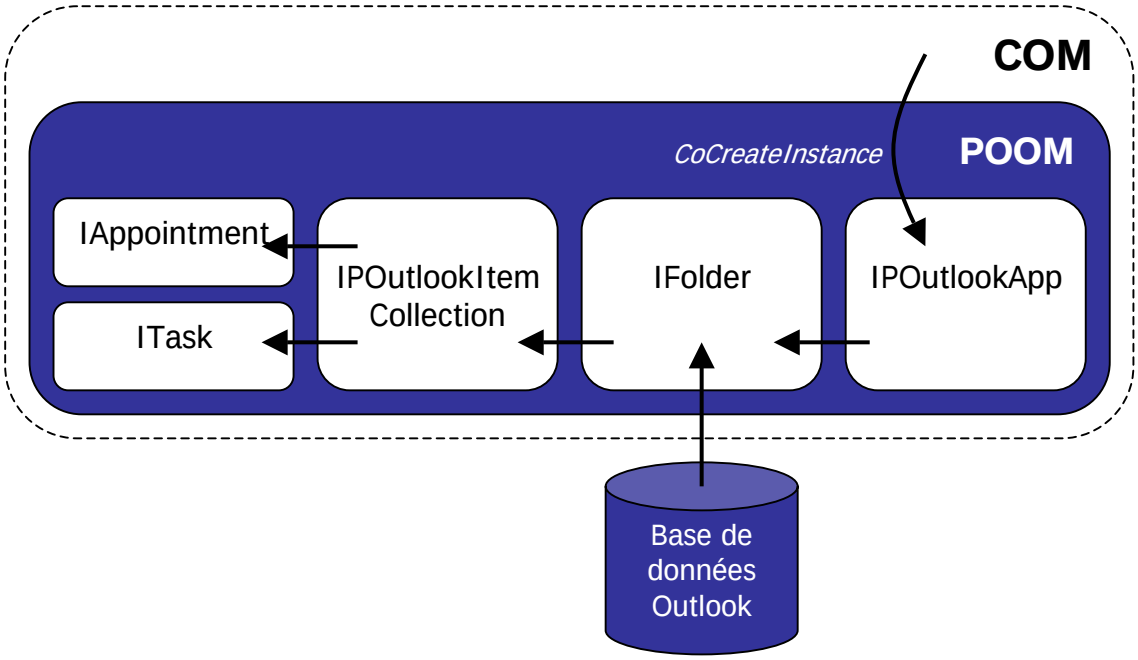
La partie métier de l'application se chargera de lire les données du calendrier, d'effectuer une requête de synchronisation auprès du serveur puis de mettre à jour le calendrier.
Pour effectuer ces tâches, le module s'appuiera sur deux API :

- L'API POOM (Pocket Outlook Object Model) fournit par Microsoft pour accéder aux données de Pocket Outlook,
- L'API SyncClient accompagnant Sync4j pour réaliser la synchronisation.

### 7.3.1 - L'API POOM

Les données manipulées par Pocket Outlook sont accessible par l'intermédiaire d'un modèle objet définit dans l'API POOM. Cette API s'intègre au modèle COM (Component Model Object) de Windows et permet donc d'accéder aux données après avoir établie une connexion avec l'application Pocket Outlook.

Le schéma suivant met en évidence les éléments permettant à l'accès aux données de Pocket Outlook.




L'objet racine de ce schéma est le IPOutlookApp. Cet objet permet d'accéder à un dossier de la base de donnée Pocket Outlook. En effet, les données sont réparties dans la base en cinq catégories, chaque catégorie étant associé à un dossier (folder). Le module développé pour la synchronisation s'intéressera plus particulièrement aux dossiers oIFolderCalendar et oIFolderTasks.
Le contenu du dossier est accessible à partir d'un objet de type IFolder. Il est notamment possible de récuper l'ensemble des items d'un dossier sous la forme d'un objet IPOutlookItemCollection. Cet objet est en fait une collection d'objets spécifiques aux données contenues dans le dossier accédé. Il peut ainsi contenir des objets de type IAppointement ou ITask.

L'avantage de ce modèle est que les objets en bout de chaine comme les IAppointment ou ITask permettent non seulement d'accéder simplement aux données enregistrés dans l'aaplication Outlook, mais également d'effectuer des actions bas-niveau comme la suppression d'un item ou sa modification.
La classe IPOutlookItemCollection offre les méthodes permettant d'ajouter de nouveaux items (rendez-vous, tâche) à la base de données Outlook.
L'ensembles de ces objets sont liés dynamiquement à l'application Pocket Outlook.

### 7.3.2 - L'API SyncClient C++

L'API SyncClient C++ fournit l'ensemble des outils (classes C++, interfaces) permettant la réalisation d'une synchronisation SyncML.
Pour une présentation détaillée de l'organisation de cette API, référez-vous à la section 5 de ce document.

Afin d'établir une connexion avec un serveur de synchronisation, les objets de l'API, notamment le SyncManager, nécessite un certain nombre de configurations (adresse du serveur de synchronisation, identification de la source de données ciblée). Ces configurations sont accessibles par les fonction de l'API à partir d'une zone de la base de registre de Windows Mobile définit dans l'application sous le champs « application_uri ».
Le module de synchronisation développé proposera une interface de configuration permettant d'éditer les données de la base de registre.

La partie métier du module sera chargé de convertir les informations récupérés dans le calendrier Outlook au format vCalendar puis de les synchroniser avec le serveur de synchronisation. Les fonctions se chargeant de la conversion effective des données entre le modèle objet Outlook (IAppointment, ITask) et le format vCalendar formeront les éléments principaux à développer. Ils se composeront principalement de deux fonctions de traduction permettant d'obtenir une chaîne au format vCalendar à partir d'un objet IAppointment ou ITask et inversement.
Le partie métier du module se chargera également de veiller à la qualité de la liaison entre le PDA et le serveur de synchronisation.

## 7.4 - GESTION DES MODIFICATIONS

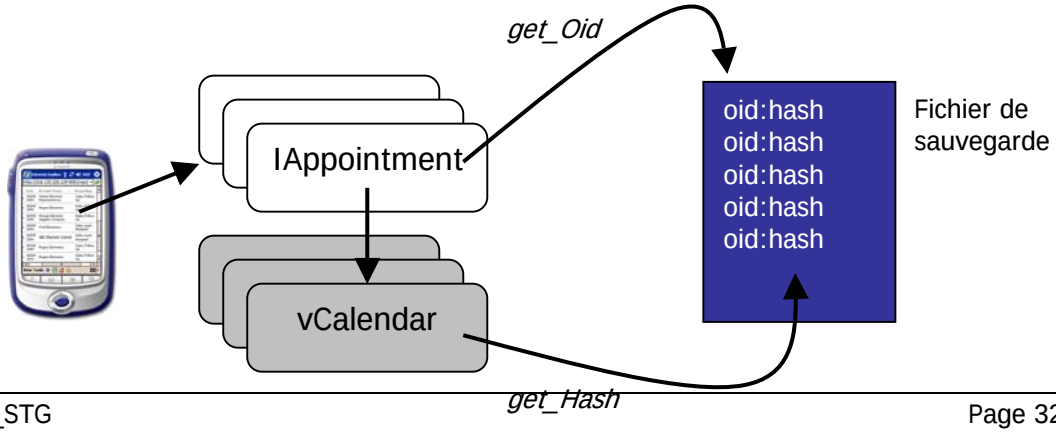
D'une manière générale, la synchronisation consiste à rapporter au serveur de synchronisation les changements apparus dans le calendrier du client depuis la dernière synchronisation. Ces changements peuvent prendre plusieurs formes :

- ajout d'un nouvel élément
- modification d'un élément
- suppression d'un élément

L'objet SyncSource de l'API cliente Sync4j décrit précédemment doit contenir l'ensemble des modifications opérées sur le calendrier avant d'être synchronisé avec le serveur par l'intermédiaire du SyncManager.
Afin de garder en mémoire les modifications apportés au calendrier par l'utilisateur entre deux synchronisations, la méthode suivante sera mise en place :

- Lors de la première synchronisation, l'ensemble des données de Pocket Calendar sont converties au format vCalendar. Pour chaque item traduit, on calcule un « hash » qui correspond à un entier unique associé à la trame vCalendar.
- Cet entier est alors associé à l'identifiant unique de l'item (oid) dans Pocket Calendar puis enregistré dans un fichier de sauvegarde sur le Pocket PC.
- Lors de la prochaine synchronisation, on traduit à nouveau chaque item au format vCalendar pour les envoyer au serveur. Cependant, avant l'envoi, on calcule le « hash » de chaque item et l'on vérifie s'il est identique au « hash » sauvegardé. Si le « hash » est différent, 'item a été modifié. Il sera donc ajoutés aux éléments modifiés.
- Si le « hash » n'existe pas dans le fichier de sauvegarde, c'est qu'il s'agit d'un nouvel item. Il sera donc ajoutés aux éléments nouveaux. Enfin, les items du fichier de sauvegarde n'ayant pas été associé à des items du calendrier sont à ajouter aux éléments supprimés.
- Après chaque synchronisation, le fichier de sauvegarde est mis à jour.

Le schéma suivant décrit la méthode de sauvegarde des items (par exemple les rendez-vous).



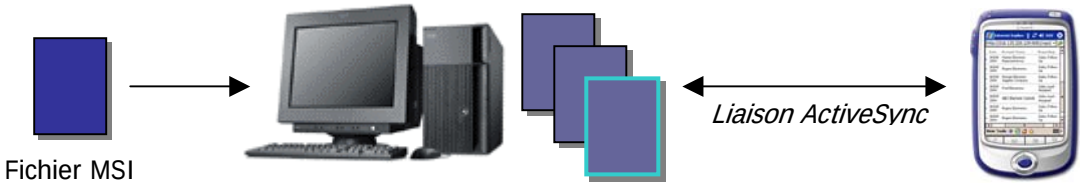
## 7.5 - DEVELOPPEMENT DE L'IHM

Le module de synchronisation pour Pocket PC s'intègrera directement dans le mode « Calendrier » de l'application Pocket Outlook. Ainsi l'IHM du module se limitera à une boîte de dialogue de configuration et une boîte de dialogue indiquant la progression en cours de synchronisation.
Ces deux éléments seront également développés en C++ sous la forme de deux classes SynchroConfigDialog et SynchroDialog. Ils utiliseront les librairies standards de l'API Windows.

## 7.6 - DISTRIBUTION DU MODULE

Le module sera distribué sous la forme d'un package MSI. Il s'agit en fait d'une archive exécutable depuis une station de travail relié au Pocket PC par l'intermédiaire d'une liaison SyncML. Cette archive contient différentes versions du module spécifiques à l'architecture du Pocket PC, compilés sous la forme de fichier CAB. Le fichier CAB correspondant à l'architecture cible est chargé et installé sur le Pocket PC automatiquement grâce au gestionnaire d'application de Windows CE.

Le schéma suivant détaille l'organisation du déploiement par l'intermédiaire d'un package MSI :




Le programme Cabwiz.exe et le compilateur MSI permettent de générer respectivement les fichiers CAB et la package MSI à partir de fichiers de configurations.

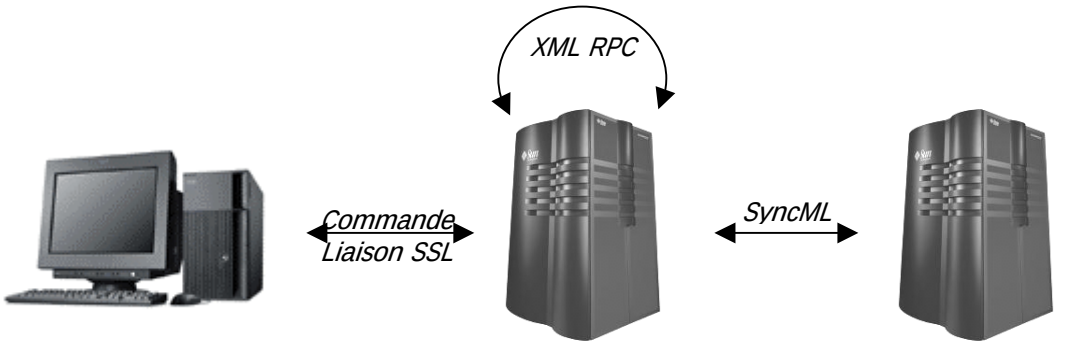
# SECTION 8 - CLIENT DE SYNCHRONISATION POUR OPENGROUPWARE

## 8.1 - GENERALITEES

Le client de synchronisation pour OpenGroupware (Ogo) sera principalement développé en Java.
Il se présentera sous la forme de différents modules permettant d'initialiser la synchronisation indifféremment depuis une option du menu outils de Mozilla Firefox, ou depuis une application Java/swing, à partir de n'importe quel ordinateur disposant d'un accès au serveur OpenGroupware.
Les principaux modules permettant la synchronisation d'Ogo sont les suivants :

- Un daemon de synchronisation attendant une commande pour initier la synchronisation entre Ogo et le server de synchronisation. Ce module utilisera principalement l'API JOGI pour se connecter au server Ogo via une liaison XML-RPC et l'API Sync4j Client pour effectuer la synchronisation avec le serveur Sync4j.
- Une application de commande pouvant se connecter au daemon afin de lancer la synchronisation. Cett application ne nécessitera que l'API standard Java pour être exécuté.
- Une extension Mozilla Firefox permettant d'exécuter l'application de commande depuis le menu du navigateur.

L'installation de ces différents modules se fera à partir d'une archive xpi pour la partie cliente et de scripts bash et shell pour le module Sync4j et le daemon de synchronisation.
Le figure ci-dessous représente l'organisation générale des modules du client de synchronisation Ogo.

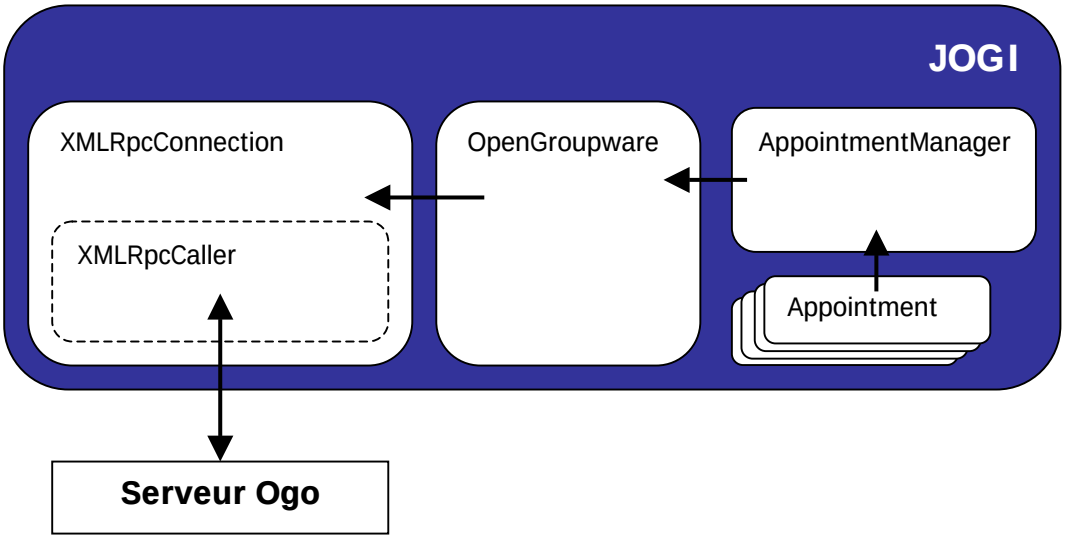



L'avantage de cette organisation est qu'elle limite le traffic réseau à l'envoi d'une commande de synchronisation et l'échange SyncML entre le serveur Ogo et Sync4j.

## 8.2 - LE DAEMON DE SYNCHRONISATION

Le rôle du daemon de synchronisation est d'initier la synhronisation entre le serveur Ogo et le serveur de synchronisation Sync4j.
Il est exécuté directement sur le serveur Ogo afin de limiter la charge réseau lors de la récupération des informations du calendrier d'Ogo.
Ce daemon est attaché à un port défini et attend une commande lui permettant d'effectuer une synchronisation entre le serveur Ogo et Sync4j.
La commande de synchronisation contient les informations de connection permettant au daemon de se connecter au serveur Ogo.
Afin d'éviter une attaque de type MIM lors du passage de la commande, celui-ci sera effectué via une liaison sécurisée.
Lorsqu'il reçoit une commande valide sur son interface réseau, le daemon se comporte comme un client de synchronisation classique. Il initialise la synchronisation en récupérant les mises à jours du calendrier Ogo puis en envoyant les commandes de modification correspondant au serveur Sync4j.

L'API JOGI permet d'accéder aux informations du serveur Ogo à partir d'un client Java. via une interface XML-RPC.
Le schéma suivant met en évidence les éléments permettant d'accéder au serveur Ogo.




L'objet AppointmentManager peut-être considéré comme une liaison dynamique avec le serveur Ogo. Il permet non seulement de récupérer le contenu du calendrier mais également d'y ajouter de nouveaux items.
L'API JOGI ne permet cependant pas d'obtenir la date de dernière modification des items, ainsi un système d'historique doit être mis en place afin de suivre les modifications du calendrier.
Une méthode similaire à celle décrite dans la section concernant le client Pocket PC sera mise en place afin de garder en mémoire les modifications opérées sur le serveur Ogo.
La principale différence résidera dans l'enregistrement de la table permettant de stocker les identifiants des items associés au hash des données leur correspondant. En effet, celle-ci sera sauvegardée en « serialisant » directement un objet Java de type Hashtable plutôt que d'utiliser un fichier texte.

Le daemon de synchronisation devra être lancé manuellement sur le serveur Ogo. Il pourra être exécuté en mode utilisateur pour éviter tout problème de sécurité lié à l'utilisation de Java.

Afin de réaliser une synchronisation avec le serveur Sync4j, l'application cliente doit disposer d'une classe qui implémente la classe SyncSource de l'API Sync4jClient. Celle-ci doit en particulier implémenter les méthodes permettant d'accéder aux items du calendriers d'Ogo et d'ajouter ou supprimers des items. Les principales méthodes à définir sont les suivantes :

- getAllSyncItems : cette méthode retourne la liste de tous les items de la source
- getNewSyncItems : cette méthode retourne la liste des items ajoutés sur Ogo depuis la dernière synchronisation
- getUpdatedSyncItems : cette méthode retourne la liste des items mis à jour sur Ogo depuis la dernière synchroisation
- getDeletedSyncItems : cette méthode retourne la liste des items supprimés sur Ogo depuis la dernière synchronisation
- setSyncItem : cette méthode permet d'ajouter un nouvel item ou une mise jour d'item pour Ogo
- removeSyncItem : cette méthode permet de supprimer un item du serveur Ogo

Une classe OgoItemsManager permettra de gérer les modifications sur le serveur Ogo est de générer les listes d'items à partir des données sauvegardés.

## 8.3 - L'APPLICATION DE COMMANDE CLIENTE

Afin de lancer une synchronisation, il est nécessaire d'établir une connection sécurisé avec le daemon de synchronisation pour lui envoyer les informations nécessaire à la synchronisation.
Cette étape sera réalisée par une application cliente Java exécutée depuis un poste client.
Sa tâche se limitera a établir une connection sécurisée avec le serveur puis à transmettre l'identifiant et le mot de passe Ogo de l'utilisateur.
Cette application sera utilisable depuis le menu Outils de Firefox ou par l'intermédiaire d'une interface graphique Swing. Il est donc prévu une utilisation en mode console ou interactif.

L'extension du menu de Firefox ainsi que l'installation de l'application cliente Java se feront par l'intermédiaire d'une archive xpi.

L'ensemble des paramètres de configurations de l'application de commande et de l'extension Firefox seront enregistrés dans le répertoire personnel de l'utilisateur.