UNO (Universal Network Objects)

Comme tout concept moderne UNO suit certaines règles d'organisation on parle de *spécifications*. Ces spécifications définissent qui fait quoi au sein de l'architecture.
Afin de ne pas perturber le programmeur avec un trop grand nombre de définitions et d'aspects abstraits il faut retenir dans un premier temps que les spécifications UNO définissent 05 grands concepts qui sont interdépendants :
Les *interfaces*, les *services*, les *propriétés, les méthodes* et les *modules.*

Les interfaces

Chaque *interface* est simplement constituée d'un ensemble d'une ou de plusieurs ***méthodes***, et vous pouvez utiliser ces ***méthodes*** pour:

- Obtenir ou définir les paramètres,
- Contrôler le fonctionnement de toutes les fonctionnalités définies par l'*interface*.

On peut effectivement penser qu'une *interface* est un *type*….

Les services

Un *service* UNO est un composant. Il s'agit de la pierre angulaire d'OOo. Chaque service se compose d'une ou plusieurs interfaces et il dispose d'un ensemble de *types* qui lui sont associés Il y a quatre types associés aux services: les *Constantes*, les *Structures*, les *Enums (Enumérations)*, et les *Exceptions.*

- Les *Constantes*, et les *Enums (Enumérations)* sont des valeurs prédéfinies.
  - Une *énumération* est un ensemble de valeurs numériques, regroupées en fonction de leurs usages.
- Les *structures* sont des types composés de plusieurs éléments. Une *structure* UNO, contient des propriétés qui sont directement accessibles : la représentation interne n'en est pas cachée.
- Les *exceptions* définissent le mécanisme de traitement des erreurs à l'exécution.

Les propriétés

Les *propriétés* représentent les données que met à disposition un objet. Elles sont accessibles via les méthodes génériques *getPropertyValue()* et *setPropertyValue()*.
Elles peuvent comme les méthodes, être d'un *type* simple (*char*, *long*, *float*, *string*...) ou plus complexes tels que déjà évoqués (*structures*, *énumérations*…).

Les *séquences* sont des ensembles d'éléments de même type.
Les *singletons* désignent les objets nommés dont une seule instance est autorisée dans le contexte d'un composant UNO.

Les modules

Pour des raisons d'organisation et de commodité les *services* similaires et leurs *propriétés* sont regroupés en *modules*.

En conclusion

- Une *interface* est un *type* composé d'une ou plusieurs *méthodes*
- Un *service* se compose d'une ou de plusieurs *interfaces*
- Un ensemble de *propriétés* (*constantes*, e*nums*, *exceptions*, et *structures*) est associé avec le service.
- Pour organiser le tout on a regroupé les *services* similaires et leurs *propriétés* en *modules*.

Nous avons vu l'organisation d'UNO du niveau le plus bas à celui le plus haut de l'architecture. Certains des modules que nous utiliserons sont imbriqués dans d'autres modules. En fait, tous les modules que nous allons utiliser sont imbriqués dans un module central : com.sun.star. Nous avons maintenant une vue d'ensemble du modèle Objet d'Ooo.

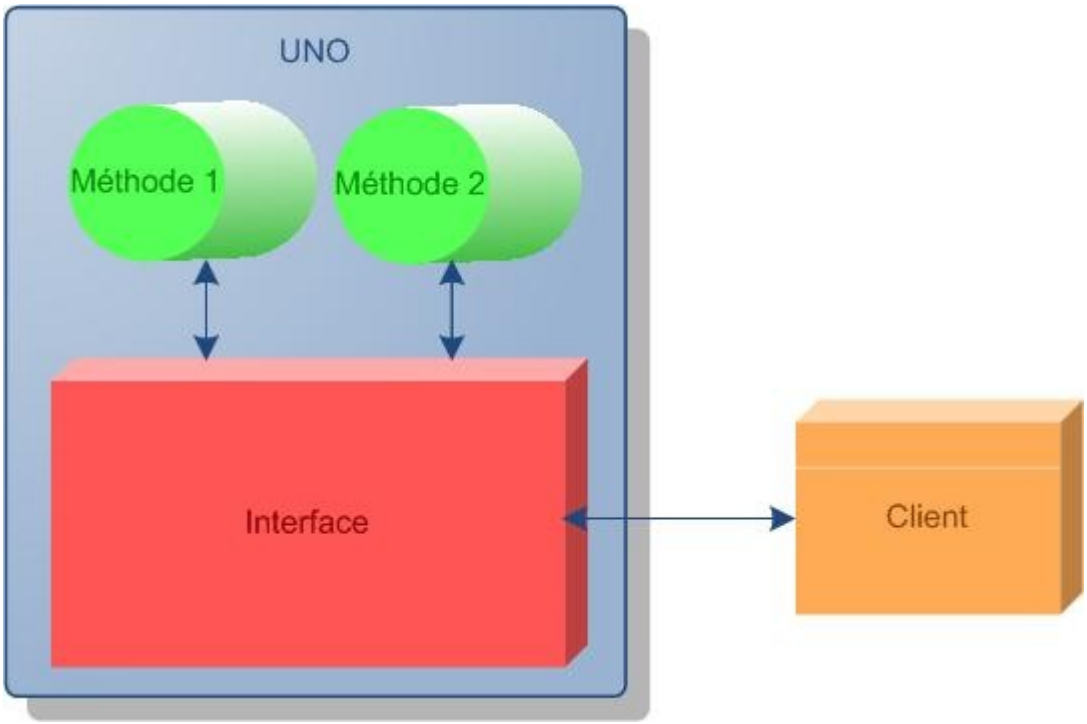
De façon plus complète

Les spécifications UNO définissent :

Les interfaces

Une *interface* définit la manière dont quelque chose interagit avec son environnement. Par exemple, un ordinateur a plusieurs interfaces : une "écran", une "unité centrale", une "périphériques", une "système d'exploitation" etc... Une interface selon UNO rassemble un groupe de sous-routines et des déclarations de fonctions. Les arguments et les types de retour sont précisés en même temps que la fonctionnalité.
Vous pouvez utiliser l'interface pour récupérer les "*données d'un objet"*, "*un ensemble de données dans un objet*," ou dire *"à un objet de faire quelque chose"*.

- L'interface indique comment un objet peut être utilisé, mais elle ne dit rien sur son fonctionnement ou sa conception interne. Par exemple, si une interface contient des méthodes permettant d'obtenir un entier qui représente une largeur, il est alors naturel de supposer que l'objet contient une propriété nommée *largeur*. Il est possible, toutefois, que la largeur soit calculée directement par l'objet ou qu'elle soit dérivée d'autres propriétés. Ainsi l'interface ne précise pas la façon dont la largeur est obtenue, juste qu'il est possible de l'obtenir.



- Tous les noms d'interface UNO commencent par la lettre "X" qui permet facilement de reconnaître une interface. Par exemple, l'interface com.sun.star.text.XTextRange spécifie une section de texte, à la fois le début et de fin de sa position. Les Objets qui supportent l'interface *XTextRange* sont aussi utilisés pour déterminer la position de l'objet dans un texte.
- Les méthodes définies par l'interface *com.sun.star.text.XTextRange sont :*

Method	Description
getText()	Renvoie l'interface com.sun.star.text.XText qui contient XTextRange.
getStart()	Renvoie la référence à la position de début de com.sun.star.text.XTextRange.
getEnd()	Renvoie la référence à la position de fin de com.sun.star.text.XTextRange.
getString()	Renvoie une variable de type string qui inclut le texte à l'intérieur de la plage (Range).
setString(str)	Définit (Set) la chaine pour cette place de texte, remplace le texte existant, et efface les styles qui y sont attachés.

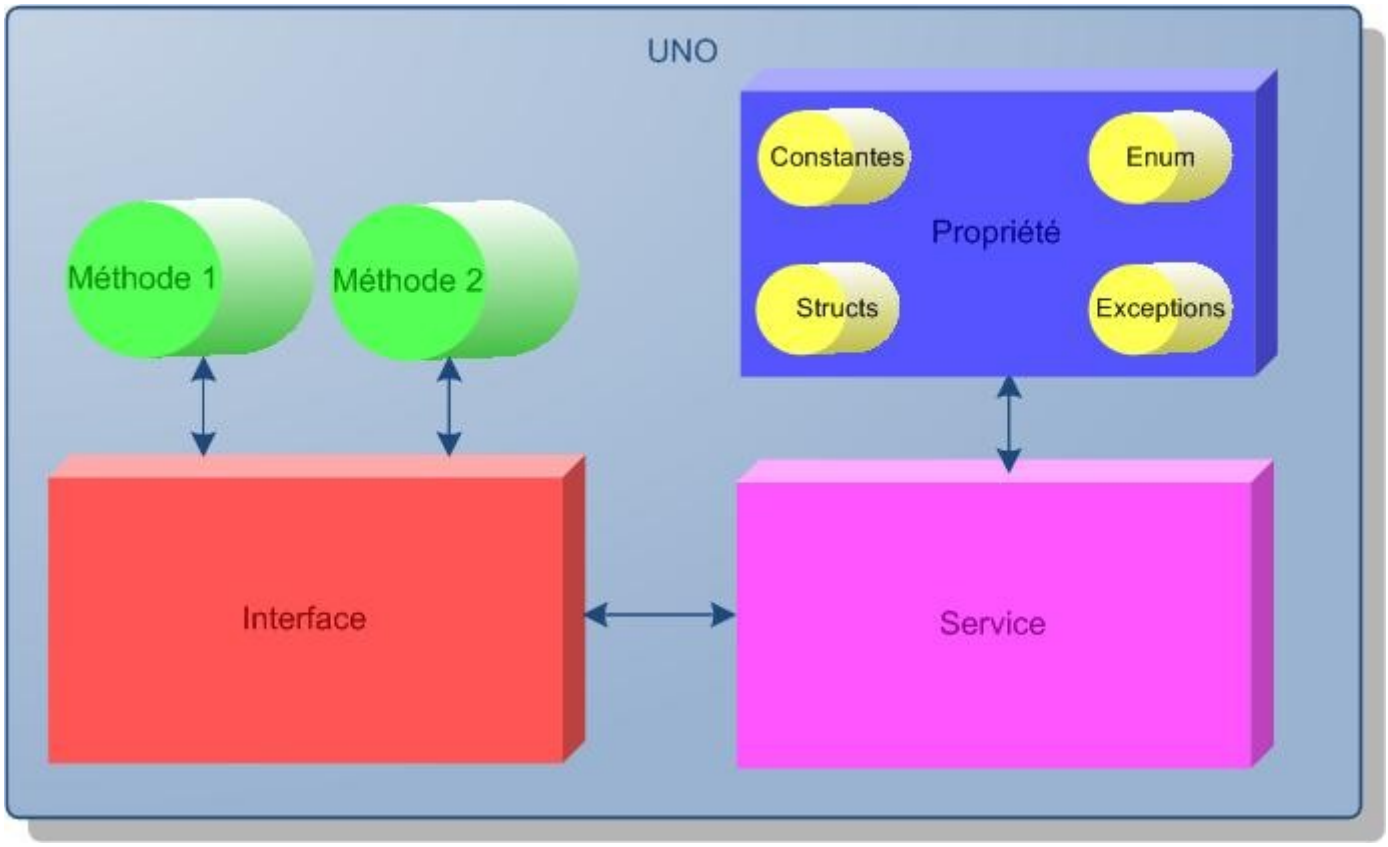
- Une interface UNO peut être dérivée d'une autre. Ce n'est pas quelque chose que vous faites, mais c'est quelque chose qui est fait par le designer de l'interface. Les interfaces dérivées prennent en charge toutes les méthodes définies dans l'interface dont elles sont dérivées. Par exemple, l'interface com.sun.star.text.XTextCursor étend l'interface XTextRange afin de lui permettre d'élargir la plage sélectionnée. L'interface XTextRange ne fournit

aucune méthode pour modifier un texte par lui-même. Cependant tout objet qui implémente l'interface XTextCursor, supporte toutes les méthodes de XTextRange et les nouvelles méthodes introduites par l'interface XTextCursor. Les méthodes définies par les interfaces XTextCursor et XTextRange sont utilisées pour manipuler les documents OOo Writer.

- Chaque interface UNO est dérivée de com.sun.star.uno.XInterface.
- ***En conclusion :*** Une *interface* définit les aspects extérieurs particuliers d'un objet auxquels on accède sans rien savoir de leur représentation interne. Pour celui qui les implémente : ce sont des spécifications abstraites, que l'on peut interpréter comme correspondant à un *Type* particulier.

Les services

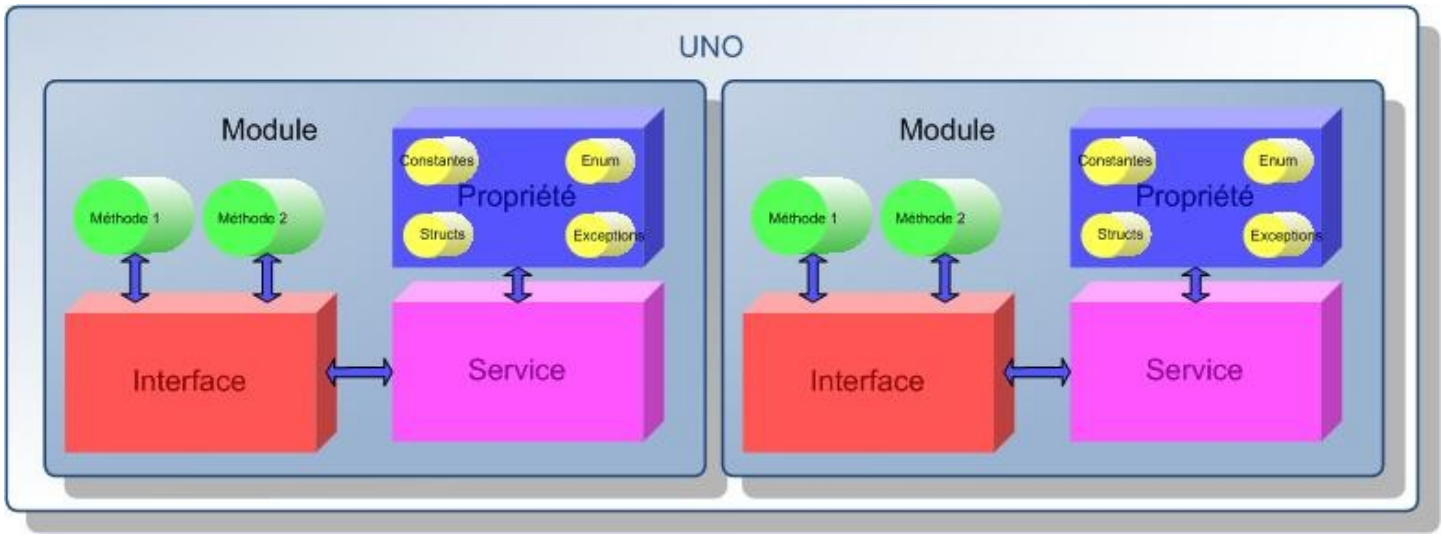
- Un *service* spécifie le comportement d'un objet. C'est à dire que lorsque l'on appelle un service celui instancie un composant à l'appel de son nom.
- C'est seulement depuis la version 2.0 que ce concept est définit concrètement. Ainsi depuis la version 2.0 un service spécifie un objet qui implémente une interface. Ce service n'existe que dans un contexte de *composant*. Pour des raisons de compatibilité de nombreux *services* « mal définis » ont été conservés.
- Un *service* UNO définit un objet en combinant des *interfaces* et des *propriétés* pour encapsuler certaines fonctionnalités utiles. Une *interface* UNO définit la manière dont un objet interagit avec le monde extérieur, une *structure* UNO définit une collection de données et un *service* UNO les amalgame. Strictement parlant, un *service* est seulement une définition. UNO est l'objet réel créé tel que défini par le *service*. Un *service* peut inclure plusieurs *services* et des *interfaces*. Une *interface* définit généralement un seul aspect d'un *service* et a donc généralement une portée moins étendue.
- Parfois, un *service* peut avoir un nom similaire à une *interface*; chercher le X dans le nom permet de déterminer s'il s'agit d'une *interface* ou un *service*. Par exemple, le *service* com.sun.star.text.TextCursor implémente (met également en œuvre) d'autres *services* et des *interfaces*. Sur la base de l'*interface* XTextCursor, tous les TextCursor peuvent se déplacer à gauche et à droite d'un certain nombre de caractères et sauter au début ou à la fin du texte. Le *service* TextCursor, cependant, peut éventuellement mettre en œuvre l'*interface* com.sun.star.text.XWordCursor, qui contient des *méthodes* liées aux *mots*.
- ***En conclusion :*** On peut poser le postulat que les objets UNO sont simplement appelés des *services*.



Les modules

Les *modules* sont des espaces de noms (les « packages » de Java). Ils regroupent toutes sortes d'objets UNO qui structurent l'API d'un point de vue logique. Ils ne sont pas structurés en fonction des différentes applications d'OOo (Writer, Calc, Draw...) mais selon des catégories fonctionnelles souvent communes à toutes ces applications.

Si l'on examine la structure « physique » d'OOo (ses fichiers et dossiers) on s'aperçoit qu'OOo n'est pas constitué de plusieurs gros *Exécutables*, mais de petits *lanceurs* (*swriter.exe*, *scalc.exe* etc…) qui démarrent différemment le même fichier : *SOffice.bin*



A l'examen de l'organisation d'UNO on peut affirmer (n'en déplaise à ses détracteurs) qu'OpenOffice illustre parfaitement ce qu'est un *Intergiciel* bien conçu.
http://www.emse.fr/~vercouter/cours/intergiciels/intergiciels.pdf

La connexion avec OpenOffice

Le préalable à toute programmation est déjà de se connecter ou de référencer notre programme cible vers notre langage de programmation.

En Java :

Nous l'avons déjà vu, un nouveau projet quel qu'il soit doit référencer les packages OOo.
Les packages essentiels sont :

- sandbox.jar
- ridl.jar
- unoil.jar
- jurt.jar
- juh.jar

Depuis la version 3.0 les packages sont situés dans des endroits divers.

- <Dossier d'Installation>\ program\basis\program\classes
- <Dossier d'Installation>\ program\URE\java

De plus le dossier <Dossier d'Installation>\ program\URE\java intègre deux nouveaux packages :

- java_uno.jar
- unoloader.jar

Afin d'éviter certains problèmes de localisation des classes entre elles vous pouvez copier l'ensemble des classes dans un dossier qui vous sera personnel.

Avec CLI :

La démarche est la même qu'en java et est habituelle pour les programmeurs .net.
Les bibliothèques à référencer sont préfixés par "cli_":

- Cli_uno.dll: il s'agit du pont CLI-UNO qui réalise l'interaction entre le code managé (CLI) et UNO. Il ne fournit pas de *Types* publics.
  Cli_cppuhelper.dll: Fournit le code d'amorçage et de démarrage Les *Types* de cette bibliothèque sont toujours utilisés par les programmes clients.
- Cli_ure.dll: Contient des classes d'aide qui sont utiles pour la mise en œuvre des interfaces UNO. Les *Types* de cette bibliothèque ne sont pas toujours utilisés.
- Cli_oootypes.dll: (à partir de OOo 3.0) Fournit des classes et interfaces pour les composants et les clients. C'est une collection de toutes les interfaces UNO actuellement utilisées dans OOo. Les *Types* de cette bibliothèque sont toujours utilisés par les programmes clients.
- Cli_uretypes.dll: (à partir de OOo 3.0) fournit les types URE
- Cli_basetypes.dll: Comme son nom l'indique, elle fourni certains types de base. Les *Types* de cette bibliothèque sont toujours utilisés par les programmes clients. En outre, cli_uretypes.dll et cli_oootypes.dll en dépendent.

Sauf pour cli_uno.dll, elles sont installées dans le Global Assembly Cache (GAC).Vous pouvez également les retrouver dans le SDK dans le dossier « cli »
Afin d'éviter certains problèmes de localisation des classes entre elles vous pouvez copier l'ensemble des classes dans un dossier qui vous sera personnel.

Les imports nécessaires en VB.Net pour faciliter la saisie du code sont :

```
Imports unoidl.com.sun.star.lang
Imports unoidl.com.sun.star.uno
Imports unoidl.com.sun.star.bridge
Imports uno.util
```

Ce sont les mêmes en C#

```
using uno.util
using unoidl.com.sun.star.lang;
using unoidl.com.sun.star.uno;
using unoidl.com.sun.star.bridge;
```

Avec COM ou OLE:

Le protocole étant le même pour ces deux modes nous ne parlerons plus que de connexion via COM.

Il n'y a aucune référence à ajouter, c'est COM qui va le faire pour nous lors de la phase de connexion.

La connexion proprement dite

Les éléments principaux

Le Service Manager

UNO introduit le concept de gestionnaire de service (service managers) qui peuvent être considérer comme des « usines » qui créer des services.

Ex : com.sun.star.configuration.ConfigurationProvider correspond au panneau de Configuration d' OpenOffice (Outils, Options, Configuration). Il existe un *Service Manager* principal qui permet l'instanciation d'un composant dans un contexte UNO. Ce dernier a accès à une base de données des composants enregistrés et est capable de les créer à l'appel de leurs noms.
Une fois le ServiceManager obtenu vous pouvez ensuite créer un service depuis le ServiceManager en utilisant sa méthode *createInstance* (non disponible directement en java).
Quelque soit le langage de programmation, le ServiceManager a des méthodes pour créer un service avec des arguments, c*reateInstanceWithArguments*, et pour obtenir une liste de tous les services supportés, *getAvailableServiceNames*.
Une des premières chose à obtenir pour travailler avec UNO est donc d'initialiser le ServiceManager principal.

OOo Basic implémente une fonction CreateUnoService qui permet de simplifier la création de ces services, mais il est très simple de le faire depuis le servicemanager que l'on appelle directement par GetProcessServiceManager ()

Le ServiceManager est obtenu de la façon suivante

```
code
```

Composant et contexte

Un composant est défini par un gestionnaire de service et un certain nombre de données associées (un contexte). Ainsi un service existe toujours dans un contexte de composant. Le composant est le service manager qui délivre le service ou les données utilisées par le service.
Une connexion est considérée comme un client du process OpenOffice : OpenOffice est un serveur d'application.
Ce serveur a son propre contexte de composant et donc son propre service manager auxquels les programmes clients peuvent se connecter pour utiliser les fonctionnalités d'OpenOffice.
Ainsi :

Lors de la connexion :

- Le programme client initialise UNO : et obtient le contexte de composant du process d'OpenOffice.
- De façon interne le process d'initialisation créer un service manager local.
- Il établit une connexion avec une instance active d'OpenOffice (il créé une instance d'OpenOffice si nécessaire)
- Retourne le « remote » contexte de composant
- La méthode getServiceManager offre alors le « remote » service manager du composant
- Qui offre alors l'accès aux fonctionnalités au travers de l'API.

Objets : Dans UNO un objet est un document ou une procédure identifiée au sein d'un processus qui a :

- des méthodes que l'on peut appeler
- des attributs que l'on peut obtenir (get) ou définir (set)

Les méthodes et les attributs qu'offre un objet sont spécifiés par les interfaces que l'objet supporte.
Un objet peut par exemple créé par la méthode createInstanceWithContext.