	Spécification technique du module	**Date rédaction :** **22 avril 2004.**
	Diffusion restreinte	**Date validation :**
Référence : UpComp-UpsOms-000003-A Spécification technique du module UpsOms.doc		

Suivi des versions-révisions et des validations du document.			
Ce document annule et remplace tout document diffusé de version-révision antérieure. Dès réception de ce document, les destinataires ont pour obligation de détruire les versions-révisions antérieures, toutes les copies, et de les remplacer par cette version. Si les versions-révisions antérieures sont conservées pour mémoire, les destinataires doivent s'assurer qu'elles ne peuvent être confondues avec cette présente version-révision dans leur usage courant.			
Version.	**Date.**	**Auteurs.**	**Création, modification ou validation.**
A	23 nov. 2003.	JPD.	Création.

1 Tables

1.1 Table des matières

1.2 Table des illustrations

2 Références

2.1 Glossaire

Liste des définitions des termes employés.	
Ce tableau recense tous les termes, les concepts particuliers ainsi que les abréviations employés dans ce document.	
Terme, concept, abré.	**Définition du terme, du concept ou de l'abréviation.**
Contrainte	Voir page 23.
Journal	Voir page 24.
Photographie	Voir page 24.
Session	Voir page 12.
Tâche	Voir page 12.
Transaction	Voir page 24.

2.2 Ressources

Liste des documents applicables et en référence.

Un document est **applicable** à partir du moment où son contenu est validé et que l'activité ou le projet fait partie de son périmètre d'application. Il est obligatoire d'appliquer son contenu.

Un document est en **référence** à partir du moment où son contenu n'est pas validé ou que l'activité ou le projet ne fait partie de son périmètre d'application. Il est recommandé d'appliquer son contenu mais cela n'est pas obligatoire.

Un document applicable est indicé par ***A1***, ***A2***, ***A3***, etc. Un document en référence est indicé par ***R1***, ***R2***, ***R3***, etc.

Index.	Nom du document.	Commentaire.
A1	UpComp-Plan Qualité-000005	Méthode documentaire.
A2	UpComp-Plan Qualité-000006	Processus de management de projet.
A3	UpComp-Plan Qualité-000046	Méthode de spécification technique d'un module.
A4	UpComp-Upsoms-000002	Plan documentaire du projet.
A5	UpComp-UpsVm-000003	Plan de programmation.
A6	UpComp-UpsVm-000004	Programmation en ***C--***.
R7	http://www.up-comp.com	Site ***Internet*** d'***Up ! Application System***.
A8	UpComp-UpsKrn-000003	Spécification technique du module ***UpsKrn***.

3 Introduction

3.1 Objet du document

L'objet de ce document est de décrire le contenu technique du module logiciel ***Up ! Object Management System*** pour le projet ***Up ! Application System***.

Ce document est rédigé et approuvé par la ***Maîtrise d'Oeuvre*** (**MOE**).

3.2 Audience

Ce document s'adresse aux :

- Directeurs de projets et chefs de projets.
  Pour la compréhension du module technique.
- Ingénieurs de développement.
  Pour savoir comment est conçu le module technique.

Pour aider ces personnes à remplir le document ***Spécification technique d'un module***, leur manager et la cellule de support projet se tiennent à leur disposition.

3.3 Pré-requis

Le pré-requis est la connaissance des documents suivants :

- **Méthode documentaire** [A1].
- **Processus de management de projet** [A2].
- **Méthode de spécification technique d'un module** [A3].

Nous rappelons que tous les documents applicables ou référencés pour le projet ***Up ! Application System*** sont tracés dans le **Plan documentaire** [A4].

4 Interaction avec l'environnement

4.1 Description

L'objet du module logiciel ***Up ! Object Management System*** est de

- Gérer les objets d'***Up ! Application System***.
  Le cycle de vie complet des objets – création, consultation, modification, suppression, destruction.
- Gérer la base de données objet interne.
  Accès concurrent, transaction, importation / exportation, archivage, reprise.
- Offrir un fonctionnement efficace.
  Parallélisme via le multi thread ou le multi processus.
- Gérer les objets longs.
  Via le type privé **Buffer**.

4.2 Paramètres

Tous les paramètres sont documentés sur le **Site Internet d'*Up ! Network*** [R7].

4.3 Particularités

4.3.1 Compilation

Néant.

4.3.2 Exécution

Néant.

4.4 Application Program Interfaces

Néant.

5 Choix techniques

5.1 Adressage des objets

Cette section décrit comment l'adressage des objets est réalisé par ***Up ! Object Management System***.

Les différentes alternatives possibles sont les suivantes :

- Utiliser l'adressage classique du langage ***C++***.
- Utiliser l'adressage classique du langage ***C***.
- Utiliser un mécanisme d'adresse logique masquant la mise en oeuvre physique.

La troisième alternative a été retenue pour la raison suivante :

- Pagination des objets.
  L'adressage classique du langage ***C++*** ne permet pas une pagination des objets dans des fichiers d'échange du fait des champs cachés et des tables des méthodes non maîtrisés.
- Partage des objets.
  L'adressage classique du langage ***C++*** ne permet pas un partage des objets entre plusieurs processus.
- Circulation des objets.
  L'adressage classique du langage ***C++*** ne permet pas une circulation des objets entre plusieurs programmes, sans la mise en oeuvre de normes lourdes telles ***Corba*** ou ***DCom***.
- Sûreté de fonctionnement.
  L'adressage classique du langage ***C*** ne permet pas de mettre en oeuvre des mécanismes de vérification des adresses en vue d'éviter un dysfonctionnement fatal.

  Compte-tenu de ces raisons, il n'est pas possible de changer de choix.

5.1.1 Stockage des objets

Un objet est conservé dans un entrepôt. Il y a au plus 65 535 entrepôts et l'entrepôt numéro ***1*** est l'entrepôt particulier appelé ***Système***. Un entrepôt est composé de blocs mémoire dont la taille est donnée par le paramètre **taillebloc**.

Un entrepôt est découpé en segments. Il y a au plus 65 535 segments. Un segment est constitué au plus de 255 blocs contigus de mémoire

Un segment est soit formaté pour un type de données particulier de type ***Type*** ou soit non formaté. Dans ce dernier cas, il est conserve des objets longs via le type interne ***Buffer***, qui est notamment utilisé pour les chaînes de caractères.

Quand un segment est formaté, les objets sont repérés par un index. Il y a au plus 65 535 objets dans un segment.

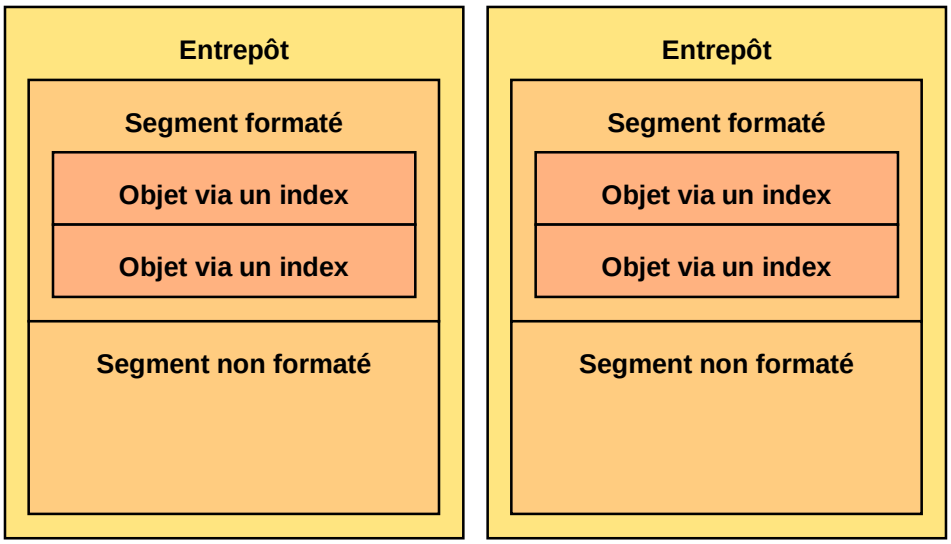

Diagramme 1 – Principe du stockage des objets

5.1.2 Nature des entrepôts

Les différentes natures des entrepôts sont les suivantes :

- Mémoire fixe versus mémoire paginée.
  Tout l'entrepôt réside en mémoire physique quand la mémoire est fixe.
  Tout l'entrepôt ne réside pas complètement en mémoire physique quand la mémoire est paginé.
  Il est adossé à un fichier d'échange représentant tout l'entrepôt. Seuls les caches de l'entrepôt permettant de conserver les segments plus utilisés sont en mémoire physique.
- Allocation unique versus allocation incrémentale.
  Toute la mémoire physique nécessaire à l'entrepôt est allouée globalement quand l'allocation est unique.
  Sinon, la mémoire physique est allouée et libérée incrémentalement en fonction des besoins.
- Mémoire compactable versus mémoire non compactable.
  Quand la mémoire est compactable, ***Up ! Object Management System*** peut déplacer les objets de la sorte à boucher les trous créer par les créations / destructions.
  Le bénéfice est une meilleure gestion de la mémoire au détriment d'un temps de gestion légèrement plus important.

5.1.3 Type des entrepôts

Les différents types d'entrepôts sont les suivants :

- Privé.
  Il contient des objets qui sont uniquement accessibles par le processus courant, à l'exclusion des autres processus s'exécutant sur la machine ou sur les autres machines.
- Protégé.
  Il contient des objets qui sont uniquement accessibles aux processus s'exécutant sur la machine locale, à l'exclusion des autres processus s'exécutant sur les autres machines.
  La communication inter processus est réalisée via ***Up ! Network***.
- Public.
  Il contient des objets qui sont accessibles à tous les processus s'exécutant soit sur la machine

locale ou soit sur les autres machines.
La communication inter machine est réalisée via ***Up ! Network***.

S'il existe un entrepôt protégé, alors l'entrepôt système doit être de type protégé. S'il existe un entrepôt public, alors l'entrepôt système doit être de type public.

5.1.4 Adresse d'un objet

Un objet est repéré par une adresse logique de type ***TypUpsVmAdresse*** contenant les champs suivants :

- ***NumeroSegment***.
  Numéro du segment dans lequel est conservé cet objet. Il est toujours positif strictement.
- ***NumeroNoeud***.
  Numéro du nœud comportant le serveur logiciel qui gère cet objet. Il est toujours positif strictement.
- ***NumeroServeur***.
  Numéro du serveur logiciel qui gère cet objet. Il est toujours positif strictement.
- ***Index***.
  Index de l'objet dans le segment. Il est toujours positif strictement.
- ***Architecture***.
  Architecture de l'objet.
- ***Options***.
  Options de l'objet.

Pour les entrepôts de mémoire fixe privée – le cas le plus fréquent –, il y a les champs suivants permettant d'accélérer la recherche d'un objet :

- ***AdresseBloc***.
  Adresse du bloc de mémoire physique dans lequel est conservé cet objet.
- ***AdresseObjet***.
  Adresse physique de l'objet.
- ***NbBlocs***.
  Nombre de blocs dans le segment dans lequel est conservé l'objet.

Tous ces champs sont uniquement accédés en lecture en dehors du module ***Up ! Object Management System***. Par convention, seul ce dernier modifie les valeurs de ces champs, sous peine de dysfonctionnements graves.

Un objet ***Nul*** est représenté par une adresse non nulle dont le champ ***NumeroSegment*** est nul.

Pour initialiser une adresse, il faut utiliser l'***Application Program Interface*** (**API**) ***UpsCom.AdresseANul***.

Pour copier une adresse dans une autre, il faut utiliser l'***Application Program Interface*** (**API**) ***UpsCom.CopierAdresse***.

Pour comparer deux adresses, il faut utiliser l'***Application Program Interface*** (**API**) ***UpsCom.AdresseEgales***. Cette fonction retourne ***Vrai*** en cas d'égalité.

5.1.5 Architecture d'un objet

L'architecture d'un objet peut être la suivante :

- Encodage en ***Big endian*** ou en ***Little endian***.
  Cela dépend de la plate-forme qui a créé l'objet.
- Encodage en 16 bits, 32 bits ou 64 bits.
  Cela dépend de la plate-forme qui a créé l'objet.
- Objet ***Corba***.
  Il a été créé par un module serveur encapsulant des types ***Corba***.
- Objet ***DCom***.
  Il a été créé par un module serveur encapsulant des types ***DCom***.
- Objet ***Java***.
  Il a été créé par un module serveur encapsulant des classes ***Java***.

  L'architecture est mémorisée par un codage bit dans le champ ***Architecture*** de l'adresse.

5.1.6 Options d'un objet

Un objet peut comporter les options suivantes :

- Appartenance.
  L'objet est la propriété d'un utilisateur particulier.
- Habilitations.
  L'objet comporte des habilitations quant à son accès à ses propriétés ou à ses méthodes.
- Interface.
  L'objet est issu d'un type qui implémente une interface.
- Propriétés dynamiques.
  L'objet peut comporter des propriétés dynamiques.
- Quota.
  L'objet doit respecter des quotas particuliers.
- Ressource.
  L'objet encapsule une ressource d'un logiciel de base.
- Transactionnel.
  Les modifications apportées à l'objet doivent être gérées par des transactions.

  L'architecture est mémorisée par un codage bit dans le champ ***Options*** de l'adresse.

5.2 Etat du noyau

Up ! Object Management System est un gestionnaire offrants des services fondamentaux quand à la manipulation des objets d'***Up ! Application System***. Or, comme pratiquement tous les concepts sont soit des objets créés stricto sensu, soit des objets encapsulant des ressources, il est nécessaire que ce système soit opérationnel.

D'autre part, ***Up ! Object Management System*** dépend des services fournis par les autres modules constituant ***Up ! Virtual Technical Machine***, aussi le gestionnaire démarre incrémentalement et s'arrête incrémentalement. Il en est de même pour les autres modules qui utilisent également les services ***Up ! Object Management System***.

Afin de savoir dans quel état est le noyau d'exécution d'***Up ! Virtual Technical Machine***, il y a l'***Application Program Interface*** (**API**) ***LireEtatNoyau*** qui retourne le résultat suivant :

- **EN_Arrete**.
  Seuls les services d'***Up ! Module*** fonctionnent.

Les modules d'***Up ! Virtual Technical Machine*** s'initialisent en autonome par ordre de priorité afin d'être robustes.

- **EN_DemarrageEnCours**.
  Les services essentiels offerts par ***Up ! Virtual Technical Machine*** fonctionnent.
- **EN_Demarre**.
  Tous les services offerts par ***Up ! Virtual Technical Machine*** sont opérationnels.
- **EN_ArretEnCours**.
  Les services essentiels offerts par ***Up ! Virtual Technical Machine*** fonctionnent encore. Les services de haut niveau ne fonctionnent plus.
- **EN_PresqueArrete**.
  Seuls les services d'***Up ! Module*** fonctionnent encore.
  Les modules d'***Up ! Virtual Technical Machine*** se terminent en autonome par ordre de priorité décroissante afin de libérer les dernières ressources.
- **EN_Arrete**.
  Seuls les services d'***Up ! Module*** fonctionnent encore.

Il est important de s'arrêter de la disponibilité des services en testant l'état du noyau pour des traitements qui sont utilisés au cours du lancement ou de l'arrêt d'***Up ! Virtual Technical Machine***.

5.3 Sessions et tâches

M Cette section décrit comment la mémoire de travail des threads est gérée par ***Up ! Object Management System***.

Les différentes alternatives possibles sont les suivantes :

- Utiliser un espace de mémoire partagé par tous les threads.
- Utiliser un espace de mémoire propre à chaque thread.

Les deux alternatives sont nécessaires pour les raisons suivantes :

- Synchronisation des threads.
  Un espace de mémoire partagée nécessite un mécanisme de synchronisation qui s'avère fort coûteux en temps d'exécution.
- Ressources nécessaires.
  Un espace de mémoire propre à chaque thread est gourmand en ressources.
- Travail collaboratif.
  Un espace de mémoire propre à chaque thread ne permet pas un travail collaboratif.

L'espace mémoire propre à chaque thread est peut être mise en oeuvre selon les alternatives suivantes :

- Utiliser la mémoire ***Thread Local Storage*** (**TLS**).
- Utiliser de la mémoire conventionnelle partitionnée.

La seconde alternative a été retenue pour la raison suivante :

- Portabilité.
  Le mécanisme de ***Thread Local Storage*** n'est pas disponible sur toutes les familles de système d'exploitation pour lesquelles ***Up ! Application System*** est disponible.

5.3.1 Session

Une **session** est une identification unique d'une unité d'exécution – le thread – pour ***Up ! Virtual Technical Machine*** permettant de retrouver rapidement :

- L'entrepôt par défaut.
  Pour créer de nouveaux objets.
- Le fichier d'échange par défaut.
  Pour créer de nouveaux objets.
- La connexion utilisée.
  Pour vérifier les habilitations et les quotas.
- Un espace mémoire propre à chaque thread.
  Pour éviter des synchronisations trop coûteuses.

Une session est modélisée par le type ***TypUpsVmSession***. Son descripteur est passé par adresse à pratiquement chaque ***Application Program Interface*** (**API**) d'***Up ! Virtual Technical Machine*** puisqu'elle est obligatoire dès lors que ***Up ! Object Management System*** est sollicité.

Une session est identifiée par un numéro non nul conservé dans le champ ***NumeroSession***.

Quand le noyau n'est pas dans l'état **EN_Demarre**, il est existe une session spéciale appelée connexion interne, dont le numéro est **CO_SessionInterne**. La session a utiliser est alors ***UpsOms.SessionPrincipal***.

Pour ouvrir une nouvelle session, suite à la création d'une thread, il faut appeler l'***Application Program Interface*** (**API**) ***UpsOms.OuvrirSession***.

Pour fermer une session existante, avant la destruction d'une thread, il faut appeler l'***Application Program Interface*** (**API**) ***UpsOms.FermerSession***.

Pour retrouver la session d'un thread dans une ***Callback*** système, il faut appeler l'***Application Program Interface*** (**API**) ***UpsOms.RechercherSessionThread***.

5.3.2 Tâche

Une **tâche** est l'assemblage de :

- Un thread.
  Pour exécuter le programme demandé.
- Une session.
  Pour retrouver le contexte d'exécution.
- Un espace mémoire dédié.
  Pour éviter des synchronisations trop coûteuses.
- Un contexte transactionnel.
  Pour masquer les créations, les modifications et les suppressions d'objets aux autres tâches tant que les informations ne sont pas cohérentes.

Quand une nouvelle tâche est créée, il faut le signaler à ***Up ! Object Management System***. Cela s'effectue par l'appel à l'***Application Program Interface*** (**API**) ***UpsOms.SynchroniserSecondeTache***.

Quand une tâche est susceptible d'être suspendue pour un long moment, par exemple lors de l'appel à ***UpsSys.Attendre***, il faut le signaler à ***Up ! Object Management System***. Cela s'effectue par l'appel à l'***Application Program Interface*** (**API**) ***UpsOms.SupprimerTacheActive***.

Quand la tâche redevient active, il faut appeler l'***Application Program Interface*** (**API**) ***UpsOms.AjouterTacheActive***.

5.4 Cycle de vie d'un objet standard

Un objet standard est géré par ***Up ! Object Management System*** en terme de cycle de vie. La caractéristique est que tous les objets de son type de rattachement ont la même taille. Les segments sont donc formatés en enregistrements de taille fixe.

5.4.1 Création d'un objet

La création d'un objet s'effectue via l'***Application Program Interface*** (**API**) ***UpsOms.AllouerUnObjet*** qui est appelée par un des constructeurs du type de l'objet. Voici un exemple :

```
TypUpsVmAdresse AdresseObjetType;
TypUpsVmAdresse AdresseObjet;
TypUpsVmUnsignedLong NumeroVerrouObjet;
TypUpsVmPointeurDonnees Implementation;
...
AllouerObjetStandard)(Session,
    &AdresseObjet, &AdresseObjetType);
if (!(*MonModIntTrtUpsOms->AllouerUnObjet)(Session,
   &AdresseObjet, &ObjetType, &NumeroVerrouObjet, &Implementation))
   {
   ...
   }
...
```

Texte 2 – Exemple de création d'un objet

Si un nouvel objet ne peut être créé, le résultat est ***NULL***.

M Quand ***UpsOms.AllouerObjetStandard*** est appelée sans passer par un constructeur, l'objet n'est pas initialisé avec des valeurs par défaut pour ses propriétés.

5.4.2 Lecture des propriétés statiques d'un objet

La lecture des propriétés d'un objet s'effectue via l'***Application Program Interface*** (**API**) ***UpsOms.RechercherObjet***. Voici un exemple :

```
TypUpsVmAdresse AdresseObjet;
TypUpsVmUnsignedLong NumeroVerrou;
TypUpsVmPointeurDonnees Objet;
TypUpsVmPointeurDonnees Implementation;

...
Objet=(*MonModIntTrtUpsOms->RechercherObjet)(Session, &AdresseObjet,
   &NumeroVerrou, VR_ObjetLecturePartagee, &Implementation);
if (!Objet)
   {
   ...
   }
...
(*MonModIntTrtUpsOms->LibererVerrou)(Session, NumeroVerrou, 0);
...
```

Texte 3 – Exemple de lecture des propriétés statiques d'un objet

Le mode de verrouillage d'un objet standard est le suivant :

- **VR_LecturePartagee**.
  L'objet est verrouillé en lecture partagée. Les autres tâches peuvent y accéder simultanément en lecture partagée seulement.
- **VR_LectureExclusive**.
  L'objet est verrouillé en lecture exclusive. Les autres tâches ne peuvent pas y accéder simultanément, ni en lecture et ni en écriture.
- **VR_EcriturePartagee**.
  L'objet est verrouillé en écriture partagée. Les autres tâches peuvent y accéder simultanément en lecture partagée ou en écriture partagée.
- **VR_EcritureExclusive**.
  L'objet est verrouillé en écriture exclusive. Les autres tâches ne peuvent pas y accéder simultanément, ni en lecture et ni en écriture.

Si l'objet n'existe pas, ***UpsOms.RechercherObjet*** retourne ***NULL***. D'autre part, l'adresse de l'implémentation est optionnelle.

Lors de la recherche d'un objet, il est possible que celui ne soit pas accessible pour une des raisons suivantes :

- L'objet est déjà verrouillé par une autre tâche dans un mode incompatible.
  Auquel cas, la tâche courante est suspendue en attendant. Elle sera réveillée quand l'objet sera disponible.
- L'objet est paginé dans un fichier d'échange.
  Auquel cas, ***Up ! Object Management System*** va libérer un cache pour charger le segment dans lequel est conservé l'objet en mémoire physique. En attendant, la tâche courante peut être suspendue s'il existe une tâche pour activer le fichier d'échange.

Quand les propriétés statiques de l'objet ont fini d'être utilisées, il faut relâcher l'objet par l'appel à ***UpsOms.LibererVerrou***..

a Quand un objet a été modifié, il faut le signaler à ***Up ! Object Management System*** par l'appel à soit :

- ***UpsOms.ObjetEstModifie***.
  Cela permet d'éviter de faire circuler un témoin de modification parmi les paramètres d'appel.
- ***UpsOms.LibererVerrou***.
  Cela est le moyen le plus rapide pour ***Up ! Object Management System***.

M Oublier le libérer un verrou peut mener à un interblocage ou à une rareté en ressources.

M Oublier de signaler qu'un objet a été modifié peut mener à perdre les modifications apportées, lors d'une pagination ou lors du transport de l'objet sur le réseau.

a Pour tester si la tâche courante peut accéder à un objet sans la suspendre, il faut utiliser l'***Application Program Interface*** (**API**) ***UpsOms.RechercherObjetSansBloquer*** qui est non bloquante.

5.4.3 Utilisation d'un objet

Up ! Object Management System comptabilise le nombre de références sur un objet.

Quand un objet est référencé par un autre objet, par une variable ou par un champ d'une structure, il faut le signaler à ***Up ! Object Management System***. Cela s'effectue par l'appel à l'***Application Program Interface*** (***API***) ***UpsOms.UtiliserObjet***.

Quand un objet n'est plus référencé par un autre objet, par une variable ou par un champ d'une structure, il faut le signaler à ***Up ! Object Management System***. Cela s'effectue par l'appel à l'***Application Program Interface*** (***API***) ***UpsOms.NePlusUtiliserObjet***.

Voici un exemple :

```
TypUpsVmAdresse *AdresseObjet;
TypUpsVmAdresse AdresseObjet2;

...
(*MonModIntTrtUpsOms->UtiliserObjet)(Session,
   (*MonModIntTrtUpsOms->CopierAdresse)(&AdresseObjet2, AdresseObjet));
...
(*MonModIntTrtUpsOms->NePlusUtiliserObjet)(Session, &AdresseObjet2);
(*MonModIntTrtUpsOms->AdresseANul)(&AdresseObjet2);
...
```

Texte 4 – Exemple d'utilisation d'un objet

5.4.4 Destruction d'un objet

Up ! Object Management System comptabilisant le nombre de références sur un objet, il sait quand un objet n'est plus utilisé. De ce fait, il appelle alors le destructeur de l'objet, s'il existe, puis libère la zone mémoire correspondante qui sera disponible pour une éventuelle prochaine création.

M Pour libérer une liste circulaire, il est nécessaire de casser un maillon sinon chaque maillon comporte toujours deux références, ce qui conduirait à une perte mémoire.

5.4.5 Autres caractéristiques d'un objet

5.4.5.1 Appartenance

Un objet peut appartenir à un utilisateur particulier qui peut en avoir l'usage exclusif au travers des habilitations. La gestion des usages et des habilitations est du ressort d'***Up ! Security Management System***.

La fonction ***UpsOms.LireObjetAppartenance*** permet de lire la référence vers l'objet représentant le propriétaire de l'objet particulier. La procédure ***UpsOms.EcrireObjetAppartenance*** permet d'écrire la référence vers l'objet représentant le propriétaire de l'objet particulier.

5.4.5.2 Propriétés dynamiques

Un objet peut posséder des propriétés dynamiques en plus de ses propriétés statiques qui sont mémorisées dans une liste de paires (nom de la propriété, valeur de la propriété). Les propriétés dynamiques sont gérées au travers d'***Application Program Interfaces*** (**API**) spécifiques d'***Up ! Kernel***.

La fonction ***UpsOms.LireObjetListeProprietesDynamiques*** permet de lire la référence vers la liste des propriétés d'un objet particulier. La procédure ***UpsOms.EcrireObjetListeProprietesDynamiques*** permet d'écrire la référence vers la liste des propriétés d'un objet particulier.

Quand un objet est détruit, ***Up ! Object Management System*** détruit automatiquement la liste de propriétés dynamiques associée.

5.5 Cycle de vie d'un objet non standard

Un objet non standard est géré par son type en terme de cycle de vie. La caractéristique est que deux objets de son type de rattachement n'ont pas la même taille. Les segments ne sont donc pas formatés en enregistrements de taille fixe.

En ce cas, le type comporte son propre mécanisme d'allocation et de libération de mémoire pour ses objets au sein d'un ou de plusieurs segments qu'il utilise comme bon lui semble nécessaire.

Un objet non standard partiellement formaté est du type ***Buffer***. Il est utilisé pour les objets de type ***Binaire*** et de type ***Caractere***. Son usage est présenté dans la suite de cette section.

Pour mémoriser une référence vers un objet de type **Buffer**, il y a le type ***TypUpsVmExtension***.

5.5.1 Création d'un objet de type Buffer

La création d'un objet de type ***Buffer*** s'effectue via l'***Application Program Interface*** (**API**) ***UpsOms.AllouerBuffer*** qui est appelée par un des constructeurs du type de l'objet.

Voici un exemple :

```
TypUpsVmAdresse AdresseObjet;
TypUpsVmLong IndexBuffer;
TypUpsVmLong NbBuffers;
TypUpsVmUnsignedLong VerrouObjet;

...
if (!(*MonModIntTrtUpsOms->AllouerBuffer)(Session, CO_TailleObjet,
   &AdresseObjet, &IndexBuffer, &NbBuffers, &VerrouObjetLibre))
   {
   ...
   }
...
```

Texte 5 – Exemple de création d'un objet

Si un nouvel objet ne peut être créé, le résultat est ***NULL***.

M Quand ***UpsOms.AllouerBuffer*** est appelée sans passer par un constructeur, l'objet n'est pas initialisé avec des valeurs par défaut pour ses propriétés.

5.5.2 Lecture d'un objet de type Buffer

La lecture des propriétés d'un objet s'effectue via l'***Application Program Interface*** (**API**) ***UpsOms.CalculerBuffer***. Voici un exemple :

```
TypUpsVmAdresse AdresseObjet;
TypUpsVmLong IndexBuffer;
TypUpsVmLong NbBuffers;
TypUpsVmUnsignedLong NumeroVerrou;
TypUpsVmPointeurDonnees Objet;

...
Objet=(*MonModIntTrtUpsOms->CalculerBuffer)(Session, &AdresseObjet,
   IndexBuffer, NbBuffers, VR_TousLesBlocs, &NumeroVerrou);
if (!Objet)
   {
   ...
   }
...
(*MonModIntTrtUpsOms->LibererVerrou)(Session, NumeroVerrou, 0);
...
```

Texte 6 – Exemple de lecture des propriétés statiques d'un objet

Le mode de verrouillage d'un objet de type ***Buffer*** est le suivant :

- **VR_TousLesBlocs**.
  Tous les blocs mémoire du segment sont verrouillés en accès exclusif.

Quand les propriétés statiques de l'objet ont fini d'être utilisées, il faut relâcher l'objet par l'appel à ***UpsOms.LibererVerrou***..

a Quand un objet a été modifié, il faut le signaler à ***Up ! Object Management System*** par l'appel à soit :

- ***UpsOms.ObjetEstModifie***.
  Cela permet d'éviter de faire circuler un témoin de modification parmi les paramètres d'appel.
- ***UpsOms.LibererVerrou***.
  Cela est le moyen le plus rapide pour ***Up ! Object Management System***.

M Oublier le libérer un verrou peut mener à un interblocage ou à une rareté en ressources.

M Oublier de signaler qu'un objet a été modifié peut mener à perdre les modifications apportées, lors d'une pagination ou lors du transport de l'objet sur le réseau.

5.5.3 Utilisation d'un objet de type Buffer

Le décompte des références sur un objet de type ***Buffer*** est à la charge de celui qui l'utilise.

Les ***Application Program Interface*** (***API***) ***UpsOms.UtiliserObjet*** et ***UpsOms.NePlusUtiliserObjet*** ne doivent pas être utilisées.

5.5.4 Destruction d'un objet de type Buffer

Up ! Object Management System ne comptabilisant pas le nombre de références sur un objet de type ***Buffer***, il faut lui indiquer quand il peut être détruit. Cela s'effectue par l'appel à l'***Application Program Interface*** (***API***) ***UpsOms.LibererBuffer***.

Voici un exemple :

```
TypUpsVmAdresse AdresseObjet;
TypUpsVmLong IndexBuffer;
TypUpsVmLong NbBuffers;

...
(*MonModIntTrtUpsOms->LibererBuffer)(Session, &AdresseObjet,
   IndexBuffer, NbBuffers);
...
```

Texte 7 – Exemple de lecture des propriétés statiques d'un objet

5.6 Objets particuliers

Il existe des objets qui ont une sémantique particulière pour ***Up ! Object Management System***.

5.6.1 Objet Type

Un objet de type ***Type*** se crée au moyen de l'***Application Program Interface*** (**API**) ***UpsOms.CreerObjetType***. Ses paramètres sont les suivants :

- ***AdresseObjetModule***.
  Adresse de l'objet représentant le module déclarant le type.
- ***AdresseObjetType***.
  Adresse de l'objet représentant le type. Il s'agit d'un paramètre de sortie.
- ***Allouer***.
  Fonction d'allocation des objets du type. Par défaut, l'allocateur standard d'***Up ! Object Management System*** est utilisé.
- ***Appartenance***.
  Si ***Vrai***, ce type supporte l'appartenance de l'objet à un utilisateur particulier.
- ***Corba***.
  Si ***Vrai***, ce type est en fait une encapsulation d'une interface ***Corba***.
- ***DCom***.
  Si ***Vrai***, ce type est en fait une encapsulation d'une interface ***DCom***.
- ***EnteteMethodes***.
  En-tête de la table des méthodes surclassant celles du type ***Objet***.
- ***Entrepot***.
  Entrepôt par défaut de conservation des objets. Par défaut, il s'agit de l'entrepôt ***Système***.
- ***EstUneInterface***.
  Si ***Vrai***, ce type est en fait une interface.
- ***EstUneRessource***.
  Si ***Vrai***, ce type encapsule une ressource.
- ***FichierEchange***.
  Fichier d'échange par défaut de conservation des objets. Par défaut, il s'agit du fichier d'échange par défaut de l'entrepôt par défaut.
- ***Java***.
  Si ***Vrai***, ce type est en fait une encapsulation d'une classe ***Java***.
- ***NbObjets***.
  Pour le ramasse-miettes, taille d'un paquet d'allocation en nombre d'objets.

Si 1 est transmis, l'allocation par paquet n'est pas utilisée. Auquel cas, les objets sont alloués un à un en utilisant le type ***Buffer***.

- ***NomModule***.
  Nom du module déclarant le type. Par exemple *"UpsKrn"*.
- ***NomType***.
  Nom du type. Par exemple *"Caractere"*.
- ***ProprietesDynamiques***.
  Si ***Vrai***, ce type accepte les propriétés dynamiques.
- ***Quota***.
  Si Vrai, ce type supporte les quotas.
- ***Session***.
  Du fait de la normalisation des appels.
- ***SupporteHabilitation***.
  Si ***Vrai***, ce type supporte les habilitations.
- ***TailleObjet***.
  Taille d'un objet standard de ce type.
- ***Transaction***.
  Si ***Vrai***, ce type est géré selon le mode transactionnel.

5.6.2 Objet Entrepôt

Un objet de type ***Entrepot*** est créé par l'appel à l'***Application Program Interface*** (**API**) ***UpsOms.CreerObjetEntrepot***. Voici ses paramètres :

- ***AdresseObjet***.
  Adresse de l'objet représentant l'entrepôt. Il s'agit d'une valeur de sortie.
- ***NumeroEntrepot***.
  Numéro de l'entrepôt dans la configuration du noyau d'exécution.
- ***Session***.
  Du fait de la normalisation des appels.

L'entrepôt ***Système*** est créé automatiquement par ***Up ! Object Management System*** lors de son initialisation.

Il est possible de retrouver son adresse des ***Application Program Interface*** (**API**) ***UpsOms.RechercherObjetEntrepot*** ou ***UpsOms.RechercherEntrepot***.

5.7 Interfaces

Quand un objet correspond à une interface, il s'agit d'un objet virtuel référençant un objet réel. D'autre part, l'objet réel peut implémenter plusieurs interfaces, aussi celui-ci comporte autant de champs cachés qu'il implémente directement d'interfaces correspondant aux références vers les objets virtuels.

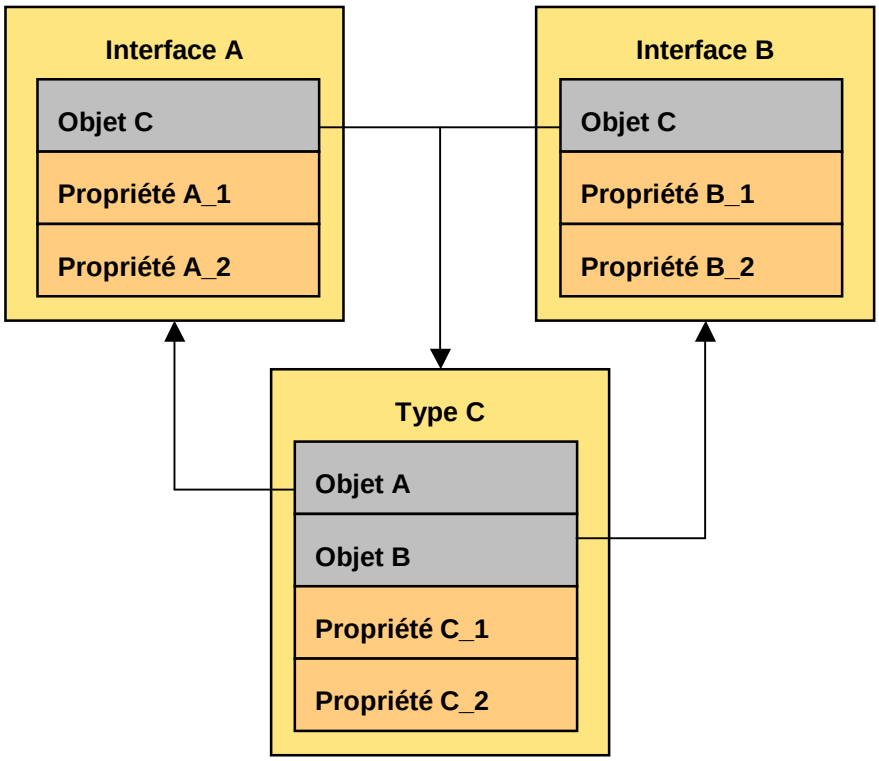

Diagramme 8 – Mise en oeuvre des interfaces

Pour retrouver l'objet réel depuis l'adresse de l'objet virtuel, il faut utiliser l'***Application Program Interface*** (**API**) ***UpsOms.RechercherObjetImplementant***. Voici un exemple :

```
TypUpsVmAdresse *AdresseObjetVirtuel;
TypUpsVmUnsignedLong NumeroVerrou;
TypUpsVmAdresse *AdresseObjetReel;

...
AdresseObjetReel=(*MonModIntTrtUpsOms->RechercherObjetImplementant)(Session,
   AdresseObjetVirtuel, NumeroVerrou, VR_ObjetEcriturePartagee);
if (!AdresseObjetReel)
   {
   ...
   }
...
(*MonModIntTrtUpsOms->LibererVerrou)(Session, NumeroVerrou, 0);
...
```

Texte 9 – Exemple de lecture de l'objet réel implémentant une interface

Le type implémentant l'interface doit changer la table des méthodes de chaque objet virtuel créé dans un champ caché. Cela s'effectue par l'appel à l'***Application Program Interface*** (**API**) ***UpsOms.SurclasserObjet***. Voici un exemple :

```
TypUpsVmAdresse *AdresseObjetReel;

...
(*MonModIntTrtUpsOms->SurclasserObjet)(Session, AdresseObjet, OffsetChampA,
   EnteteMethodesInterfaceA);
(*MonModIntTrtUpsOms->SurclasserObjet)(Session, AdresseObjet, OffsetChampB,
   EnteteMethodesInterfaceB);
...
```

Texte 10 – Exemple de surclassement de l'objet virtuel correspondant à une interface

5.8 Importation / exportation

Up ! Object Management System comporte un mécanisme d'archivage d'informations permettant de réaliser des importations et des exportations dans un format propriétaire non décrit dans ce document.

Ce mécanisme est utiliser en interne par ***Up ! Virtual Technical Machine*** pour :

- Le mécanisme de persistance.
- Le mécanisme de transmission de paramètres pour les appels distants gérés par ***Up ! Network***.

5.8.1 Exportation

Une tâche peut réaliser une exportation indépendamment des travaux réalisés par les autres tâches. Une exportation se débute par l'appel à l'***Application Program Interface*** (**API**) ***UpsOms.DebuterExportation*** dont voici les paramètres :

- ***Ecrire***.
  Procédure d'écriture d'un tampon binaire dans un flux.
- ***EstImplicite***.
  Si ***Vrai***, il s'agit d'une exportation implicite provoquée par ***Up ! Network***.
- ***NumeroNoeud***.
  Numéro du nœud réalisant l'exportation.
- ***NumeroServeur***.
  Numéro du serveur réalisant l'exportation.
- ***OptimiserExportation***.
  Si ***Vrai***, il est possible d'optimiser l'importation pour les objets des entrepôts protégés ou publics. Sauf pour le cas particulier d'***Up ! Network***, le paramètre doit valoir ***Faux***.

  Les ***Application Program Interface*** (**API**) suivantes sont alors opérationnelles :
- ***ExporterAdresseObjet***.
  Pour exporter l'adresse d'un objet et non l'objet.
- ***ExporterBinaire***.
  Pour exporter un tampon sans interpréter son contenu.
- ***ExporterEntier***.
  Pour exporter un nombre entier.
- ***ExporterObjet***.
  Pour exporter un objet connaissance son adresse.

- ***ExporterReel***.
  Pour exporter un nombre réel.
- ***ExporterVersionRevision***.
  Pour écrire la version-révision du flux.

Afin d'éviter d'exporter plusieurs fois un même objet, ce qui augmente le volume du flux et ce qui peut provoquer des bouclages, il est possible de savoir si un objet a déjà été exporté via l'appel à l'***Application Program Interface*** (**API**) ***UpsOms.ObjetDejaExporte***.

L'exportation se termine par l'appel à l'***Application Program Interface*** (**API**) ***UpsOms.TerminerExportation***.

M Les objets encapsulants les ressources ne peuvent pas être intégralement exportées par définition – la partie ressource. Aussi, pour une importation implicite, seule l'adresse de l'objet est exportée.

5.8.2 Importation

Une tâche peut réaliser une importation indépendamment des travaux réalisés par les autres tâches. Une importation se débute par l'appel à l'***Application Program Interface*** (**API**) ***UpsOms.DebuterImportation*** dont voici les paramètres :

- ***AdresseObjetEntrepot***.
  Adresse de l'objet représentant l'entrepôt dans lequel les objets vont être créés. Par défaut, il s'agit de l'entrepôt ***Système***.
- ***EstImplicite***.
  Si ***Vrai***, il s'agit d'une exportation implicite provoquée par ***Up ! Network***.
- ***Lire***.
  Fonction de lecture d'un tampon binaire dans un flux.

Les ***Application Program Interface*** (**API**) suivantes sont alors opérationnelles :

- ***ImporterAdresseObjet***.
  Pour importer l'adresse d'un objet et non l'objet.
- ***ImporterBinaire***.
  Pour importer un tampon sans interpréter son contenu.
- ***ImporterEntier***.
  Pour importer un nombre entier.
- ***ImporterObjet***.
  Pour importer un objet connaissance son adresse.
- ***ImporterReel***.
  Pour importer un nombre réel.
- ***ImporterVersionRevision***.
  Pour lire la version-révision du flux.

Afin d'éviter d'exporter plusieurs fois un même objet, ce qui augmente le volume du flux et ce qui peut provoquer des bouclages, il est obligatoire de signaler qu'un un objet a déjà été importé via l'appel à l'***Application Program Interface*** (**API**) ***UpsOms.AjouterObjetImporte***.

L'exportation se termine par l'appel à l'***Application Program Interface*** (**API**) ***UpsOms.TerminerImportation***.

M Les objets encapsulants les ressources ne peuvent pas être intégralement importés par définition – la partie ressource. Aussi, pour une importation non implicite, la partie ressource doit être reconstruite par l'appel à la méthode **AllouerRessource**.

5.8.3 Persistance

& La **persistance** permet de conserver les informations manipulées par le programme en l'état lors de son arrêt, de la sorte qu'il soit en mesure de continuer son exécution comme s'il ne se serait pas arrêté.

L'écriture du fichier de persistance est réalisée par l'appel à l'***Application Program Interface*** (**API**) ***UpsOms.EnregistrerPersistanceModules***.

La lecture du fichier de persistance est réalisée par l'appel à l'***Application Program Interface*** (**API**) ***UpsOms.DebuterChargementPersistance***. Pour chaque module, le chargement s'effectue de la sorte :

- Appel à ***UpsOms.DebuterChargementPersistanceModule***.
- Importation de chaque information propre au module.
  Pour chaque objet représentant un type, il est nécessaire de rétablir la table des méthodes par l'appel à ***UpsOms.MiseAJourEnteteMethodesObjets***.
- Appel à ***UpsOms.TerminerChargementPersistanceModule***.

La lecture du fichier de persistance est réalisée par l'appel à l'***Application Program Interface*** (**API**) ***UpsOms.TerminerChargementPersistance***.

5.9 Cohérence des données

Up ! Object Management System gère les accès concurrents aux objets du fait du noyau multi-tâches. Cependant, le modèle métier d'une entreprise nécessite de maintenir les informations cohérentes. De ce fait, ***Up ! Object Management System*** intègre quatre mécanismes classiques des bases de données relationnelles :

- Les contraintes.
- Les transactions.
- Les photographies.
- Les journaux.

Ces mécanismes n'impactent que les objets dont les types sont déclarés comme étant transactionnels, afin d'économiser les ressources.

5.9.1 Contraintes

& Une **contrainte** est une règle de gestion devant être vérifiée, faute de quoi les informations ne sont pas fonctionnellement cohérentes. Elle agit comme un réflexe sur le modèle de données.

Pour les types non transactionnels, les contraintes sont vérifiées à la volée, au fur et à mesure des créations, des modifications et des suppressions des informations. Une contrainte est mémorisée par l'appel à l'***Application Program Interface*** (**API**) ***UpsOms.MemoriserContrainte***.

Pour les types transactionnels, les contraintes sont vérifiées au moment de la validation d'une transaction.

5.9.2 Transactions

&

Une **transaction** permet à une tâche de masquer les modifications qu'elle est en train d'effectuer sur les propriétés statiques ou dynamiques des objets – création, modification et suppression – à la vue des autres tâches. Les transactions peuvent être imbriquées.

Quand les modifications sont cohérentes, cette tâche les rend visibles aux autres tâche en validant la transaction. Cette action est indivisible. Une transaction est validée par l'appel à l'***Application Program Interface*** (**API**) ***UpsOms.ValiderTransaction***.

Quand il existe un dysfonctionnement, cette tâche peut annuler ses modifications en invalidant la transaction. La base d'objets revient alors à son état initial. Une transaction est invalidée par l'appel à l'***Application Program Interface*** (**API**) ***UpsOms.InvaliderTransaction***.

5.9.3 Photographies

&

Pour réaliser des calculs cohérents, une tâche peut avoir besoin de prendre une **photographie** de l'ensemble des informations. Une photographie est initialisée par l'appel à l'***Application Program Interface*** (**API**) ***UpsOms.Photographier***.

Tant que la tâche demande à maintenir la photographie, toutes les modifications apportées par les transactions validées par les autres tâches sont mises de coté pour cette première. Une transaction est terminée par l'appel aux ***Application Program Interface*** (**API**) ***UpsOms.ValiderTransaction*** ou ***UpsOms.InvaliderTransaction***.

5.9.4 Journaux

&

Pour les programmes persistants regroupant des informations critiques, le mécanisme de **journaux** permet d'assurer qu'aucune donnée ne sera perdue et qu'aucune incohérence ne sera introduite dès lors qu'il existe un dysfonctionnement fatal. Ce mécanisme est mis en oeuvre quand le programme est persistant et qu'il existe des transactions.

La synchronisation des journaux et des entrepôts est réalisée périodiquement par ***Up ! Object Management System***. Elle peut être provoquée par l'appel à l'***Application Program Interface*** (**API**) ***UpsOms.SynchroniserEntrepotsEtJournal***.

6 Modèle de données

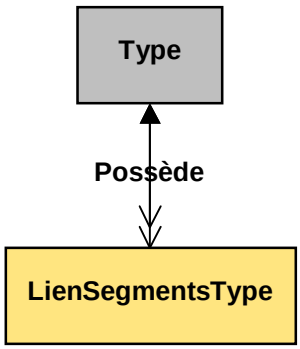

Diagramme 11 – Modèle physique des données publiques du module *Up ! Object Management System*

Toutes les entités suivantes sont décrites dans le fichier **upsoms.e** :

Entité.	Description.
LienSegmentsType.	Segments utilisés par un type de données pour conserver ses objets.
Type.	Modélisation d'un type.

Tableau 12 – Glossaire du modèle physique des données publiques du module *Up ! Object Management System*

7 Composants techniques

Le module ***Up ! Object Management System*** pour le projet ***Up ! Application System*** est constitué des composants suivants :

Fichiers du module.	
• Fichier **upsoms.e** – Définition des méthodes pour la gestion des objets. • Fichier **upsoms.h** – En-tête privé de ***Up ! Object Management System***. • Fichier **upsoms.def** – Exportation des symboles pour ***Windows***.	
Composants.	**upsoms0**.
Description.	
Interface entre ***Up ! Object Management System*** et ***Up ! Module***.	
Fichiers.	
• Fichier **upsoms0.cpp** – Fichier source. • Fichier **upsoms0.h** – En-tête privé de **upsoms0.cpp**. • Fichier **upsoms0.e** – En-tête protégé de **upsoms0.cpp**.	
Composants.	**upsoms1**.
Description.	
• Gestion des caches mémoire d'***Up ! Object Management System***. • Ramasse-miettes d'***Up ! Object Management System***.	
Fichiers.	
• Fichier **upsoms1.cpp** – Fichier source. • Fichier **upsoms1.h** – En-tête privé de **upsoms1.cpp**. • Fichier **upsoms1.e** – En-tête protégé de **upsoms1.cpp**.	
Composants.	**upsoms2**.
Description.	
• Utilitaire d'***Up ! Object Management System***. • Gestion de l'importation et de l'exportation d'objets.	
Fichiers.	
• Fichier **upsoms2.cpp** – Fichier source. • Fichier **upsoms2.h** – En-tête privé de **upsoms2.cpp**. • Fichier **upsoms2.e** – En-tête protégé de **upsoms2.cpp**.	
Composants.	**upsoms3**.
Description.	
• Gestion des verrous d'***Up ! Object Management System***. • Gestion des sessions.	
Fichiers.	
• Fichier **upsoms3.cpp** – Fichier source. • Fichier **upsoms3.h** – En-tête privé de **upsoms3.cpp**. • Fichier **upsoms3.e** – En-tête protégé de **upsoms3.cpp**.	

Composants.	**upsoms4**.
Description.	
Initialisation d'***Object Management System***.	
Fichiers.	
• Fichier **upsoms4.cpp** – Fichier source. • Fichier **upsoms4.h** – En-tête privé de **upsoms4.cpp**. • Fichier **upsoms4.e** – En-tête protégé de **upsoms4.cpp**.	
Composants.	**upsoms5**.
Description.	
• Tâche **Archivage**. • Tâche **EnregistrementJournal**. • Tâche **FichierEchange**. • Tâche **Moniteur**. • Tâche **RetasserTousLesTypes**. • Tâche **RetasserType**. • Tâche **Scruptateur**. • Tâche **Statistiques**.	
Fichiers.	
• Fichier **upsoms5.cpp** – Fichier source. • Fichier **upsoms5.h** – En-tête privé de **upsoms5.cpp**. • Fichier **upsoms5.e** – En-tête protégé de **upsoms5.cpp**.	
Composants.	**upsoms6**.
Description.	
Méthodes du type privé **Buffer**.	
Fichiers.	
• Fichier **upsoms6.cpp** – Fichier source. • Fichier **upsoms6.h** – En-tête privé de **upsoms6.cpp**. • Fichier **upsoms6.e** – En-tête protégé de **upsoms6.cpp**.	
Composants.	**upsoms7**.
Description.	
• Méthodes du type privé **ImageAvant**. • Gestion des images avant. • Gestion des transactions. • Gestion des photographies.	
Fichiers.	
• Fichier **upsoms7.cpp** – Fichier source. • Fichier **upsoms7.h** – En-tête privé de **upsoms7.cpp**. • Fichier **upsoms7.e** – En-tête protégé de **upsoms7.cpp**.	
Composants.	**upsoms99**.
Description.	

Interface entre ***Up ! Object Management System*** et ***Up ! Kernel***.
Fichiers.
• Fichier **upsoms99.cpp** – Fichier source. • Fichier **upsoms99.h** – En-tête privé de **upsoms99.cpp**. • Fichier **upsoms99.e** – En-tête protégé de **upsoms99.cpp**.

Tableau 13 – Composants techniques du module

Fin de document