	Spécification technique du module	**Date rédaction : 14 mars 2004.**
	Diffusion restreinte	**Date validation :**
Référence : UpComp-UpsLg1-000003-A Spécification technique du module UpsLg1.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.**
Instruction implicite	Voir page 11.
Preuve de programme	Voir page 9.

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-Upslg1-000002	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-UspKrn-000003	Plan d'écriture d'un module.

3 Introduction

3.1 Objet du document

L'objet de ce document est de décrire le contenu technique du module logiciel ***Up ! Lg1*** 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 ! Lg1*** est de :

- Modéliser les concepts évolués.
  Par extension ou adaptation des concepts de base, de la sorte à avoir une offre linguistique au moins équivalente à celles des langages de programmation les plus évolués :
  - Types polymorphes.
  - Type évolués tel ***ArbreBinaire***, ***Liste***, ***Reference*** et ***Tableau***.
  - Exceptions.
  - Files d'attente.
  - Segments.
  - Opérateur **||** sur les appels.
  - Expressions **`Dans`** et **`Entre`**.
  - Expressions de groupe **`EcartType`**, **`Max`**, **`Min`**, **`Moyenne`**, **`Somme`** et **`Variance`**.
  - Instruction **`Selon`** ... **`Fin Selon`** et **`AttraperException`**.
- Vérifier la consistance des programmes.
  Par application d'algorithmes de preuve de programme. Quand une anomalie potentielle est détectée, alors :
  - Elle est signalée si elle est décidable.
  - Un code de contrôle dynamique est ajouté si elle est indécidable.
- Optimiser les programmes.
  En factorisant le code redondant et en mémorisant les résultats intermédiaires.
- Préparer la génération.
  En ajoutant du code caché de la sorte à passer d'un langage de cinquième génération à un langage de quatrième puis de troisième génération.

Le module logiciel ***Up ! Lg1*** est une extension d'***Up ! 5GL***. A son tour, ***Up ! Lg1*** peut être étendu selon le même principe – fonction de rappels et tables d'extension.

4.2 Paramètres

Up ! Lg1 ne possède aucun paramètre.

4.3 Particularités

4.3.1 Compilation

Néant.

4.3.2 Exécution

Néant.

4.4 Application Program Interfaces

Néant.

5 Choix techniques

5.1 Extension des définitions linguistiques évoluées

Les définitions linguistiques évoluées peuvent être étendues via un ou plusieurs modules secondaires dont les caractéristiques sont les suivantes, dans la limite de ***CO_NbLangages2Max*** extensions :

- L'interface de données doit être une extension de ***Lg2XDonnees***.
  Ceci permet de spécifier à ***Up ! Lg1*** la taille de l'espace mémoire privé nécessaire pour l'extension de chaque concept évolué.
  Quand aucun espace mémoire n'est pas nécessaire, la taille à spécifier est ***0***.
- L'interface de traitements doit être une extension de ***Lg2XTraitements***.
  Ceci permet de spécifier à ***Up ! Lg1*** les fonctions de rappels nécessaires pour étendre les algorithmes sur chaque concept évolué.
  Quand une fonction de rappel n'est pas nécessaire, la fonction de rappel doit être ***UpsKrn.AppelImpossibleApiNull***.

Chaque concept comporte une table d'extension dénommée ***TableExtension*** conservant l'espace mémoire privé nécessaire à un module particulier. L'index dans cette table est le numéro d'extension du module donné par ***Lg2XDonnees.InterfaceLangage1.EnteteOption.NumeroOption***.

Voici le schéma d'une extension d'un concept de base :

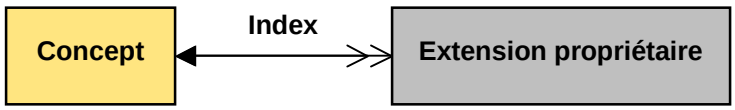


Diagramme 1 – Extension d'un concept évolué

M Les extensions possibles des concepts ne sont pas représentées dans le modèle de données.

5.2 Alignement des structures de données

Les structures de données ont un alignement imposé constant quel que soit le module et quelle que soit la plate-forme. Cette norme permet de calculer l'offset d'une propriété d'un type ou d'une interface en fonction de ce qui la précède.

Voici les alignements retenus :

Type de données.	Alignement logique.	Alignement physique.
TypUpsVmAdresse.	`sizeof`(***TypUpsVmLong***).	4 octets.
TypUpsVmChar.	`sizeof`(***TypUpsVmChar***).	1 octet.
TypUpsVmDouble.	`sizeof`(***TypUpsVmLong***).	4 octets.
TypUpsVmLong.	`sizeof`(***TypUpsVmLong***).	4 octets.
TypUpsVmPointeurDonnees.	`sizeof`(***TypUpsVmLong***).	4 octets.
TypUpsVmPointeurTraitements.	`sizeof`(***TypUpsVmLong***).	4 octets.
TypUpsVmShort.	`sizeof`(***TypUpsVmShort***).	2 octets.
TypUpsVmUnsignedChar.	`sizeof`(***TypUpsVmChar***).	1 octet.
TypUpsVmUnsignedLong.	`sizeof`(***TypUpsVmLong***).	4 octets.

Type de données.	Alignement logique.	Alignement physique.
TypUpsVmUnsignedShort.	`sizeof`(*TypUpsVmShort*).	2 octets.

Tableau 2 – Alignement des structures de données

5.3 Factorisation des expressions

Dans un bloc de code – le corps d'un appel, le corps d'une boucle, le corps d'un test, etc. –, ***Up ! Lg1*** tente de reconnaître des expressions ou des parties d'entre elles déjà calculées et qui ne sont pas modifiées de la sorte à :

- Préparer la détection des incohérences.
  Cela est expliqué dans la section suivante.
- Factoriser le code.
  Pour que l'exécution soit plus rapide.

  Voici un exemple de code non factorisé :

```
MaChaineA = "A" + MaChaineB + "C";
MaChaineC = "A" + MaChaineB;

MonTableau[I].MonObjet.A = 0;
MonTableau[I].MonObjet.B = Faux;
MonTableau[I].MonObjet.A = "";
```

Texte 3 – Exemple d'expression non factorisée

Voici le même exemple de code une fois factorisé :

```
Temporaire1 = "A" + MaChaineB;
MaChaineA = Temporaire1 + "C";
MaChaineC = Temporaire1;

Temporaire2 = MonTableau[I].MonObjet;
Temporaire2.A = 0;
Temporaire2.B = Faux;
Temporaire2.A = "";
```

Texte 4 – Exemple d'expression non factorisée

Une expression peut être factorisée si :

a

- Elle n'emploie que des fonctions avec l'inflexion `Optimiser`.
  Sinon elle réalise par convention un effet de bord.
- Elle n'emploie pas de variables globales dont l'accès en écriture est public.
  Sinon une autre tâche pourrait la modifier.

5.4 Preuve de programme

&

La **preuve de programme** consiste à vérifier avec minutie la sémantique du code écrit par l'utilisateur de la sorte à soit :

- Détecter une erreur de raisonnement.
  Dès la compilation.

- Vérifier à l'exécution les raisonnements.
  Par ajout du code caché pour qu'une erreur de sémantique soit signalée avant tout dysfonctionnement.

  Voici un exemple de code comportant des erreurs de raisonnement :

```
Procedure MaProcedure()
Variable
   I : Entier;
   J : Entier;
   K : Booleen;

Debut
I = 10;
J = 20;
Si I==0 Alors
// Le test est toujours faux.
   Ecran.Ecrire("coucou\n");
Fin Si

Pour J=1 JusquA 3 Faire
   Ecran.Ecrire("coucou\n");
   TantQue I<20 Et J<0 Faire
// Le test est toujours faux.
      Ecran.Ecrire("coucou\n");
   Fin TantQue
   Ecran.Ecrire("coucou\n");
Fin Pour
Fin Procedure
```

Texte 5 – Exemple de code comportant des erreurs de raisonnement

a

Le principe de la preuve de programme est de conserver un contexte d'exécution par un bloc de code – le corps d'un appel, le corps d'une boucle, le corps d'un test, etc. :

- Une liste de variables modifiées.
- Des listes d'assertions :
  - Les expressions toujours vraies.
    Suite à une affectation par exemple.
    La condition d'arrêt d'une boucle au début du corps de la boucle.
  - Les expressions toujours fausses.
    La condition d'arrêt d'une boucle suite au corps la boucle, si elle ne comporte pas d'instruction **`Arreter`**, par exemple.
    La condition d'un test dans une de ses alternatives par exemple.
- Des indicateurs de présence d'instructions pertinentes pour décidabilité la preuve de programme.
  `Arreter`, **`Continuer`**, etc.

5.5 Temporaires

Les temporaires sont utilisés pour :

- Passer des paramètres aux appels de procédure, de fonction et de méthode.
  Quand ils en possèdent.

- Récupérer le résultat d'une fonction ou d'une méthode fonctionnelle.
  Parce que le code produit par ***Up ! Compiler*** est en ***C--*** dont le niveau d'abstraction ***L3G***.
- Mémoriser le résultat d'expressions factorisées.
  Selon le principe décrit dans l'une des sections précédentes.

  La séquence d'usage d'un temporaire pour une expression élémentaire est la suivante :

1) Remplissage temporaire.
   Avec le résultat d'une expression pour un paramètre d'entrée et avec une valeur par défaut pour un paramètre de sortie. Au besoin, une conversion de type de données ou de nullité est réalisée, qui peut engendrer une exception en cas d'inconsistance.
2) Appel de la procédure, de la fonction ou de la méthode.
   Avec le temporaire pour les paramètres et éventuellement un temporaire pour le résultat.
3) Lecture du temporaire.
   Pour récupérer les valeurs des paramètres de sortie. Au besoin, une conversion de type de données ou de nullité est réalisée, qui peut engendrer une exception en cas d'inconsistance.

&

Up ! Lg1 ajoute des instructions cachées au code, appelées **instructions implicites**, avant et après chaque instruction explicite. Elles sont conservées dans deux listes dénommées ***InstructionsImplicitesAvant*** et ***InstructionsImplicitesApres***.

Quand les expressions se composent, du fait de la facilité apportée par le niveau d'abstraction ***L4G*** :

Expression1 **`OperateurA`** (Expression2 **`OperateurB`** Expression3)

a

L'ordre de remplissage des temporaires, d'appel et de lecture des temporaires dépend, d'une part, des parenthèses isolant une partie de l'expression, d'autre part, de la priorité des opérateurs.

5.6 Codifications

5.6.1 Types

La nature d'un type est définie par l'énuméré ***EnuUpsLg1Type*** :

- **TY_ArbreBinaire**.
  Pour une liste équivalente au type ***ArbreBinaire***.
- **TY_Liste**.
  Pour une liste équivalente au type ***Liste***.
- **TY_Reference**.
  Pour une liste équivalente au type ***Reference***.
- **TY_Selon**.
  Pour une liste équivalente au type ***Selon***.
- **TY_Tableau**.
  Pour une liste équivalente au type ***Tableau***.

5.6.2 Expressions

La nature d'une expression est définie par l'énuméré ***EnuUpsLg1Expression*** :

- **EX_Dans**.
  L'expression est **`Dans`**.

- **EX_Entre**.
  L'expression est **`Entre`**.
- **EX_Exception**.
  L'expression est **`Exception`**.
- **EX_FileDAttente**.
  L'expression est la mise en file d'attente d'un appel de procédure, de fonction ou de méthode.
- **EX_Groupe**.
  L'expression est un opérateur de groupe – **`EcartType`**, **`Max`**, **`Min`**, **`Moyenne`**, **`Somme`** et **`Variance`**.
- **EX_Tableau**.
  L'expression est la sélection d'un élément d'un tableau.
- **EX_Temporaire**.
  L'expression est la valeur du temporaire.
- **EX_Segment**.
  L'expression est un segment de données.
- **EX_Selection**.
  L'expression est la sélection de la partie polymorphe d'un type.

5.6.3 Index d'un tableau

La nature d'un index d'un tableau est définie par l'énuméré ***EnuUpsLg1IndexTableau*** :

- **TX_Entier**.
  L'index est un nombre entier.
- **TX_Enumere**.
  L'index est un énuméré.

5.6.4 Validité des expressions

L'état de validité des expressions, au cours de la factorisation du code, est défini par l'énuméré ***EnuUpsLg1ValiditeExpression*** :

- **VE_NonCalculee**.
  La validité ou l'invalidité de l'expression n'a pas encore été établi.
- **VE_Valide**.
  L'expression est valide. Son résultat déjà calculé peut être employé.
- **VE_Invalide**.
  L'expression est invalide. Son résultat déjà calculé ne peut plus être employé.

5.6.5 Cohérence des expressions

L'état de cohérence des expressions, au cours de la preuve de programme, est défini par l'énuméré ***EnuUpsLg1Coherence*** :

- **CO_Coherence**.
  L'expression est cohérente.
- **CO_Incoherence**.
  L'expression est incohérente.

- **CO_Erreur**.
  Une erreur interne est survenue lors du calcul de cohérence de l'expression.

5.6.6 Valeur d'une condition

La valeur d'une condition exprimée sous forme d'une expression, au cours de la preuve de programme, est définie par l'énuméré ***EnuUpsLg1Condition*** :

- **CD_Vrai**.
  La condition est toujours ***Vrai***.
- **CO_Faux**.
  La condition est toujours ***Faux***.
- **CD_NeSaitPas**.
  La condition ne peut être évalué.

5.6.7 Provenance des expressions

La provenance des expressions, au cours de la preuve de programme, est définie par l'énuméré ***EnuUpsLg1ProvenanceExpression*** :

- **PE_Contrainte**.
  L'expression est employée dans le calcul d'une contrainte.
- **PE_Initialisation**.
  L'expression est employée dans une fonction d'initialisation telle **`DebuterComposant`**.
- **PE_Instruction**.
  L'expression est employée dans le cas général.

5.6.8 Segments

La nature d'un segment est définie par l'énuméré ***EnuUpsLg1Segment*** :

- **SG_Prive**.
  Segment de données privé.
- **SG_Protege**.
  Segment de données protégé.
- **SG_Public**.
  Segment de données public.

5.6.9 Instructions

La nature d'une instruction est définie par l'énuméré ***EnuUpsLg1Instruction*** :

- **IN_Selon**.
  Instruction **`Selon`**.
- **IN_AttraperException**.
  Instruction **`AttraperException`**.
- **IN_Implicite**.
  Instruction **`Implicite`**.

5.6.10 Instructions implicites

La nature d'une instruction implicite est définie par l'énuméré ***EnuUpsLg1Implicite*** :

- Pour le contrôle de consistance :
  - **ll_TesterExpressionNonNulle**.
    Vérifie que la valeur d'une expression est non nulle.
  - **ll_TesterExpressionSelon**.
    Vérifie que la valeur d'une expression à sélectionner est dans l'intervalle de valeurs attendues.
  - **ll_TesterExpressionAlias**.
    Vérifie que la valeur d'une expression est conforme à la contrainte d'un type défini par l'instruction **`Alias`**.
- Pour la décomposition des instructions ***L4g*** en ***L3g*** :
  - **ll_AffecterTemporaire**.
    Affectation des champs d'un temporaire avant d'effectuer un appel comportant des paramètres d'entrée.
  - **ll_LireTemporaire**.
    Lecture de la valeur des champs d'un temporaire après avoir effectuer un appel comportant des paramètres de sortie.
  - **ll_CalculerEtSi**.
    Instruction partielle d'une cascade de **`EtSi`**.
  - **ll_CalculerOuSinon**.
    Instruction partielle d'une cascade de **`OuSinon`**.

5.6.11 Types de temporaires

Le type de temporaire est défini par l'énuméré ***EnuUpsLg1TypeTemporaire*** :

- **TT_AccesDistribue**.
  Temporaire servant à effectuer un appel distribué via ***Up ! Network***.
- **TT_Appel**.
  Temporaire pour passer les paramètres à un appel de procédure, de fonction ou de méthode dont la référence est inconnue. Il n'est pas possible de connaître le type de la structure de ses paramètres.
- **TT_PassageParametre**.
  Temporaire servant à transmettre les paramètres d'une procédure, d'une fonction ou d'une méthode.
- **TT_Persistant**.
  Temporaire utilisé dans un cas générique pour la mémorisation d'une expression factorisée.
- **TT_Normal**.
  Temporaire utilisé dans un cas générique pour la décomposition d'une expression.
- **TT_Postfixe**.
  Temporaire employé pour les opérateurs ++ ou -- employés en post fixé.
- **TT_Prefixe**.
  Temporaire employé pour les opérateurs ++ ou -- employés en préfixé.
- **TT_VerifierObjet**.
  Temporaire servant pour la conversion d'un objet d'un type dans un autre.

5.6.12 Objets

La nature d'un objet est définie par l'énuméré ***EnuUpsLg1Objet*** :

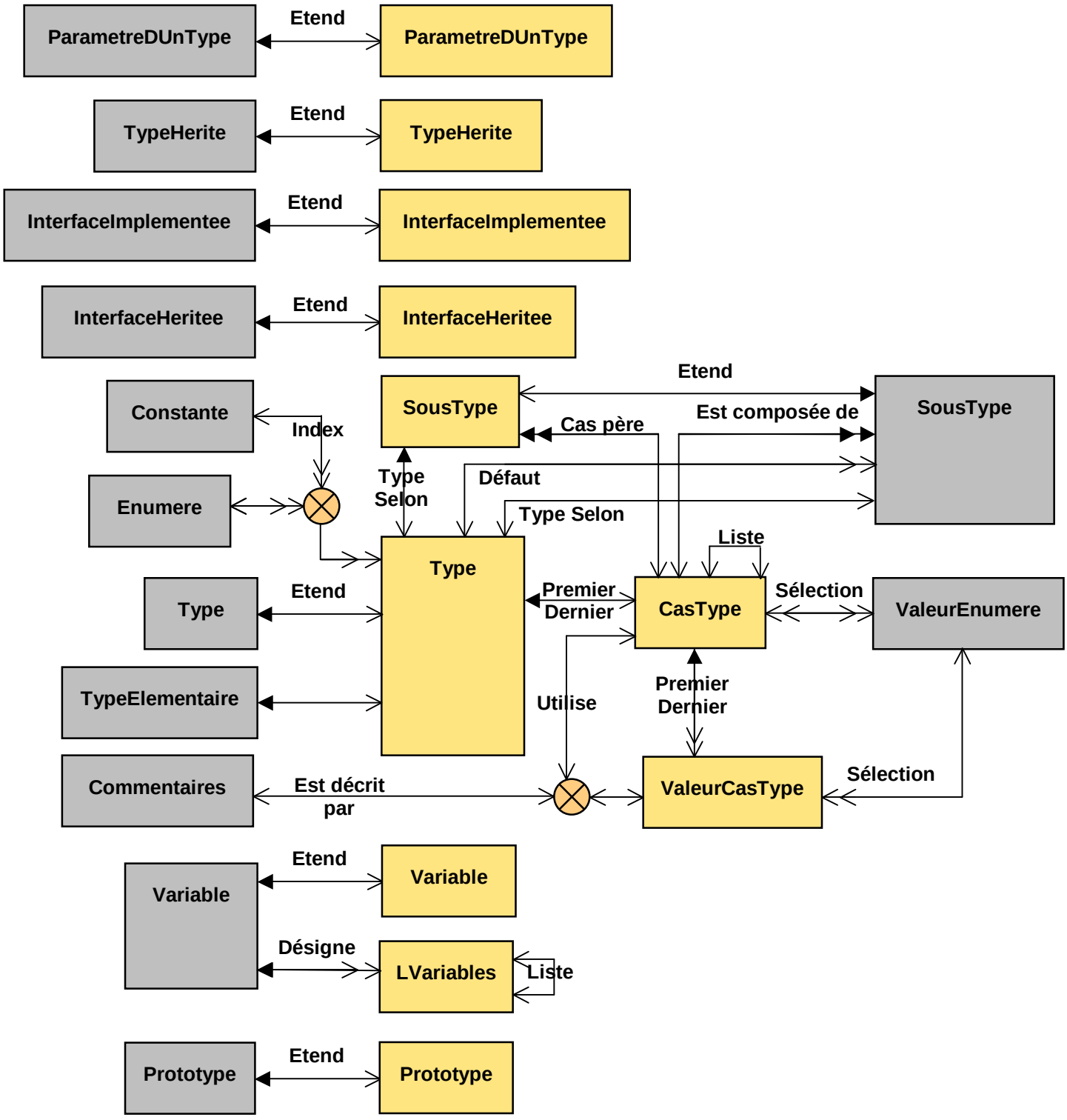
- **OB_Exception**.
  L'objet est une exception.
- **OB_FileDAttente**.
  L'objet est une file d'attente.
- **OB_Segment**.
  L'objet est un segment.

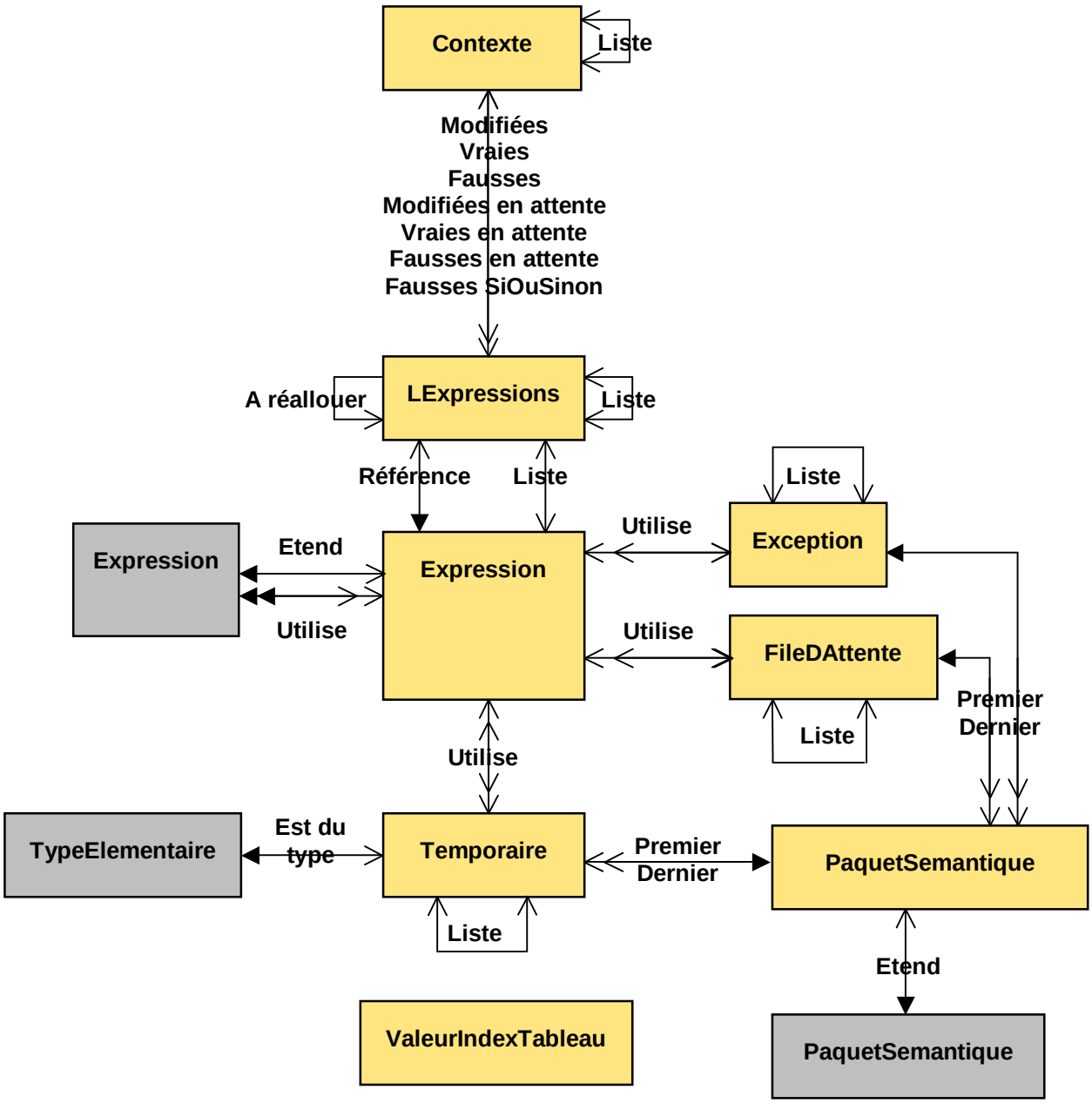
5.6.13 Exceptions

L'effet d'une exception sur les transactions est défini par l'énuméré ***EnuUpsLg1ExceptionSurTransaction*** :

- **ET_TransactionInchangee**.
  L'exception est sans effet.
- **ET_TransactionInvalidee**.
  La transaction de premier niveau de la tâche recevant l'exception est invalidée.
- **ET_TransactionToutesInvalidees**.
  Les transactions de la tâche recevant l'exception sont toutes invalidées.

6 Modèle de données





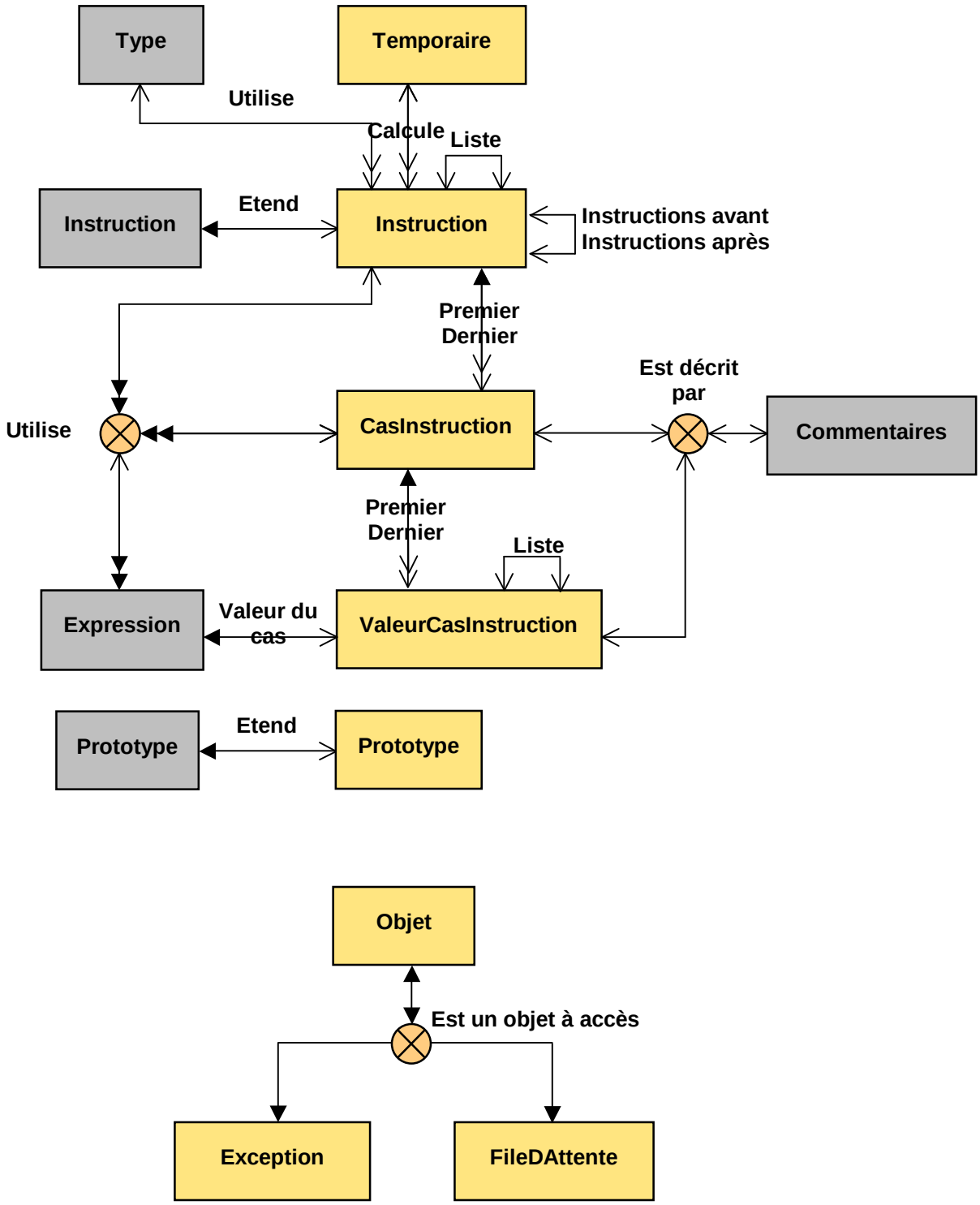


Diagramme 6 – Modèle physique des données publiques du module *Up ! Lg1*

M Le modèle de données utilise des allocateurs et des tables, à des fins d'optimisation, qui ne sont pas représentés sur le diagramme.

Toutes les entités suivantes sont décrites dans le fichier **upslg1.e** :

Entité.	Description.
CasInstruction.	Cas définissant une partie polymorphe d'un algorithme.
CasType.	Cas définissant une partie polymorphe d'un type.
Commentaires.	Commentaires sur une définition.
Constante.	Constante dont la valeur est énumérée, entière, réelle ou chaîne de caractères.
Contexte.	Contexte d'exécution regroupant des variables modifiées, des assertions vraies et des assertions fausses.
Enumere.	Enuméré comportant au moins une valeur énumérée.
Exception.	Exception correspondant à une erreur.
Expression.	Expression élémentaire.
FileDAttente.	File d'attente de traitement de travaux.
Instruction.	Instruction élémentaire.
InterfaceHeritee.	Interface héritée par une autre interface.
InterfaceImplementee.	Interface implémentée par un autre type.
LExpressions.	Liste d'expressions pointées.
LVariables.	Liste de variables pointées.
PaquetSemantique.	Paquet de définitions locales portant une sémantique.
ParametreDUnType.	Paramètre attendu d'un type.
Prototype.	Prototype d'un appel de procédure, de fonction ou de méthode.
SousType.	Paramètre d'un prototype ou propriété d'un type ou d'une interface.
Temporaire.	Temporaire de calcul.
Type.	Type ou interface.
TypeElementaire.	Type utilisé dans une déclaration.
TypeHerite.	Type hérité par un autre type.
ValeurCasInstruction.	Valeur d'un cas définissant une partie polymorphe d'un algorithme.
ValeurCasType.	Valeur d'un cas définissant une partie polymorphe d'un type.
ValeurEnumere.	Valeur d'un énuméré.
ValeurIndexTableau.	Valeur de l'index pour un tableau.
Variable.	Variable locale ou globale.

Tableau 7 – Glossaire du modèle physique des données publiques du module *Up ! Lg1*

7 Composants techniques

Le module ***Up ! Lg1*** pour le projet ***Up ! Application System*** est constitué des composants suivants :

Fichiers du module.	
• Fichier **upslg1.e** – Définition de ***Up ! Lg1***. • Fichier **upslg1.h** – En-tête privé de ***Up ! Lg1***. • Fichier **upslg1.def** – Exportation des symboles pour ***Windows***.	
Composants.	**upslg1_0**.
Description.	
Interface entre ***Up ! Lg1*** et ***Up ! Module***.	
Fichiers.	
• Fichier **upslg1_0.cpp** – Fichier source. • Fichier **upslg1_0.h** – En-tête privé de **upslg1_0.cpp**. • Fichier **upslg1_0.e** – En-tête protégé de **upslg1_0.cpp**.	
Composants.	**upslg1_1**.
Description.	
Utilitaires pour l'analyseur sémantique évolué.	
Fichiers.	
• Fichier **upslg1_1.cpp** – Fichier source. • Fichier **upslg1_1.h** – En-tête privé de **upslg1_1.cpp**. • Fichier **upslg1_1.e** – En-tête protégé de **upslg1_1.cpp**.	
Composants.	**upslg1_2**.
Description.	
• Gestion des types ***ArbreBinaire***, ***Liste***, ***Reference*** et ***Tableau***. • Gestion des parties polymorphes d'un type. • Calcul des offsets des propriétés d'un type ou d'une interface. • Construction de l'expression d'un tableau. • Transformation du type de l'objet générique d'un ***ArbreBinaire***, ***Liste***, ***Reference*** ou ***Tableau*** en le type définit dans l'instruction de haut niveau correspondant.	
Fichiers.	
• Fichier **upslg1_2.cpp** – Fichier source. • Fichier **upslg1_2.h** – En-tête privé de **upslg1_2.cpp**. • Fichier **upslg1_2.e** – En-tête protégé de **upslg1_2.cpp**.	
Composants.	**upslg1_3**.
Description.	

- Construction de l'expression **`ArbreBinaireDe`**.
- Construction de l'expression **`ListeDe`**.
- Construction de l'expression **`ReferenceDe`**.
- Construction de l'expression **`TableauDe`**.
- Construction de l'expression **`Max`**, **`Min`**, **`Somme`**, **`Moyenne`**, **`EcartType`** et **`Variance`**.
- Construction de l'instruction **`Selon`**.
- Construction de l'expression **`AttraperException`**.
- Importation des définitions des modules.

Fichiers.

- Fichier **upslg1_3.cpp** – Fichier source.
- Fichier **upslg1_3.h** – En-tête privé de **upslg1_3.cpp**.
- Fichier **upslg1_3.e** – En-tête protégé de **upslg1_3.cpp**.

Composants.	**upslg1_4**.

Description.

Services de cohérence pour les expressions.

Fichiers.

- Fichier **upslg1_4.cpp** – Fichier source.
- Fichier **upslg1_4.h** – En-tête privé de **upslg1_4.cpp**.
- Fichier **upslg1_4.e** – En-tête protégé de **upslg1_4.cpp**.

Composants.	**upslg1_5**.

Description.

Preuve de programme élémentaire pour les conditions des tests et des boucles.

Fichiers.

- Fichier **upslg1_5.cpp** – Fichier source.
- Fichier **upslg1_5.h** – En-tête privé de **upslg1_5.cpp**.
- Fichier **upslg1_5.e** – En-tête protégé de **upslg1_5.cpp**.

Composants.	**upslg1_6**.

Description.

Gestion des contextes – liste de variables modifiées, assertions vraies et assertions fausses.

Fichiers.

- Fichier **upslg1_6.cpp** – Fichier source.
- Fichier **upslg1_6.h** – En-tête privé de **upslg1_6.cpp**.
- Fichier **upslg1_6.e** – En-tête protégé de **upslg1_6.cpp**.

Composants.	**upslg1_7**.

Description.

Création des instructions implicites élémentaires.

Fichiers.

• Fichier **upslg1_7.cpp** – Fichier source. • Fichier **upslg1_7.h** – En-tête privé de **upslg1_7.cpp**. • Fichier **upslg1_7.e** – En-tête protégé de **upslg1_7.cpp**.	
Composants.	**upslg1_8**.
Description.	
• Gestion des instructions implicites composées. • Preuve de programme.	
Fichiers.	
• Fichier **upslg1_8.cpp** – Fichier source. • Fichier **upslg1_8.h** – En-tête privé de **upslg1_8.cpp**. • Fichier **upslg18.e** – En-tête protégé de **upslg1_8.cpp**.	
Composants.	**upslg1_99**.
Description.	
Interface entre ***Up ! Lg1*** et ***Up ! Kernel***.	
Fichiers.	
• Fichier **upslg1_99.cpp** – Fichier source. • Fichier **upslg1_99.h** – En-tête privé de **upslg1_99.cpp**. • Fichier **upslg1_99.e** – En-tête protégé de **upslg1_99.cpp**.	

Tableau 8 – Composants techniques du module

Fin de document