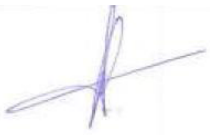
Agence ou Service : NTIC

Projet : Développement du noyau AMDA-NG (3ème partie) et intégration avec l'IHM

DOSSIER DE CONCEPTION DU NOYAU AMDA-NG (3EME PARTIE) ET DE SON INTEGRATION AVEC L'IHM AMDA

Rédigé par : Benjamin Renard Mathias Mazel	Diffusé à : CNES / IRAP 
Approuvé par : Chef de projet AKKA – N. Lormant Responsable projet CNES – N. Dufourg	

LISTE DES MODIFICATIONS DU DOCUMENT

Vers.	Date	Paragraphe	Description de la modification
01.00	01/09/14	2., 3.1.1 , 3.1.2	Création du document. Conception préliminaire pour « AMDA_Integration ». Mise à jour des conceptions détaillées des modules « Parameters » et « Plot » de « AMDA_Kernel ».
01.1	30/10/14	2., 3.1, 3.2	Mise à jour de la conception préliminaire pour « AMDA_Integration ». Ajout de la conception détaillée pour « AMDA_Integration ». Mise à jour de la conception détaillée du module « AMDA_Kernel ».
01.2	30/01/15	3.1	Mise à jour de la conception détaillée du module « AMDA_Kernel ».

SOMMAIRE

1 INTRODUCTION

Ce document présente la conception mise en œuvre dans le cadre de la troisième phase d'industrialisation du noyau AMDA-NG et de la mise en place de son intégration avec l'IHM.

Il s'appuie sur les documents de référence [R1], [R2] et [R3] qui décrivent l'architecture et la conception de la première et de la seconde phase d'industrialisation du projet.

Concernant la deuxième prestation et en particulier le module Plot, nous considérons le document [R3] **suffisant pour la compréhension des mécanismes mis en œuvre en terme de conception pour :**

- **Comprendre l'utilité et la fonction de toutes les classes misent en œuvre,**
- **Lire une requête XML d'entrée et alimenter tous les attributs des classes instanciées,**
- **Alimenter les valeurs par défaut non précisées dans la requête,**
- **La notion de contexte utilisée pour la lecture arborescente d'une requête XML,**
- **Le principe de tracé avec plplot et les notions de page, panel, axes, plot area…**

Hormis l'intégration du noyau AMDA-NG à l'IHM, cette troisième partie du projet consiste essentiellement en une évolution de l'existant pour apporter de nouvelles fonctionnalités.

La partie conception préliminaire ne sera donc pas développée pour le module « AMDA_Kernel », puisqu'elle s'appuie sur la conception existante.

2 CONCEPTION PRELIMINAIRE POUR LE MODULE « AMDA_INTEGRATION »

2.1 DESCRIPTION GENERALE

Le module « AMDA_IHM » est une application Web développée en ExtJS pour sa partie cliente, et en PHP pour sa partie serveur.

Le module « AMDA_Kernel » se présente sous forme d'un ensemble d'exécutables, développés en C++.

Le module « AMDA_Integration » se présente comme une librairie de communication entre ces deux modules indépendants.

2.2 ARCHITECTURE DE SYSTEME

Le schéma suivant représente l'architecture générale de l'application AMDA :

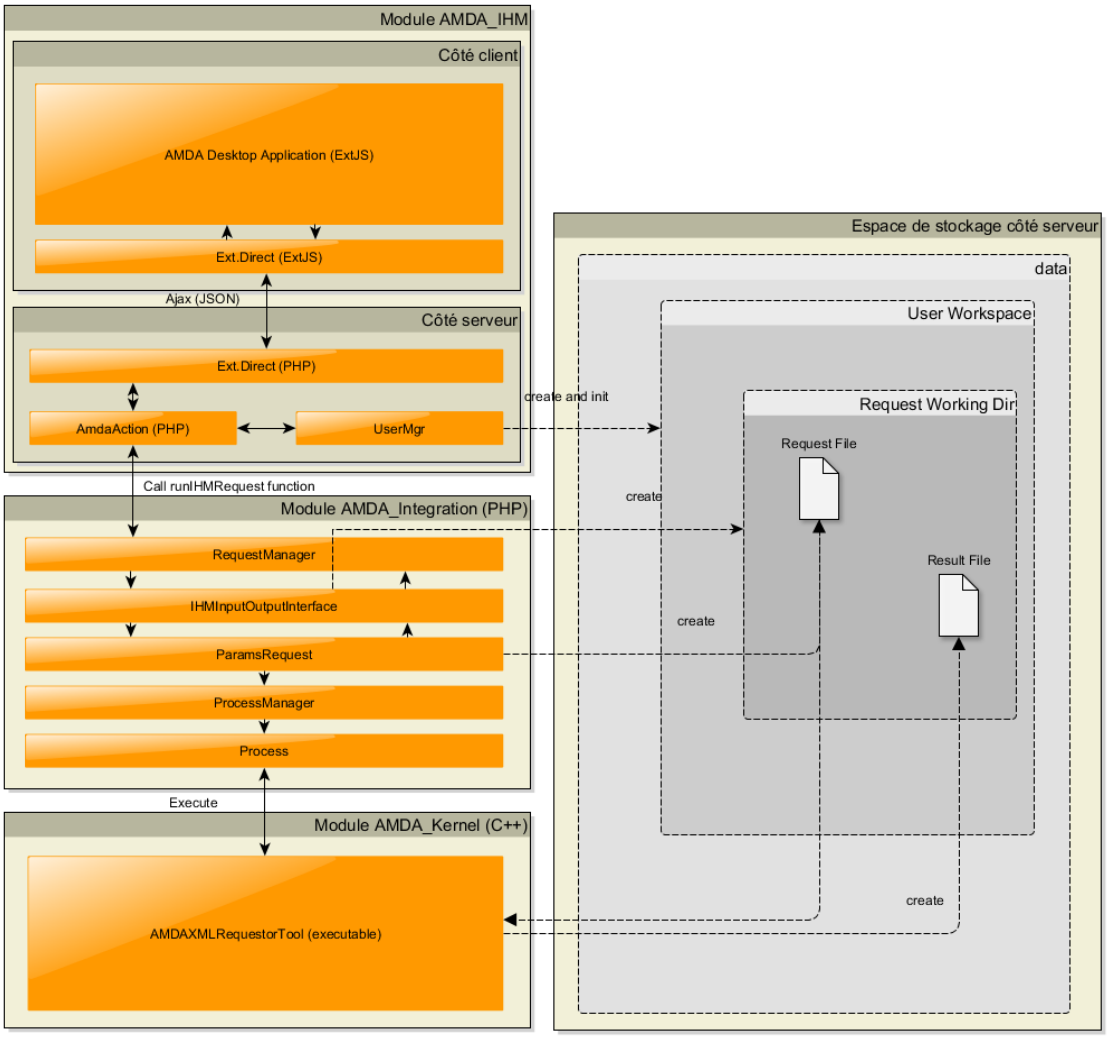


Figure 1 - Architecture générale de l'application AMDA

Le module « AMDA_Integration » est appelé par le module « AMDA_IHM » via l'appel à la fonction « runIHMRequests » de la classe « RequestManager » (cf. §2.3.1).

L'échange de données entre ces deux modules s'effectue sous forme d'objets PHP (cf. §2.3.2 et §2.3.4).

Le module « AMDA_Kernel » est appelé par le module « AMDA_Integration » via l'exécution d'une commande shell gérée par la classe « ProcessManager » (cf. §2.3.5). L'échange de données entre ces deux modules s'effectue par des fichiers écrits dans un répertoire de travail.

2.3 DESCRIPTION DE CHAQUE MODULE

2.3.1 Classe RequestManager

Cette classe constitue le point d'entrée du module « AMDA_Integration ».

Pour le moment, seul le traitement de requêtes provenant du client « AMDA_IHM » est implémenté via la fonction « runIHMRequests ».

Cette fonction attend en entrée :

- Le nom de l'utilisateur à l'origine de l'exécution de la requête,
- Un objet php contenant les paramètres d'entrée de la requête,
- La fonction à appliquer.

Elle a pour responsabilité d'instancier :

- Un objet « IHMOutputInputInterface » (implémentation de l'interface « InputOutputInterface » cf. §2.3.2),
- Un objet d'une implémentation de la classe « RequestAbstract », en fonction du type de requête à exécuter.

Elle effectue ensuite la séquence de traitement d'une requête telle que décrite dans §2.3.3.

Cette fonction retourne un objet PHP contenant les paramètres de sortie de la requête.

2.3.2 Interface InputOutputInterface

Il s'agit d'une interface dont les implémentations ont pour responsabilités :

- de fournir un objet de type « RequestDataClass » (cf. §2.3.4) en fonction du type de la requête à partir des paramètres d'entrée de la requête du client (fonction « getInputData »),
- de fournir le résultat de la requête sous forme d'un objet PHP compréhensible par le client (fonction « getOutput »),

Dans le cas d'une requête traitée par la fonction « runIHMRequests », c'est l'implémentation « IHMInputOutputClass » qui est utilisée.

2.3.3 Classe RequestAbstract

Il s'agit d'une classe abstraite dont les implémentations ont pour responsabilité de fournir les différentes fonctions de la séquence d'exécution d'une requête :

- « setData » : permet de fournir l'objet « RequestDataClass » contenant les paramètres d'entrée de la requête,
- « init » : initialise la requête en contrôlant notamment les paramètres d'entrée,
- « run » : exécute la requête,
- « getData » : fournit l'objet « RequestDataClass » contenant le résultat de l'exécution de la requête.

Une implémentation de cette classe est proposée pour chaque type de requête.

2.3.4 Classe RequestDataClass

Cette classe constitue la classe de base de la structure d'accès unifié aux données d'une requête.

Elle peut être dérivée en fonction des besoins spécifiques d'une requête.

2.3.5 Classe ProcessManager

Cette classe a pour responsabilités de gérer l'exécution des processus lancés par le module (lancement d'un processus, arrêt d'un processus, récupération du statut).

3 CONCEPTION DETAILLEE

3.1 DESCRIPTION DETAILLEE DU MODULE AMDA_INTEGRATION

3.1.1 Implémentation « InputOutputInterface » pour l'IHM d'AMDA

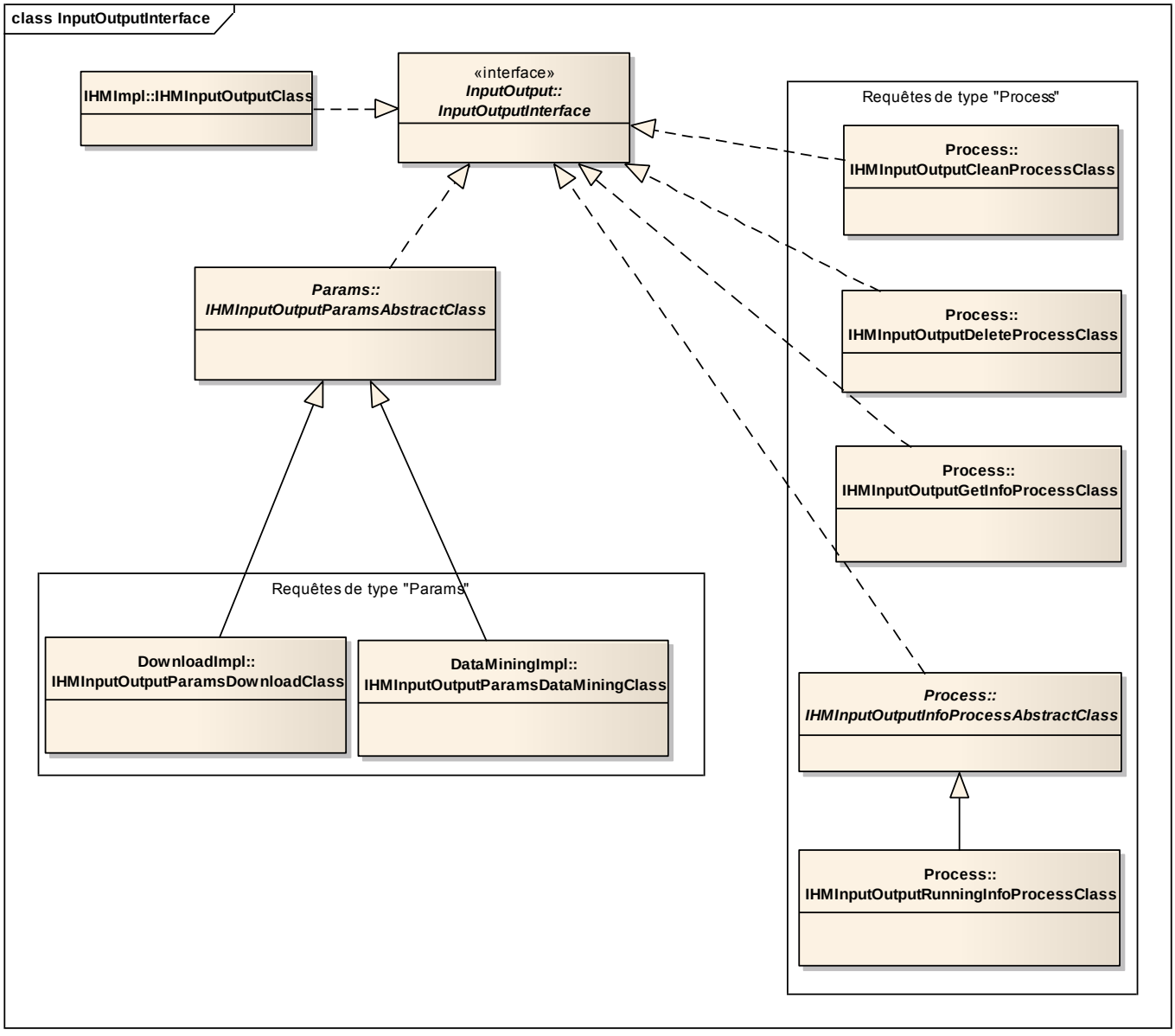


Figure 2 - Diagramme de classe de l'implémentation InputOutputInterface pour l'IHM

La classe « IHMInputOutputClass » implémente l'interface « InputOutputInterface ». La responsabilité de cette implémentation est de gérer les entrées / sorties du client « AMDA_IHM » avec le module « AMDA_Integration ».

Cette classe se présente comme un routeur redirigeant les demandes vers d'autres implémentations de l'interface « InputOutputInterface » en fonction du type de requête.

Il existe deux types de requêtes :

- Les requêtes de type « Params », c'est-à-dire des requêtes travaillant sur des données de paramètres (requêtes de type « download », « data mining » et « plot » au niveau de l'IHM).
- Les requêtes de type « Process », c'est-à-dire des requêtes travaillant sur des processus (requêtes de type « jobs » au niveau de l'IHM).

3.1.2 Implémentations de RequestAbstractClass et structures de données

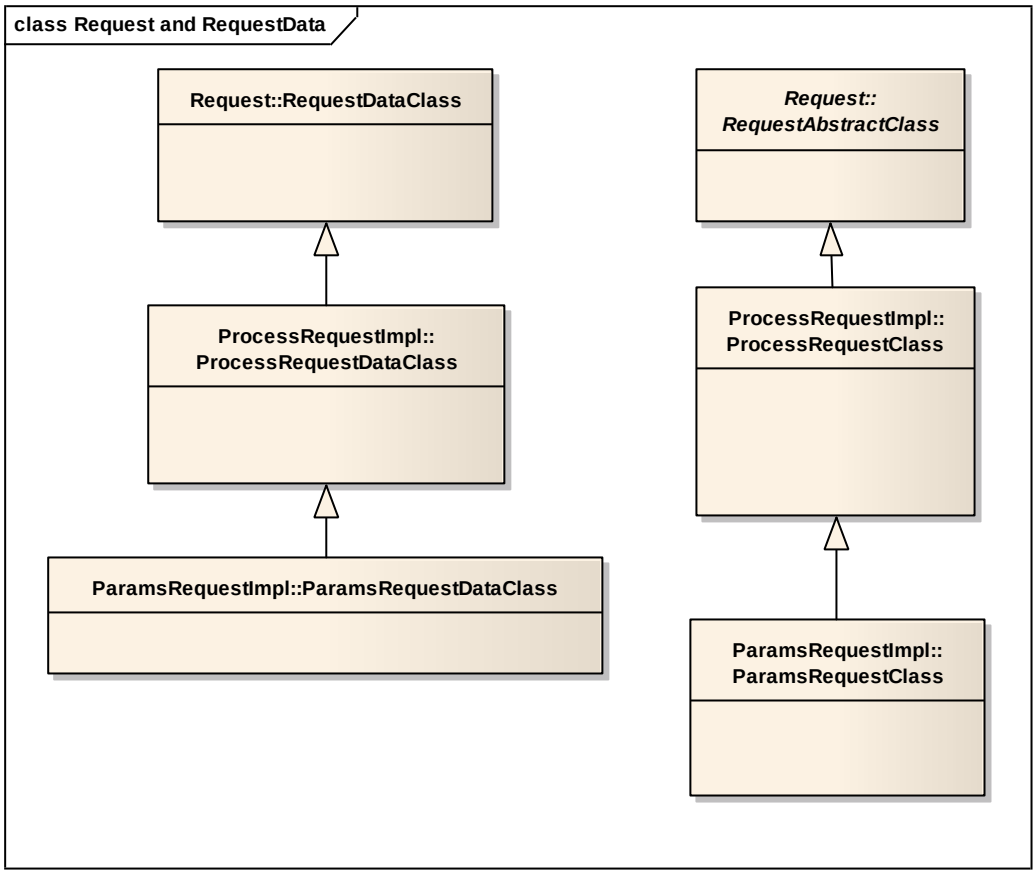


Figure 3 - Implémentations de RequestAbstractClass et structures de données

L'ensemble des requêtes sont traitées à partir de deux implémentations de RequestAbstractClass :

- L'implémentation « ProcessRequestClass » gérant l'ensemble des requêtes s'appliquant sur les processus exécutés par le module, en utilisant une structure de données de type « ProcessRequestDataClass »,
- L'implémentation « ParamsRequestClass », héritant de la classe « ProcessRequestClass », dont la responsabilité est d'exécuter une requête du module AMDA_Kernel à partir de l'exécutable « amdaXMLRequestorTool », en utilisant une structure de données de type « ParamsRequestDataClass » qui hérite de « ProcessRequestDataClass ».

3.1.3 Diagramme de séquences du traitement d'une requête de type download de l'IHM AMDA

Le diagramme suivant représente la séquence générale d'une requête de type download, provenant de l'IHM AMDA.

Par soucis de visibilité, il ne présente pas les séquences effectuées au niveau de la classe « IHMInputOutputClass » et au niveau de la classe « ParamsRequestClass ».

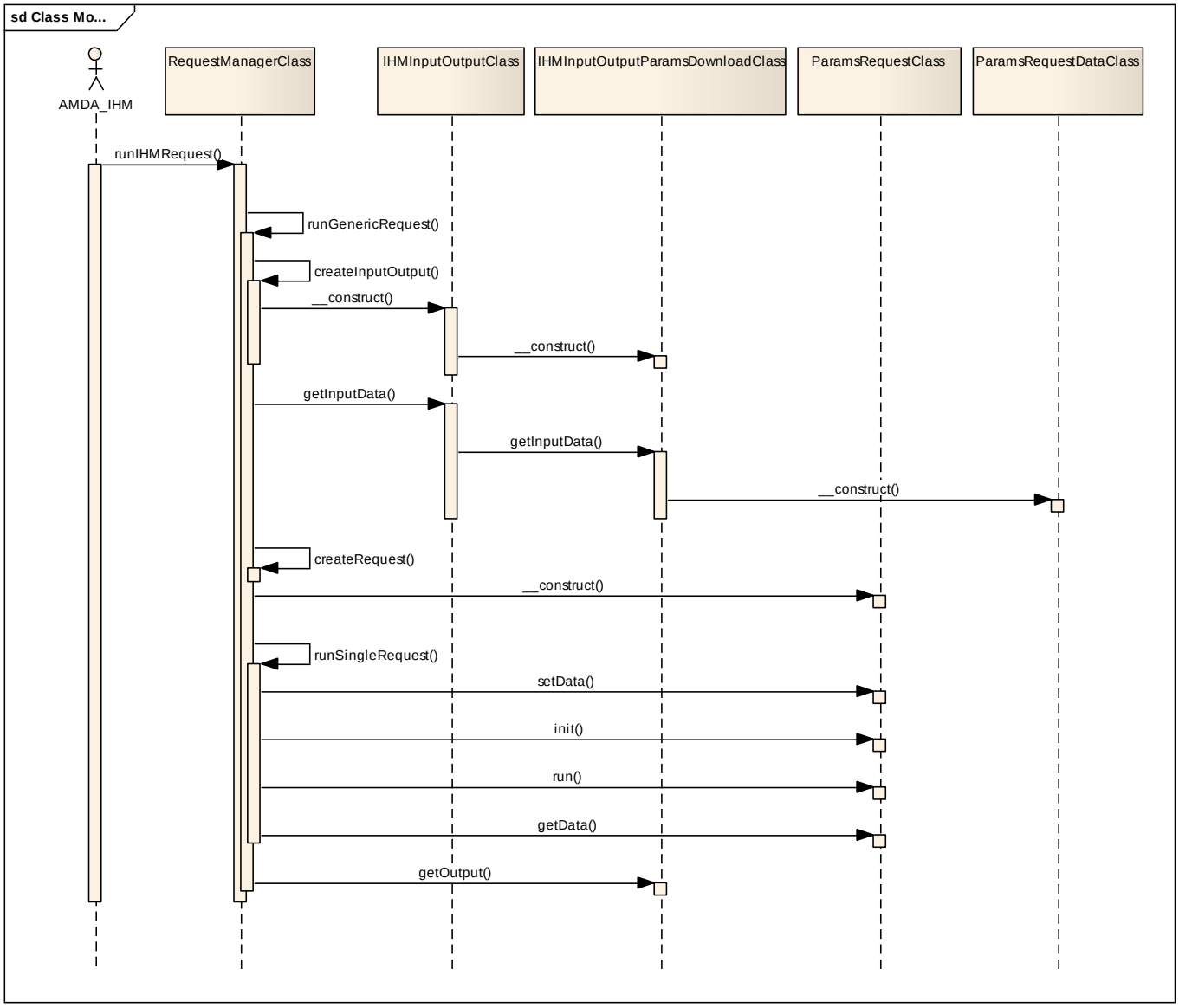


Figure 4 - Diagramme de séquence général pour une requête de type download

3.1.4 Gestion des processus exécutés

La classe « ProcessRequestClass » a pour responsabilité de traiter les requêtes s'appliquant sur les processus exécutés par le module.

Pour se faire, elle utilise la classe « ProcessManagerClass ».

3.1.4.1 Fichier de gestion des processus

La classe « ProcessManagerClass » enregistre tous les processus exécutés, ainsi que leurs statuts, dans un fichier XML.

L'accès concurrentiel à ce fichier est garanti par le système.

3.1.4.2 Exécution d'un processus

La fonction « runProcess » de la classe « ProcessManagerClass » permet l'exécution d'un nouveau processus, sous forme d'une commande shell.

Pour se faire, une instance de la classe « ProcessClass » est créée, puis exécuté.

Deux cas de figure se présentent :

- Si le paramètre « batchEnabled » de la fonction est à « false », l'application se mettra en attente jusqu'à la fin de l'exécution du processus,
- Si le paramètre « batchEnabled » de la fonction est à « true », l'application laissera le processus s'exécuter en mode asynchrone si le temps d'exécution dépasse le temps configuré au niveau de la classe « KernelConfigClass ».

Un processus exécuté est désigné par la suite par un « id » et enregistré dans le fichier de gestion des processus (cf. §3.1.4.1).

3.1.4.3 Récupération des informations d'un processus

La fonction « getProcessInfo » de la classe « ProcessManagerClass » permet de récupérer les informations concernant un processus désigné par son « id ».

Pour se faire, une instance de la classe « ProcessClass » est créée et initialisée à partir des informations sur le processus contenues par le fichier de gestion des processus (cf. §3.1.4.1).

Cette instance est ensuite sollicitée pour mettre à jour le statut du processus dans le fichier de gestion des processus.

Enfin, le statut du processus est retourné par la fonction.

3.1.4.4 Suppression d'un processus

La fonction « deleteProcess » de la classe « ProcessManagerClass » permet de supprimer un processus désigné par son « id ».

Pour se faire, une instance de la classe « ProcessClass » est créée et initialisée à partir des informations sur le processus contenues par le fichier de gestion des processus (cf. §3.1.4.1).

Cette instance est ensuite sollicitée pour stopper l'exécution du processus.

Enfin, le processus est supprimé du fichier de gestion des processus.

3.2 DESCRIPTION DETAILLEE DU MODULE AMDA_KERNEL

3.2.1 Module Parameters

3.2.1.1 Ré-échantillonnage des paramètres

Les paramètres d'une requête peuvent être amenés à être ré-échantillonnés en interne en fonction des besoins lors de leurs traitements par les différentes implémentations de « ParamOutput ».

Ces fonctionnalités sont accessibles via les fonctions du « ParameterManager » :

- « getSampledParameter » : ré-échantillonnage d'un paramètre à partir d'un temps d'échantillonnage,
- « getSampledParameterUnderRefParam » : ré-échantillonnage d'un paramètre à partir d'un paramètre de référence. Le paramètre de référence définie la liste des temps sur lesquels le ré-échantillonnage doit être effectué.

Les appels à ces fonctions construisent le nouveau paramètre correspondant à l'échantillonnage demandé, s'il n'existe pas déjà, et retourne un pointeur vers le paramètre correspondant. Les implémentations de « DataWriter » agrégés par ces paramètres sont des « Process » :

- « sampling_classic » : pour le ré-échantillonnage d'un paramètre à partir d'un temps d'échantillonnage,
- « sampling_under_refparam » : pour le ré-échantillonnage d'un paramètre à partir d'un paramètre de référence.

3.2.1.2 Mode de ré-échantillonnage « valeurs les plus proches »

Par défaut, un paramètre est ré-échantillonné en moyennant ou en interpolant les données du paramètre de référence.

Cependant, dans le cas de paramètres désignant une orbite ou un état, il peut être nécessaire d'effectuer un ré-échantillonnage en se contentant de conserver les valeurs du paramètre de référence les plus proches des temps ciblés. Toutes les implémentations d'un « DataWriter » doivent donc définir une méthode « useNearestValue » dont la responsabilité est de retourner un booléen en fonction de l'activation ou non de ce mode de ré-échantillonnage.

3.2.1.3 Définition du critère « gap physic »

La définition du critère « gaps physic » désignant l'intervalle de temps à partir duquel une absence de données pour un paramètre est à considérer comme un trou de données est définie à partir :

- Du « gapthreshold » représentant le nombre d'échantillonnage à considérer dans le calcul du critère,
- Du « minSampling » représentant le temps d'échantillonnage minimal du paramètre.

Pour le moment, le critère est calculé de la manière suivante :

$$gap_{criteria} = gapthreshold * minSampling$$

A l'avenir, ce critère devra prendre en compte le cas où le sampling du paramètre est variable.

Ce calcul est effectué par la méthode « getComputedGapSize » du « ParameterManager » et est utilisé par :

- Les Process de resampling « ProcessSamplingClassic » et « ProcessSamplingUnderRefParam » afin de ne pas interpoler des données lorsqu'elles se situent sur un trou de données,
- L'implémentation « PlotOutput » du « ParamOutput » afin de ne pas tracer les trous de données sur les plots tracés.

Une valeur par défaut à utiliser pour le « gapthreshold » est définie dans le fichier de configuration de l'application « app.properties » avec la clé : « app.param.gapthreshold ». Si cette clé n'existe pas, c'est la valeur par défaut « 5 », codée en dur au niveau du « ParameterManager » qui est utilisée.

Enfin, la valeur réelle du « gapthreshold » pour un paramètre donné peut être définie dans le fichier de définition du paramètre (nœud « gap_threshold »).

3.2.1.4 Process « statistique »

La classe « StatisticProcess » est une classe abstraite qui a la particularité d'implémenter « DataClient ». En effet, une opération « statistique » s'applique sur les données d'un paramètre de référence.

Elle contient une « StatisticOperation » dont une implémentation sera chargée d'effectuer le calcul statistique.

Le résultat du calcul statistique sera stocké au niveau d'une instance « TStatisticData » portée par la classe « StatisticOperation ».

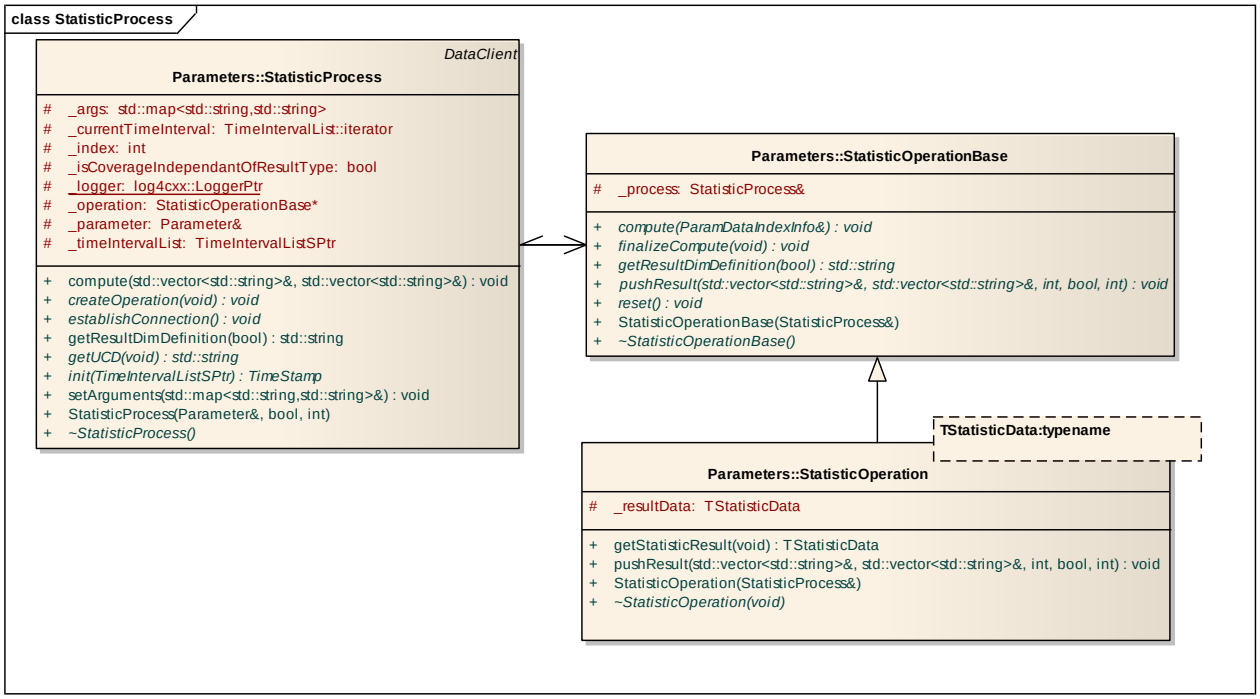


Figure 5 - Description d'un StatisticProcess

Différentes implémentations de ce type de process sont disponibles dans le plugin « StatisticProcesses » :

- « MinMaxMeanStatisticProcess » : permettant de réaliser des calculs de « min », « max » et « mean »,
- «MinVarStatisticProcess » : permettant de réaliser de « minvar ».

Les fabriques de ces process peuvent être atteintes via le singleton « ServicesServer » (cf. [R2]).

3.2.1.5 Type de données « Tab2D »

Dans le cadre de cette prestation, un nouveau type de données « Tab2D » a été implémenté.

Il consiste en la définition d'un conteneur de deux dimensions définit par la classe « Tab2DData ».

Ce conteneur a été ensuite utilisé afin de définir les « ParamData » correspondants.

Par la suite, la procédure décrite dans l'Annexe 3 de [R2] a été suivie.

3.2.2 Module Info

Le module « Info » est un module gérant **l'accès et le stockage de l'ensemble des informations sur les paramètres** portés par le « ParameterManager » du module « Parameter ».

Ces informations sont accessibles par l'ensemble des modules du noyau.

3.2.2.1 Structures de stockages des informations

Ce module se décompose notamment en quatre types de structure de stockage des informations :

- La structure « ParamInfo » contenant les informations directes d'un paramètre. Cette structure contient notamment un champ « dataset_id » faisant référence à la structure d'information sur le dataset,
- La structure « DataSetInfo » contenant les informations d'un dataset. Cette structure contient notamment un champ « instrument_id » faisant référence à la structure d'information sur l'instrument,
- La structure « InstrumentInfo » contenant les informations d'un instrument. Cette structure contient notamment un champ « mission_id » faisant référence à la structure d'information sur la mission,
- La structure « MissionInfo » contenant les informations d'une mission.

3.2.2.2 Gestion et accès aux différentes informations

A chaque type de structure d'information correspond un manager gérant l'accès à ces informations (« ParamMgr », « DataSetMgr », « InstrumentMgr » et « MissionMgr »).

Un tel manager est un singleton contenant une méthode « get***InfoFromId » (avec *** correspondant au type d'information géré par le manager) permettant de récupérer un « shared_ptr » sur la structure d'information demandé.

Ainsi, à partir d'un identifiant d'un paramètre, un module peut :

- Appeler la fonction « getParamInfoFromFile » du singleton « ParamMgr » afin d'accéder aux informations directes sur le paramètre,

- A partir du champ « dataset_id » de la structure d'information « ParamInfo », appeler la fonction « getDataSetInfoFromId » du singleton « DataSetMgr » afin d'accéder aux informations sur le dataset du paramètre,
- A partir du champ « instrument_id » de la structure d'information « DataSetInfo », appeler la fonction « getInstrumentInfoFromId » du singleton « InstrumentMgr » afin d'accéder aux informations sur l'instrument du dataset correspondant au paramètre étudié,
- A partir du champ « mission_id » de la structure d'information « InstrumentInfo », appeler la fonction « getMissionInfoFromId » du singleton « MissionMgr » afin d'accéder aux informations sur la mission de l'instrument correspondant au dataset correspondant lui-même au paramètre étudié.

3.2.2.3 Lecture des informations

Les différentes informations d'un paramètre sont contenues dans des fichiers XML.

A chaque type de structure d'information correspond un parseur gérant la lecture de ces fichiers (« ParamParser », « DataSetParser », « InstrumentParser » et « MissionParser »).

Un tel parseur hérite d'un « XMLConfigurator », et la lecture du fichier XML correspondant se fait via l'appel à la fonction « parse ».

Cette fonction retourne un « shared_ptr » vers une instance renseignée de la structure d'information correspondante.

3.2.2.4 Mise à jour des informations par le « ParamGet »

Certaines informations d'un paramètre peuvent être « surchargées » par l'implémentation du « ParamGet » qui lui est associée.

Pour se faire, la méthode « updateInfo » du « DataWriter » peut être surchargée au niveau d'une implémentation d'un « ParamGet » (pour rappel, un « ParamGet » implémente un « DataWriter » cf. [R2]).

A charge ensuite à cette méthode surchargée d'accéder aux structures d'information correspondantes au paramètre et de les mettre à jour en conséquence.

Il est à noter que cette méthode est automatiquement appelée par la méthode « init » du « DataWriter ».

3.2.2.5 Présentation des différentes classes du module info

Les 4 diagrammes de classes suivants présentent les singletons cités précédemment et les classes qui leur sont rattachées.

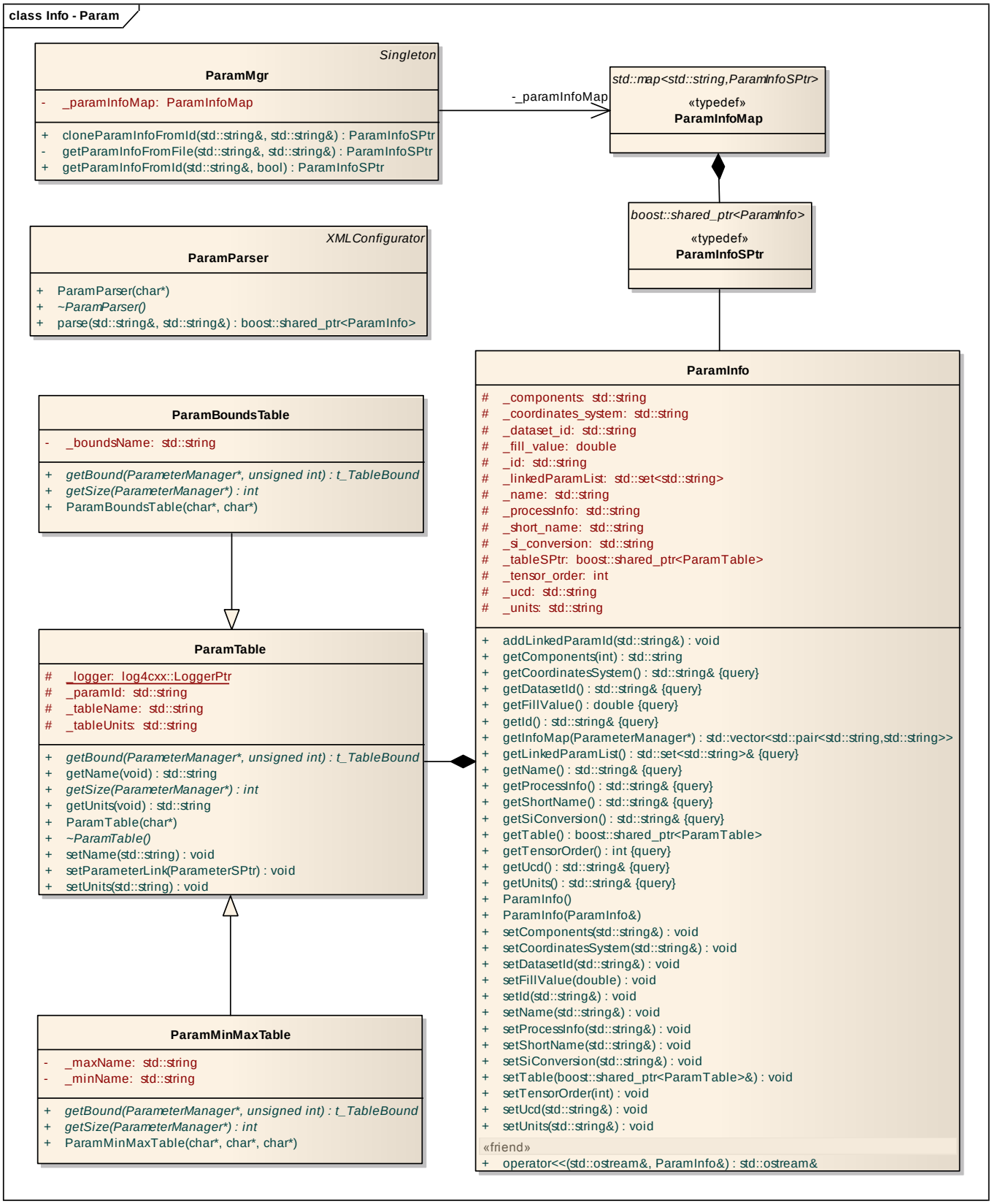


Figure 6 - Le singleton ParamMgr et les classes associées

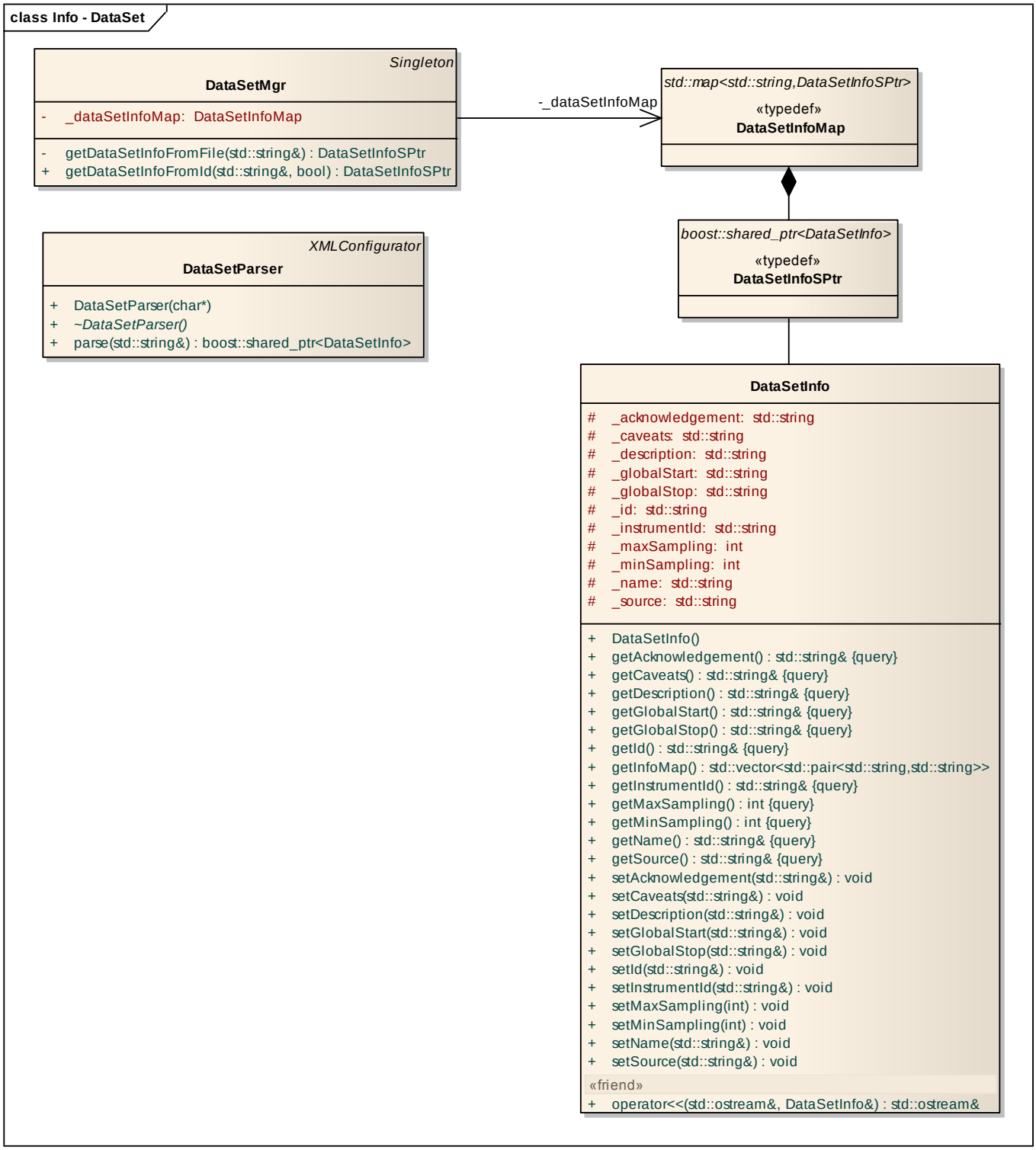


Figure 7 - Le singleton DataSetMgr et les classes associées

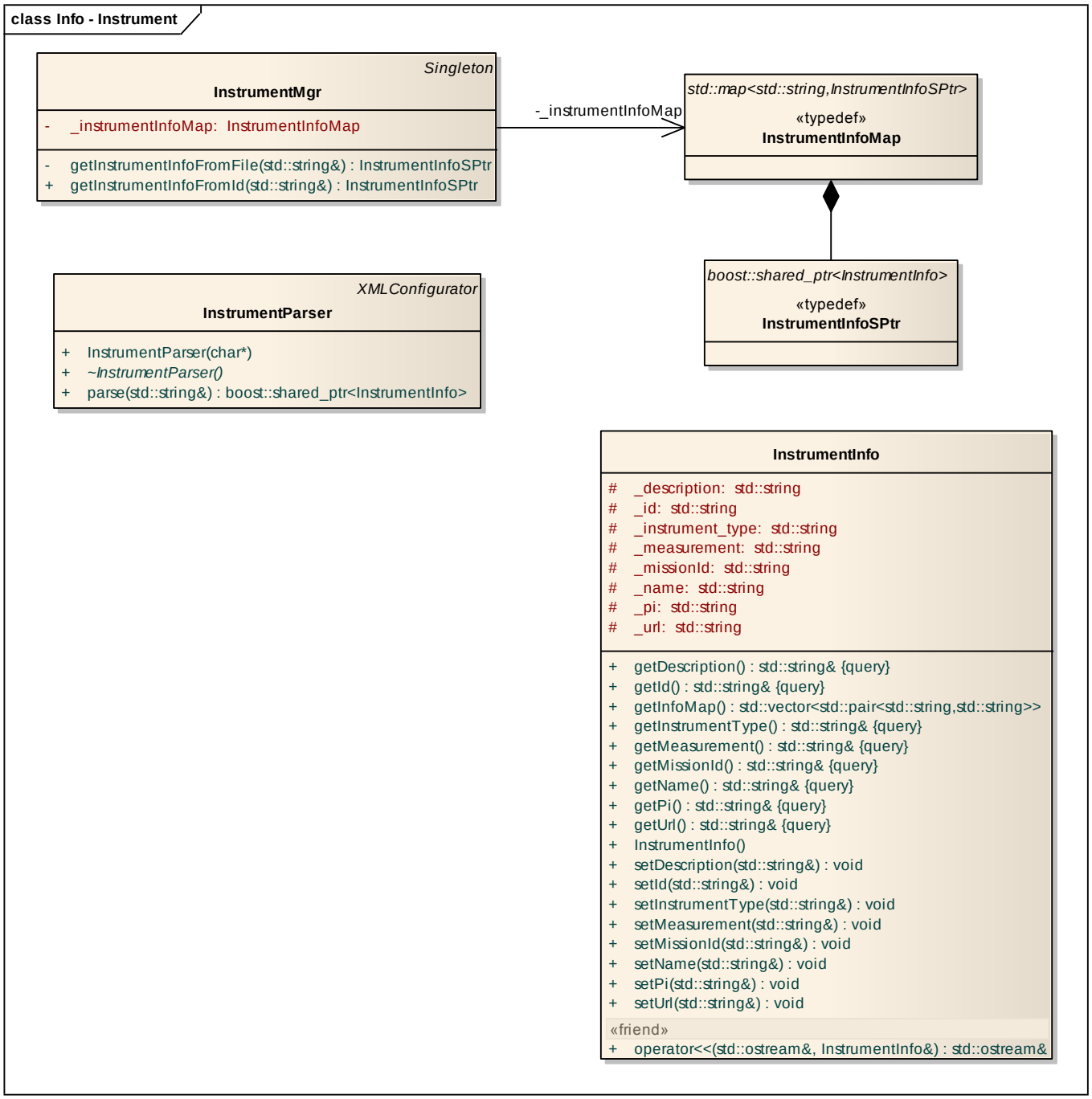


Figure 8 - Le singleton InstrumentMgr et les classes associées

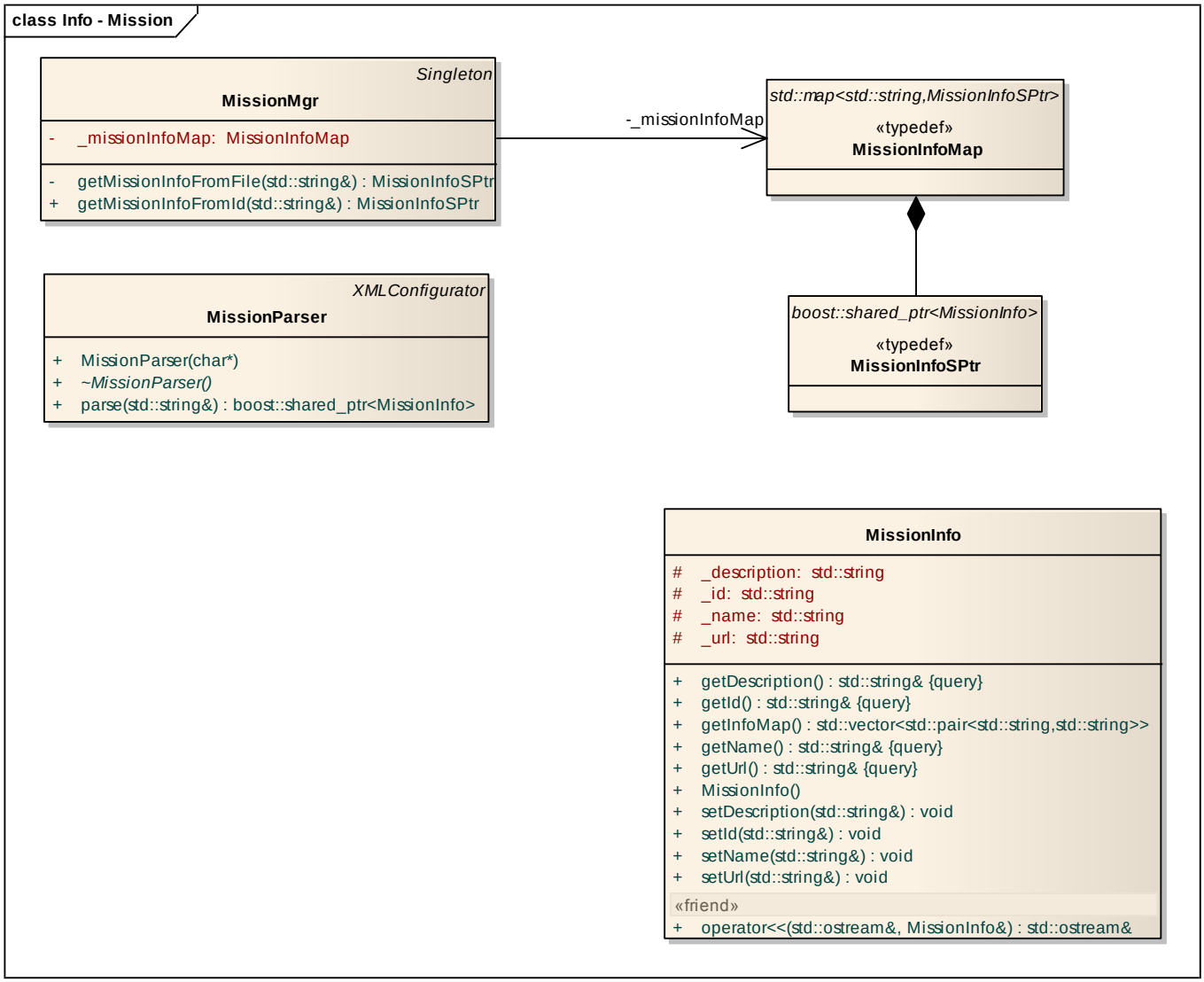


Figure 9 - Le singleton MissionMgr et les classes associées

3.2.3 Module Download

Le module « Download » est une implémentation d'un « ParamOutput » dont la responsabilité est de fournir les données d'une requête dans un fichier de sortie.

3.2.3.1 Mécanisme général du traitement d'un output de type « download »

Un output de type « download » est traité par la classe « DownloadOutput » qui est une implémentation d'un « ParamOutput » (cf. [R2]).

Lors de cette prestation, un rework important du module Download a été effectué, rendant obsolète la conception décrite dans [R3]. Dorénavant :

- Le **« DownloadOutput »** porte le mécanisme général du traitement d'un tel output, **mais reste indépendant du type de format de fichier demandé**,
- La **classe abstraite « FileWriter »** a pour responsabilité de **porter les méthodes d'écriture dans un fichier**. Une implémentation de cette classe doit être écrite pour chaque format de fichier.

La nouvelle séquence mise en œuvre pour le traitement d'un output de type « download » est décrite dans le diagramme de séquence suivant :

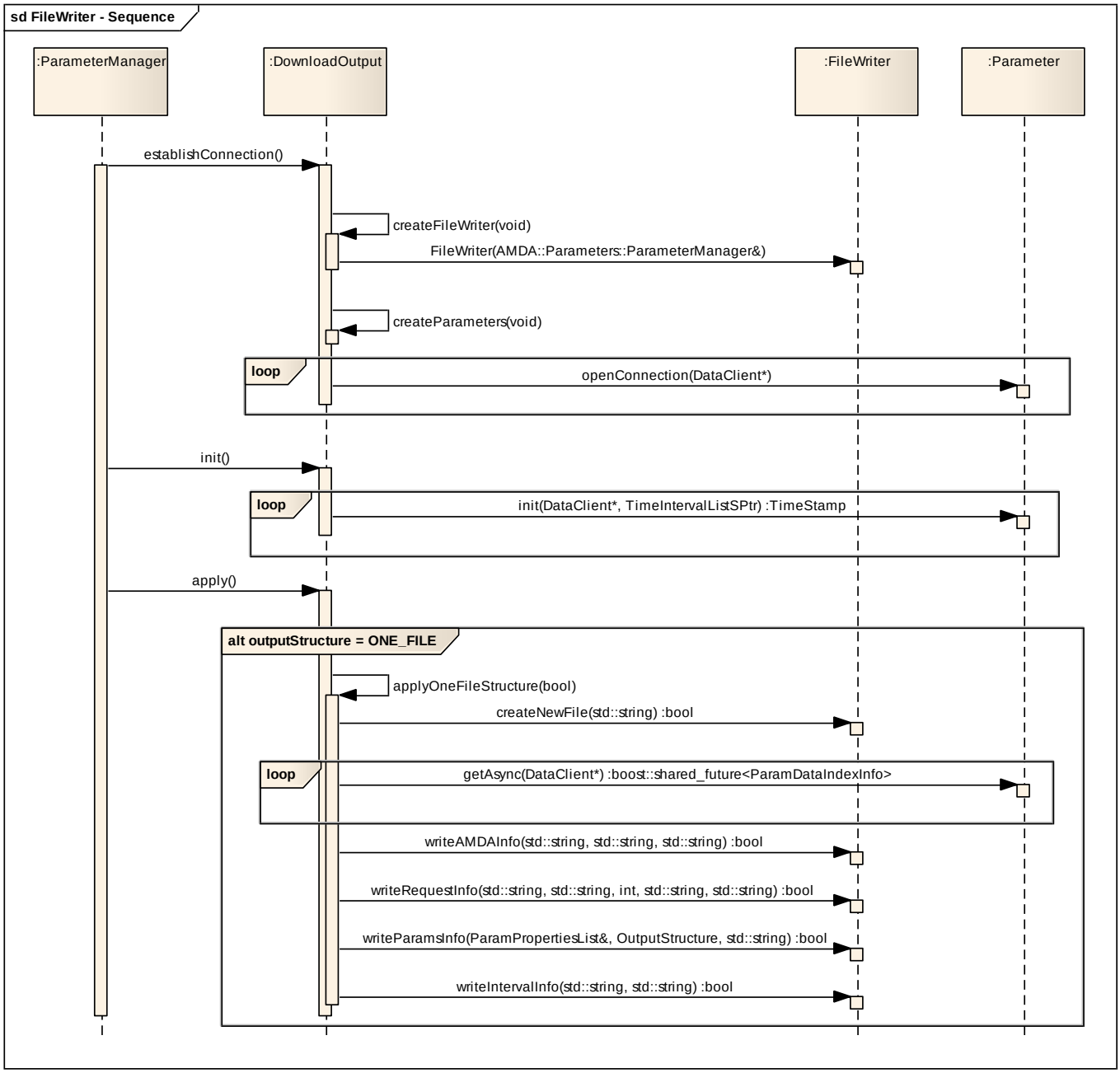


Figure 10 - Séquence générale pour le traitement d'un output de type "Download"

Cette séquence se décompose, à l'instar de toutes les implémentations d'un « ParamOutput », en trois étapes principales commandées par le « ParameterManager » (cf. [R2]) :

- L'appel à la fonction « establishConnection ». Dans le cadre du module Download, cette fonction a trois finalités :
  - Instancier une implémentation d'un « FileWriter » en fonction du format de fichier requis,
  - Constituer une liste des paramètres resamplés nécessaires au traitement de la requête, en appelant la fonction « createParameters »,
  - Etablir la connexion de chacun des paramètres requis, en appelant la fonction « openConnection » de chaque « Parameter ».
- L'appel à la fonction « init ». Cette fonction ce contente d'effectuer un appel séquentiel à la méthode « init » de chaque paramètre requis,
- L'appel à la fonction « apply » qui, et pour chaque intervalle de temps demandé en requête, appelle la fonction portant la séquence correspondante au « outputStructure » demandé en requête :
  - Dans le cas « one-file » ou « one-file-refparam », il s'agit de la fonction « applyOneFileStructure » :
    - Création du fichier s'il s'agit du traitement du premier intervalle,
    - Récupération des données de l'intervalle courant pour chaque paramètre à écrire dans le fichier,
    - Ecriture des informations sur les paramètres de la requête s'il s'agit du traitement du premier intervalle.
  - Dans le cas « one-file-per-interval » ou « one-file-per-interval-refparam », il s'agit de la fonction « applyOneFilePerInterval » :
    - Création d'un nouveau fichier courrant,
    - Récupération des données de l'intervalle courant pour chaque paramètre à écrire dans le fichier,
    - Ecriture des informations sur les paramètres de la requête. Lorsque le type de format de fichier impose une écriture des informations dans un fichier séparé, cette étape n'est réalisée que lors du traitement du premier intervalle de temps.

3.2.3.2 Ecriture des données dans un fichier

Le « DownloadOutput » implémente le « VisitorOfParamData » pour différencier les types de données et réaliser la sortie adéquate.

Cette sortie est écrite dans un fichier à partir de l'instance du « FileWriter » correspondante au format de fichier demandé.

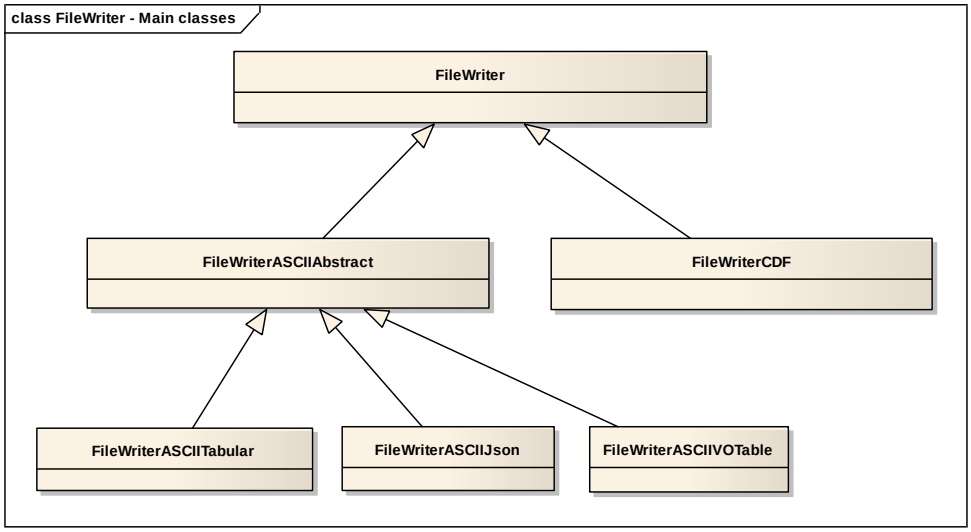


Figure 11 - Implémentations du FileWriter

Les implémentations disponibles du « FileWriter » sont :

- « FileWriterCDF » : implémentation permettant d'écrire un fichier au format CDF,
- « FileWriterASCIIAbstract » : classe abstraite servant de base commune à toutes les implémentations d'écriture d'un fichier ASCII :
  - « FileWriterASCIITabular » : implémentation permettant d'écrire un fichier au format ASCII Tabulaire (données écrites sous forme de colonnes séparées par une tabulation),
  - « FileWriterASCIIJson » : implémentation permettant d'écrire un fichier au format JSON,
  - « FileWriterASCIIVOTable » : implémentation permettant d'écrire un fichier au format VOTable 1.3.

3.2.3.3 Traitement des « petits intervalles de temps »

Les « petits intervalles de temps » correspondent à des intervalles de temps dont la taille est inférieure à la valeur du ré-échantillonnage demandé.

Ces intervalles sont exclus de la séquence de traitement normal (cf. §3.2.3.1) et sont conservés pour un post-traitement effectué par l'implémentation de la méthode « terminate » du « DownloadOutput ».

Dans cette fonction, chaque paramètre de la requête créera son propre « process statistique » portant une opération de type « mean » (cf. §3.2.1.4) afin de produire ensuite un catalogue (cf. §3.2.6) contenant, pour chaque « petits intervalles », une liste des paramètres moyennés.

3.2.4 Module Statistic

Le module « Statistic » est une implémentation d'un « ParamOutput » dont la responsabilité est de fournir les résultats de calculs statistiques d'une requête dans un fichier catalogue de sortie.

3.2.4.1 Mécanisme général du traitement d'un output de type « statistic »

Un output de type « statistic » est traité par la classe « StatisticOutput » qui est une implémentation d'un « ParamOutput » (cf. [R2]).

La séquence mise en œuvre pour le traitement d'un output de type « statistic » est décrite dans le diagramme de séquence suivant :

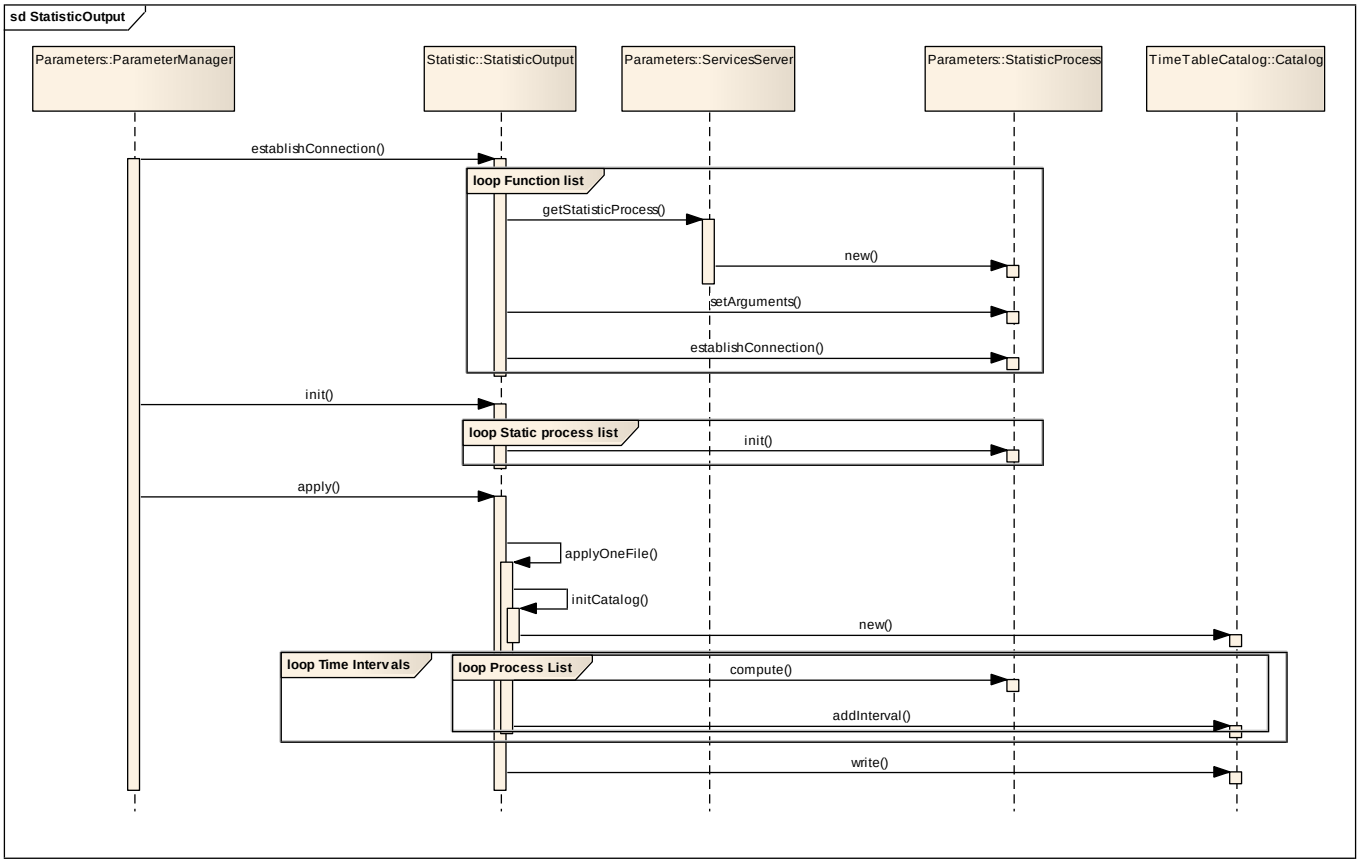


Figure 12 - Diagramme de séquence pour un StatisticOutput

Cette séquence se décompose, à l'instar de toutes les implémentations d'un « ParamOutput », en trois étapes principales commandées par le « ParameterManager » (cf. [R2]) :

- L'appel à la fonction « establishConnection ». Dans le cadre du module Statistic, cette fonction :
  - Récupère au niveau du « ServicesServer » une instance d'un « StatisticProcess » pour chaque opération demandée par la requête,
  - Etablie la connexion de chacun des process en appelant la fonction « establishConnection .
- L'appel à la fonction « init ». Cette fonction ce contente d'effectuer un appel séquentiel à la méthode « init » de chaque instance de « StatisticProcess » créée à l'étape précédente,

- L'appel à la fonction « apply » qui, et pour chaque intervalle de temps demandé en requête, appelle la fonction portant la séquence correspondante au « outputStructure » demandé en requête :
  - Dans le cas « one-file », il s'agit de la fonction « applyOneFile »,
  - Dans le cas « one-file-per-parameter » , il s'agit de la fonction « applyOneFilePerParameter ».

  Ces séquences créaient les fichiers catalogues nécessaires, calculent les résultats statistiques pour chaque intervalle de temps (à partir de la fonction « compute » des instances des « ProcessStatistic »), et insèrent ces résultats dans les fichiers catalogue.

  Une dernière consiste ensuite à écrire les fichiers catalogue résultats.

3.2.5 Module Plot

3.2.5.1 Présentation des notions de page, panel et plotArea

Dans sa version actuelle, AMDA-NG est capable de produire une ou plusieurs pages au sein d'un même fichier (pour les formats de sortie PDF et PS) ou plusieurs pages séparées (formats de sortie PNG et SVG) à partir d'un intervalle de temps ou d'une TimeTable.

Une page peut être constituée de un ou plusieurs panel dont les coordonnées sont données dans un intervalle (0..1, 0..1) sans tenir compte des marges de la page (x margin et y margin).

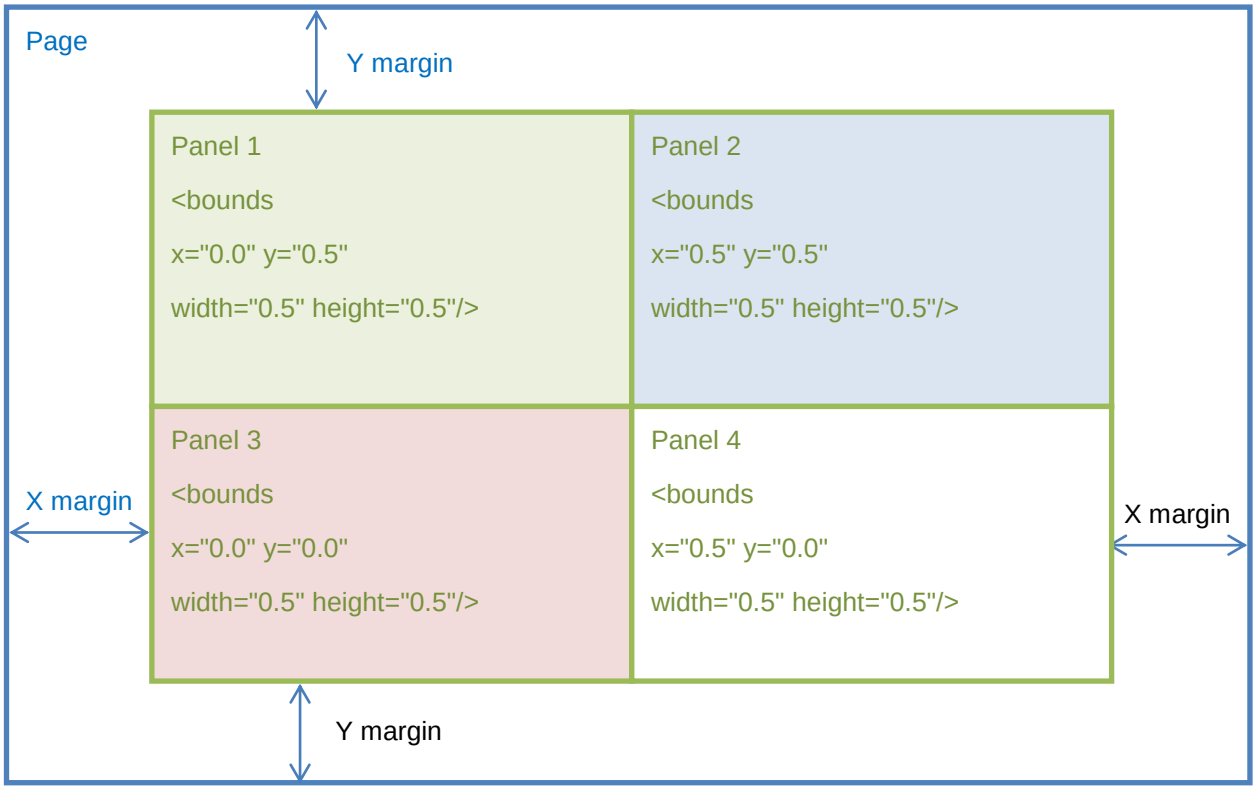


Figure 13 - Notions de page et panels

A l'intérieur d'un panel, une zone baptisée « plot area » est déterminée, celle-ci définit la zone dans laquelle les séries ou spectrogrammes sont tracés. La taille de la plot area correspond à la taille du panel à laquelle on soustrait les dimensions :

- du titre du panel,
- des axes en x et y (graduations + Textes de graduations + Légende)
- du colorAxis dans le cas d'un spectrogramme,
- des éventuelles légendes associées aux paramètres ou aux axes,
- de l'éventuel « DecoratorPlot » associé à ce panel (cf. §3.2.5.13).

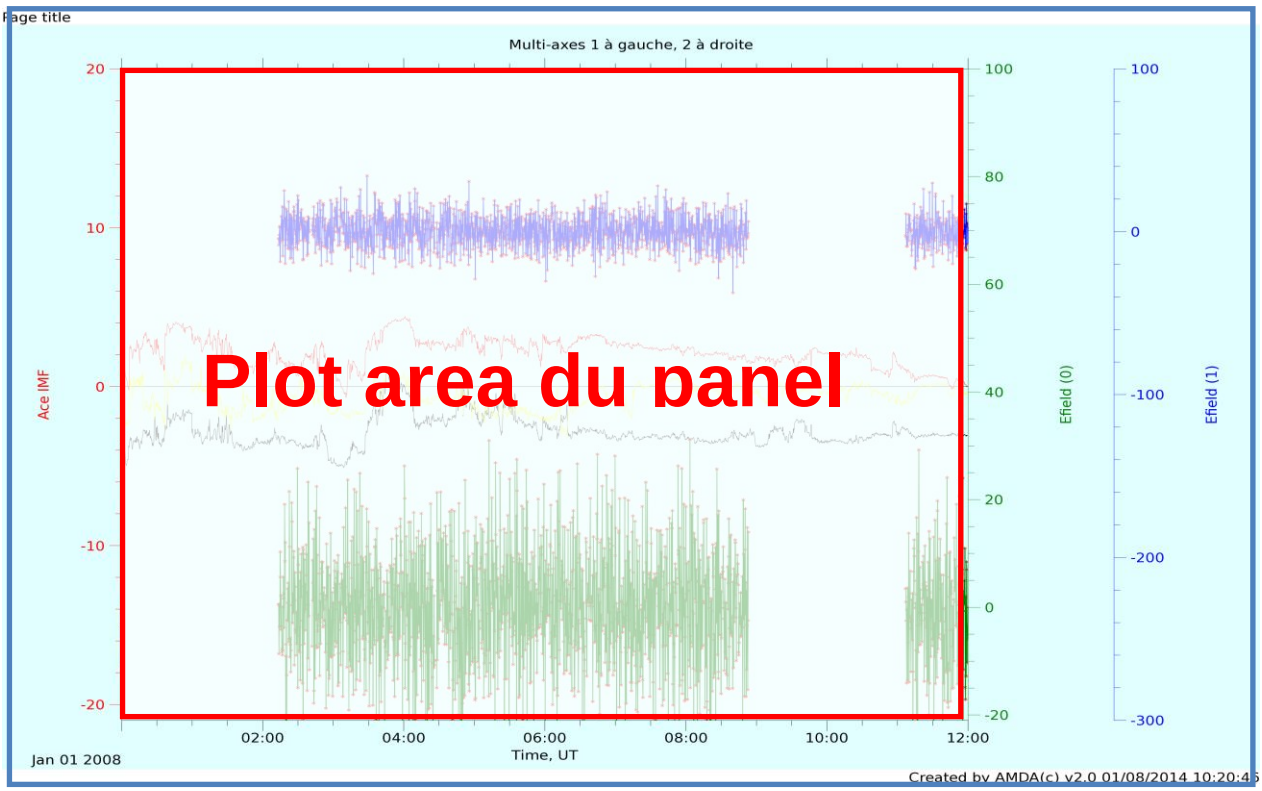


Figure 14 – Plot area pour un panel

Les classes ***Page, Panel*** *et* ***PanelPlotOutput*** *du module plot gèrent (entre autre !) les principales notions définies ci-dessus. La méthode calculatePlotArea implémentée dans la classe* ***PanelPlotOutput*** et ses classes dérivées détermine les dimensions de la plot area du panel.

L'espacement attribué à chaque axe vertical est déterminé en fonction de la présence et de la taille des éléments qui le compose (tick,, tickmark et legend) :

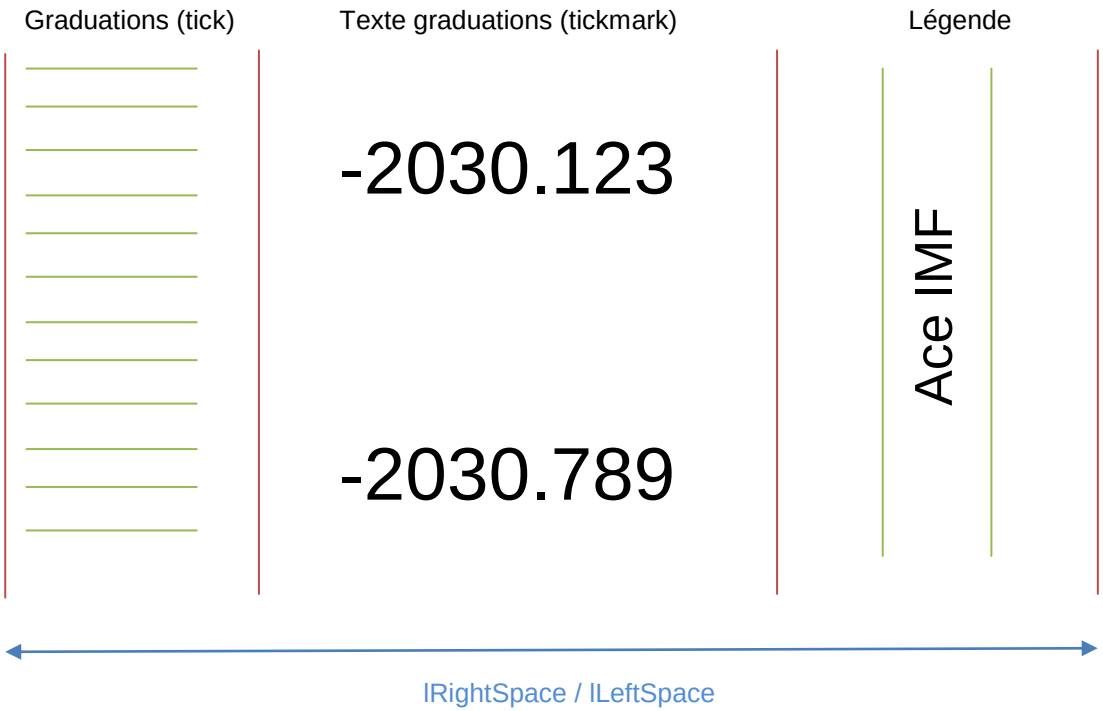


Figure 15 - Principe de calcul des espacements entre axes verticaux

3.2.5.2 Mécanisme général du traitement d'un output de type « plot »

Un output de type « plot » est traité par la classe « PlotOutput » qui est une implémentation d'un « ParamOutput » (cf. [R2]).

Lors de cette prestation, un rework important du module Plot a été effectué, rendant obsolète la conception décrite dans [R3] :

- Les « PanelPlotOutput » **ne sont plus** des implémentations de « ParamOutput ». La récupération et le stockage des données sont exclusivement gérés par la classe « PlotOutput »,
- Les responsabilités des classes « PanelPlotOutput » se limitent :
  - A la construction des paramètres resamplés en fonction du type de plot implémenté,
  - Au tracé du plot correspondant, en accédant aux données stockées par le « PlotOutput ».

La séquence de traitement de ce type de output se décompose, à l'instar de toutes les implémentations d'un « ParamOutput », en trois étapes principales commandées par le « ParameterManager » (cf. [R2]) :

- L'appel à la fonction « establishConnection ». Dans le cadre du module Plot, cette fonction a deux finalités :

- Constituer une liste des paramètres resamplés nécessaires au traitement de la requête, en appelant la fonction « createParameters » de chaque « PanelPlotOutput » (cf. §3.2.5.3.2),
- Etablir la connexion de chacun des paramètres requis, en appelant la fonction « openConnection » de chaque « Parameter ».

- L'appel à la fonction « init ». Cette fonction ce contente d'effectuer un appel séquentiel à la méthode « init » de chaque paramètre utilisé par le tracé,
- L'appel à la fonction « apply » qui, et pour chaque intervalle de temps demandé en requête, applique la séquence de tracé en fonction de l'activation, ou non, de l'option « superpose ».

3.2.5.2.1 Séquence de tracé avec l'option « superpose » à false

Lorsque l'option « superpose » est à false, chaque intervalle de temps sera tracé dans une page différente.

Le diagramme ci-dessous représente cette séquence :

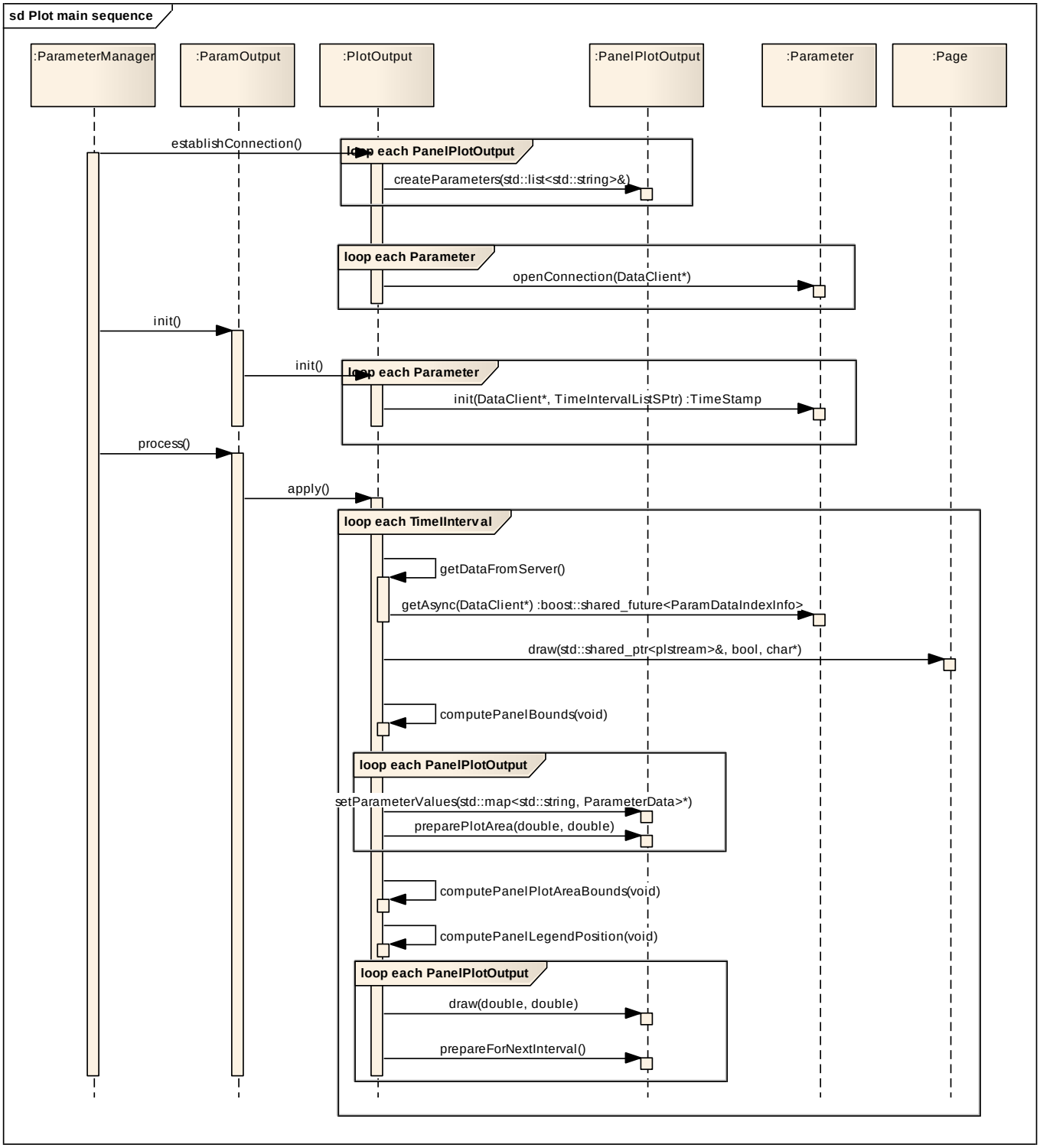


Figure 16 - Diagramme de séquence général du module Plot

Les étapes de cette séquence sont, pour chaque intervalle de temps :

- Récupération et stockage des données, par le biais de l'appel à la fonction « getDataFromServer » (cf. §3.2.5.4),
- Tracé de la nouvelle page,
- Calcul des positions des différents panels dans la page, en fonction du « layout », par le biais de l'appel à la fonction « computePanelBounds » (cf. §3.2.5.9),
- Pour chaque « PanelPlotOutput », un accès aux données stockées par le « PlotOutput » est donné via l'appel à la fonction « setParameterValues »,
- Pour chaque « PanelPlotOutput », un pré-calcul des plots area est effectué via l'appel à la fonction « preparePlotArea »,
- Recalage des plots area des différents panels, en fonction du « layout » (cf. §3.2.5.9), par le biais de l'appel à la fonction « computePanelPlotAreaBounds »,
- Recalage des positions des légendes des différents panels, en fonction du « layout » (cf. §3.2.5.9), par le biais de l'appel à la fonction « computePanelLegendPosition »,
- Tracé des différentes plots area par un appel séquentiel des fonctions « draw » des « PanelPlotOutput »,
- Et enfin, une réinitialisation des différents « PanelPlotOutput », en vue du tracé de l'intervalle de temps suivant, par le biais d'un appel séquentiel aux fonctions « reset ».

3.2.5.2.2 Séquence avec l'option « superpose » à true

Lorsque l'option « superpose » est activée au niveau de la page, le tracé des différents intervalles de temps seront superposés sur la même page.

Cette séquence peut se résumer ainsi :

- Récupération et stockage de toutes les données pour tous les intervalles de temps,
- Tracé de la page,
- Positionnement des panels sur la page en fonction du layout,
- Pour chaque « PanelPlotOutput », un accès aux données stockées par le « PlotOutput » est donné via l'appel à la fonction « setParameterValues »,
- Pour chaque « PanelPlotOutput », un pré-calcul des plots area est effectué via l'appel à la fonction « preparePlotArea »,
- Recalage des plots area des différents panels, en fonction du « layout » (cf. §3.2.5.9), par le biais de l'appel à la fonction « computePanelPlotAreaBounds »,
- Recalage des positions des légendes des différents panels, en fonction du « layout » (cf. §3.2.5.9), par le biais de l'appel à la fonction « computePanelLegendPosition »,
- **Pour chaque intervalle de temps** :

- Tracé des différentes plots area par un appel séquentiel des fonctions « draw » des « PanelPlotOutput »

3.2.5.3 Mécanisme du traitement

3.2.5.3 Préparation des paramètres à tracer

3.2.5.3.1 Rattachement des paramètres à tracer à un « PanelPlotOutput », notion de « ParameterAxes »

La lecture d'un nœud « param », définit au niveau des différents types de visualisation définis dans le fichier requête, effectue un rattachement du paramètre en question au « PanelPlotOutput » correspondant.

Ce rattachement est effectué via l'appel à la fonction « addParam » du « PanelPlotOutput » en instanciant un objet « ParameterAxes » et en le conservant dans la liste « _parameterAxesList » du « PanelPlotOutput ».

Un « ParameterAxes » a pour responsabilités de :

- Maintenir une map des propriétés des « Y séries » définies au niveau du paramètre,
- Maintenir une map des propriétés des « X séries » définies au niveau du paramètre,
- Maintenir un pointeur vers les propriétés d'un « Spectro », si un tracé de ce type est défini au niveau du paramètre,
- Fournir un ensemble de méthodes facilitant l'accès à ces différentes propriétés.

Dorénavant, un « ParameterAxes » n'est qu'un conteneur des différentes propriétés de tracé pour un paramètre donné.

3.2.5.3.2 Création des paramètres « ré-échantillonnés » pour le tracé

La méthode « establishConnection » d'établissement des connections (cf. [R2]) du « PlotOutput » commence par appeler les fonctions « createParameters » de chaque « PanelPlotOutput » afin de créer l'ensemble des « Parameter » ré-échantillonnés (si besoin) pour l'ensemble des séries et spectro à tracer (propriétés portées par les « ParameterAxes » cf. §3.2.5.3.1).

Cette fonction « createParameters » est une fonction virtuelle pouvant être surchargée en fonction des spécificités de chaque type de visualisation (cf. §3.2.5.3.3). L'implémentation de cette fonction au niveau de la classe « PanelPlotOutput » correspond au traitement standard s'appliquant à la visualisation d'une série temporelle.

Les fonctions « createSampledParameter » et « createSampledParameterUnderReferenceParameter » du « PanelPlotOutput » sont utilisées afin de créer les paramètres ré-échantillonnés en fonction des besoins. Ces deux méthodes se contentent d'encapsuler les méthodes de ré-échantillonnages du « ParameterManager » (cf. [R2]).

3.2.5.3.3 Stratégies de ré-échantillonnages des paramètres pour un tracé

La stratégie de ré-échantillonnages des paramètres diffère en fonction du fait qu'un tracé est une série temporelle, un « XYPlot ».ou un « instantPlot »

Il y a également les notions de « resolution » et de « resampling », qui peuvent être définis au niveau de la requête pour chaque séries et spectro, influençant les ré-échantillonnages à appliquer sur chaque paramètre pour un tracé.

Pour une série temporelle :

- Si aucun nœud « resampling » n'est défini le type « auto » est appliqué (cf. point suivant),
- Si le type (attribut du nœud « resampling », lorsque ce nœud est défini) est :
  - « auto » : aucun échantillonnage n'est appliqué sur le paramètre, sauf si « resolution » est défini et que le nombre de points à tracer est supérieur à cette résolution (dans ce cas, le temps d'échantillonnage permettant de se ramener à cette résolution est calculé, et le paramètre ré-échantillonné créé),
  - « manual » : un échantillonnage du paramètre est effectué (si « resolution » est défini, il n'est pas pris en considération) en utilisant le temps d'échantillonnage donné par « value » (attribut du nœud « resampling »),
  - « xparam » : non applicable à ce type de tracé, c'est le type « auto » qui est appliqué,
  - « yparam » : non applicable à ce type de tracé, c'est le type « auto » qui est appliqué.

Pour un « XYPlot » :

- Si aucun nœud « resampling » n'est défini le type « auto » est appliqué (cf. point suivant),
- Si le type (attribut du nœud « resampling », lorsque ce nœud est défini) est :
  - « auto » : c'est le type « xparam » qui est appliqué,
  - « manual » : un échantillonnage des deux paramètres de la série est effectué (si « resolution » est défini, il n'est pas pris en considération) en utilisant le temps d'échantillonnage donné par « value » (attribut du nœud « resampling »),
  - « xparam » : le paramètre Y de la série est ré-échantillonné sur les temps du paramètre de référence X, sauf si « resolution » est défini et que le nombre de points à tracer est supérieur à cette résolution (dans ce cas, le temps d'échantillonnage permettant de se ramener à cette résolution est calculé, et les paramètres sont ré-échantillonnés avec ce temps d'échantillonnage),
  - « yparam » : le paramètre X de la série est ré-échantillonné sur les temps du paramètre de référence Y, sauf si « resolution » est défini et que le nombre de points à tracer est supérieur à cette résolution (dans ce cas, le temps d'échantillonnage permettant de se ramener à cette résolution est calculé, et les paramètres sont ré-échantillonnés avec ce temps d'échantillonnage).

Pour un « InstantPlot » :

- aucun échantillonnage n'est appliqué sur le paramètre tracé.

3.2.5.4 Récupération et stockage des données des paramètres

La classe « PlotOutput » maintient une map « _parameterValues », dont :

- la clé est l'identifiant du paramètre,
- sa valeur une instance de la classe de stockage de données « ParameterData ».

De plus, la classe « PlotOutput » implémente le « VisitorOfParamData » afin d'écrire les données récupérées pour chaque paramètre dans le « ParameterData » le concernant.

Enfin, la récupération des données pour l'intervalle de temps courant est effectué par un appel à la fonction « getDataFromServer » du « PlotOutput » qui va, et pour chaque paramètres demandés, effectuer des appels aux fonctions « getAsync » tant que le serveur renvoie des données et que la notification de changement d'intervalle n'est pas reçue (cf. [R3]).

Un pointeur vers cette map est fourni à chaque instance de « PanelPlotOutput » par le biais de la fonction « setParameterValues ».

Le « gap physic » est également pris en compte dans les fonctions « visitor » du « PlotOutput » lors du remplissage de la structure de stockage « ParameterData » correspondante.

Lorsqu'un trou de données est détecté, un « NaN » est ajouté au niveau du trou de données qui aura pour effet de « couper » le tracé au moment de l'appel des fonctions de PlPlot.

3.2.5.5 Principe de dessin d'un panel :

La finalité du module plot est le tracé du ou des panels composants la page après extraction des paramètres requis.

Le point d'entrée de ce tracé se situe dans la classe **PanelPlotOutput**,et ses classe dérivées (XYPlot, TimePlot, InstantPlot, TickPlot et AsciiPlot).c'est la méthode *draw*. Notons que cette méthode est appelée après appel des méthodes de calcul de la plot Area de chaque panel qui détermine la taille de la plot area.

A titre d'exemple, le diagramme de séquence suivant détaille la construction d'un XY plot. La construction d'un Time plot ou InstantPlot respecte quasiment les mêmes séquences d'appel :

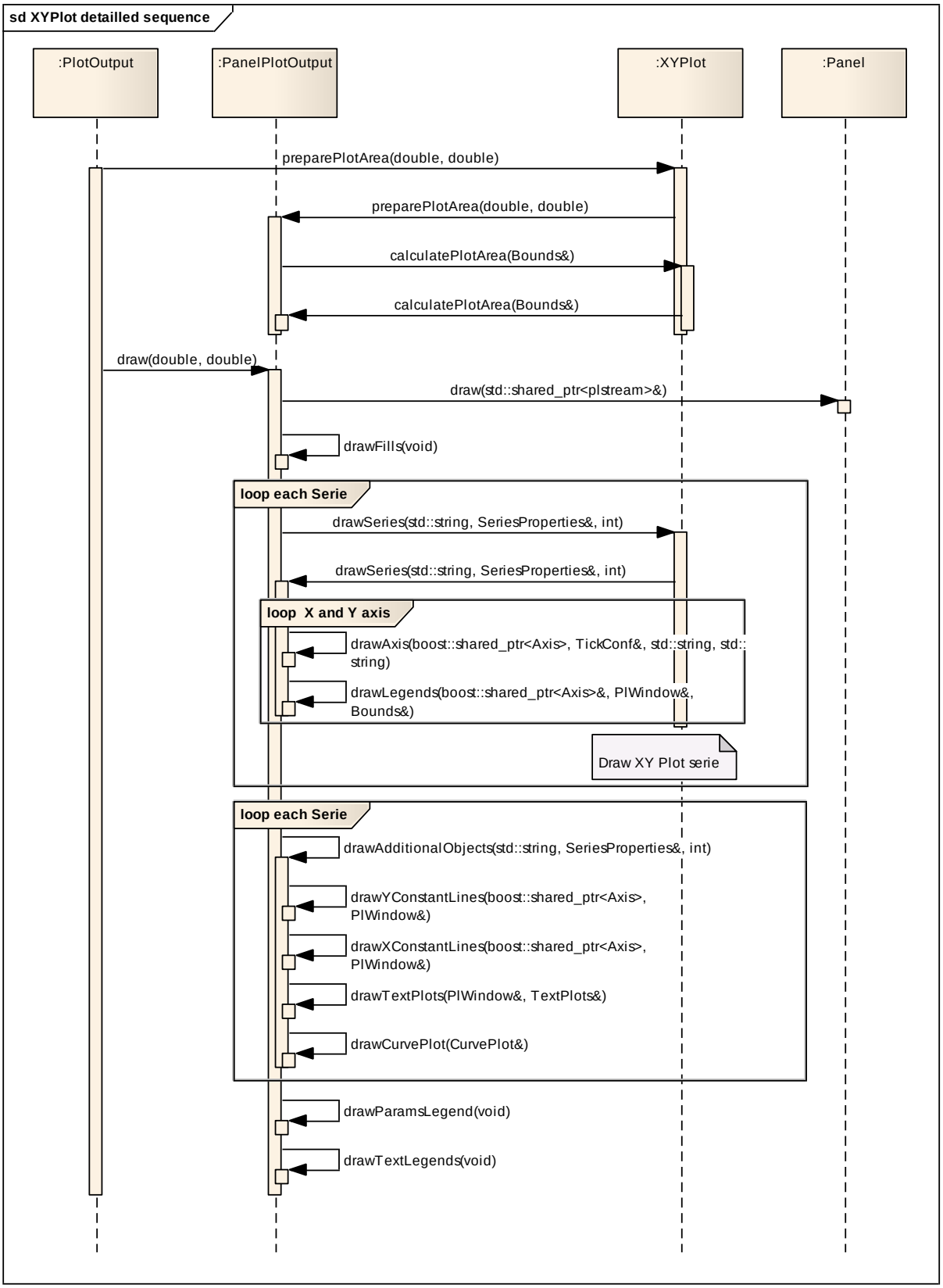


Figure 17 - Diagramme de séquence construction XYPlot

Les classes XYPlot, TimePlot, InstantPlot, TickPlot et AsciiPlotOutput héritent de la classe PanelPlotOut qui définit un ensemble de fonctions virtuelles (pures ou non) surchargées par les classes filles.

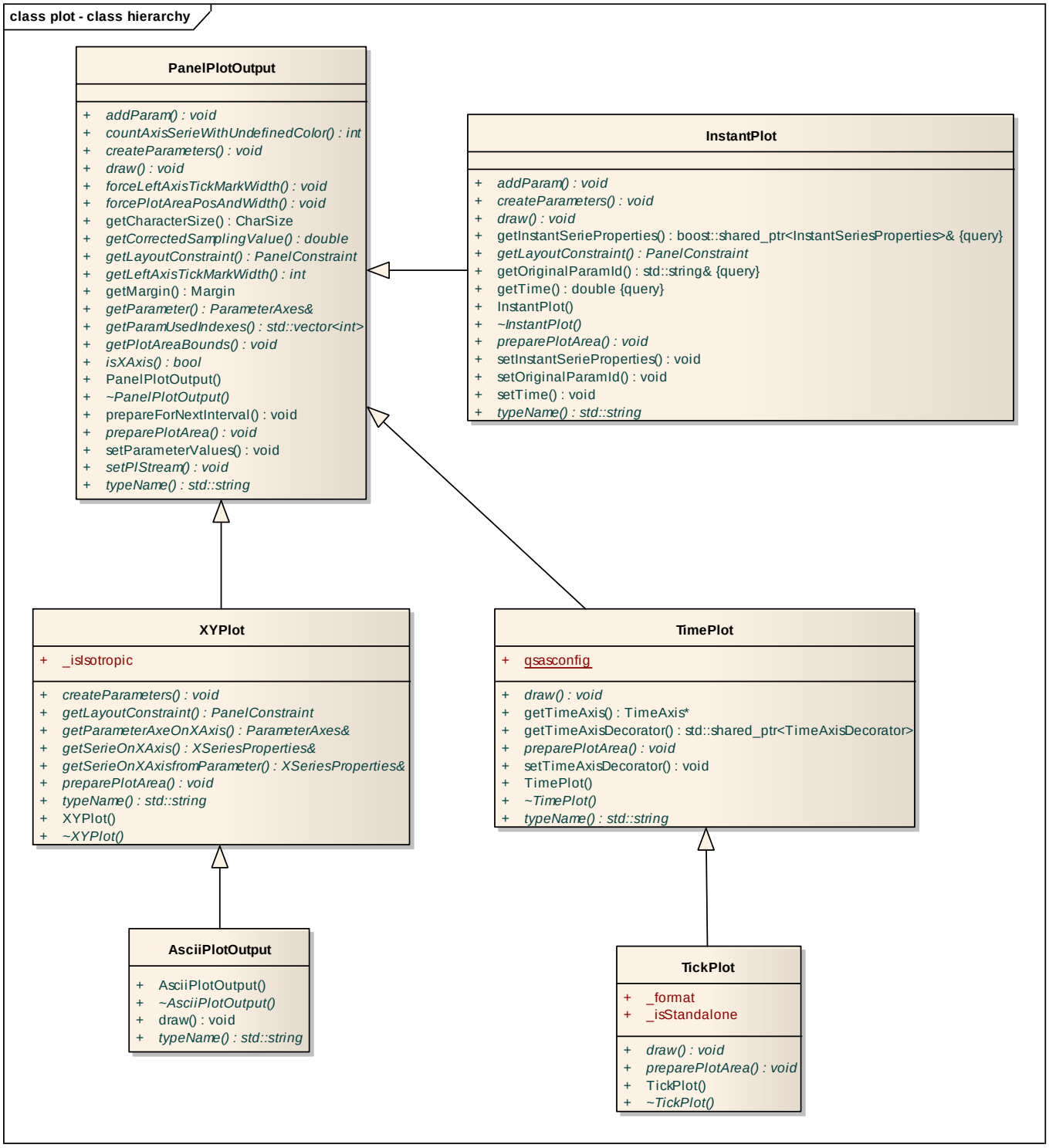
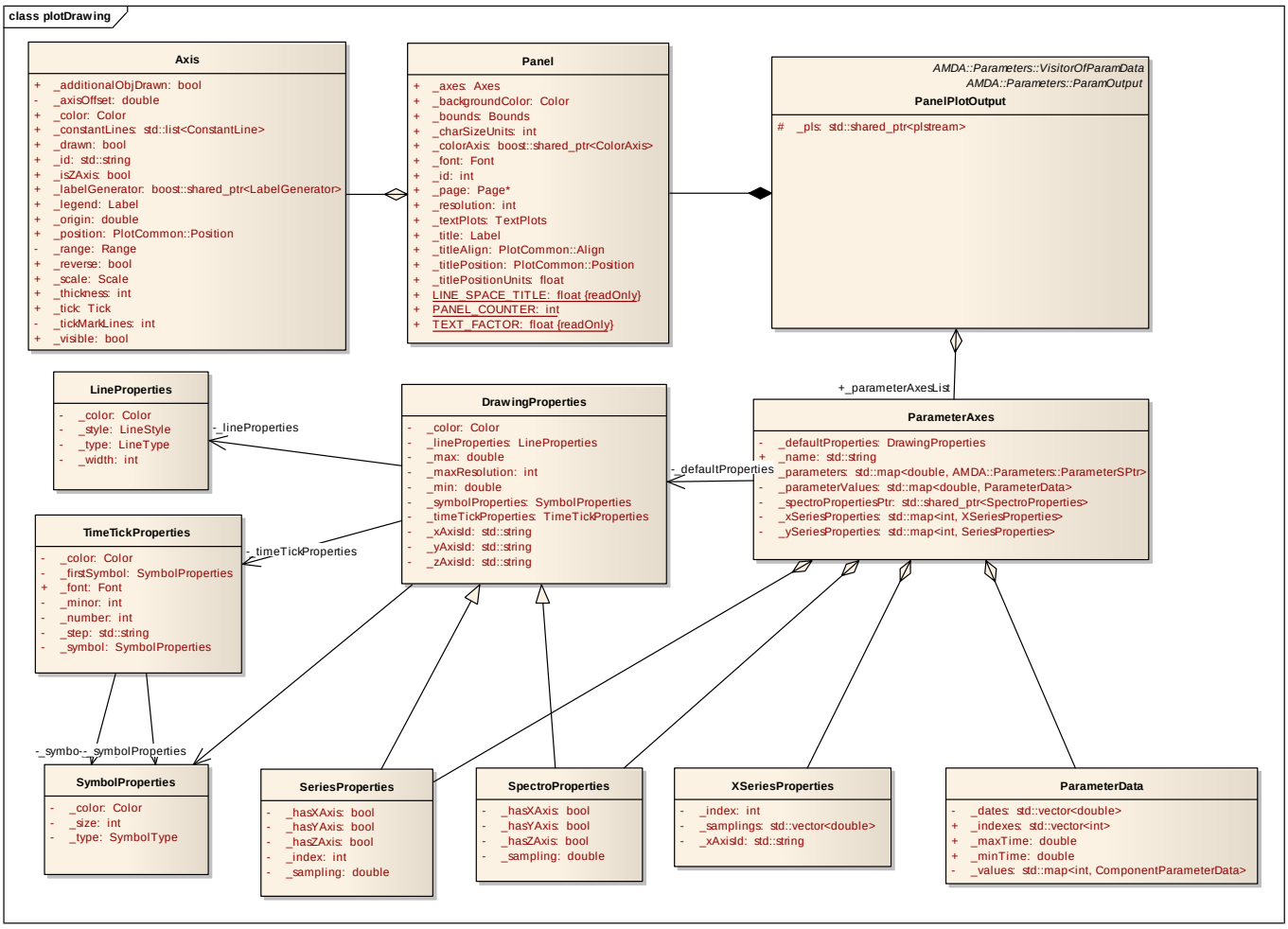


Figure 18 - Classe PanelPlotOutput et ses classes dérivées

Le diagramme de classe suivant représente les différentes propriétés graphiques associées à un tracé de paramètre qui seront exploitées par les méthodes drawXYZ des classes **PanelPlotOutput** et de ses classes dérivées pour réaliser le rendu graphique d'un panel en utilisant la librairie PlPlot.



3.2.5.6 Tracé des zones de remplissage (Fills)

Pour une courbe de type time plot, il est possible de colorer les zones situées entre une valeur Y constante et une courbe (cas 1) ou entre deux courbes (cas 2) en faisant varier la couleur en fonction du côté (supérieur ou inférieur) dans lequel on se situe.

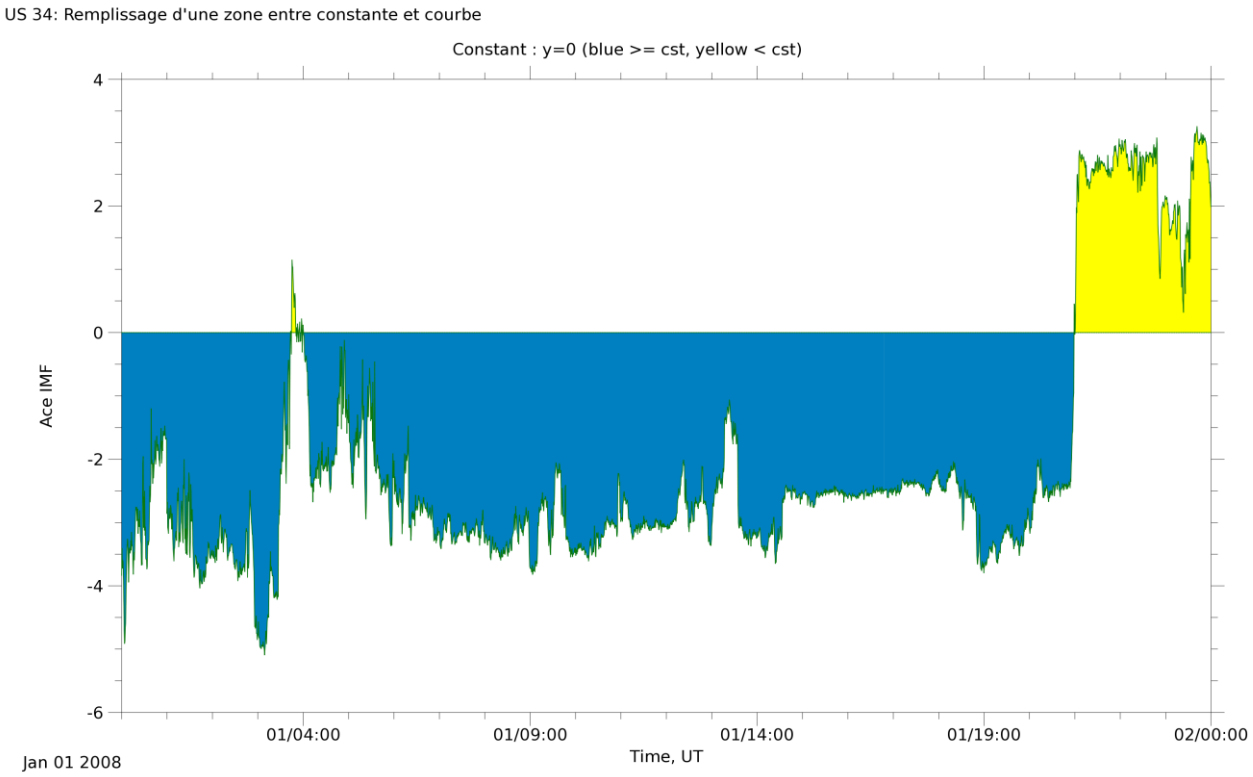


Figure 19 - Exemple de coloration entre une constante et une courbe

Pour ce faire, les nœuds fillSerieConstant et fillSerieSerie ont étés ajoutés au shéma XSD (timeplot.xsd). Ils sont constitués des attributs permettant d'identifier le type de remplissage demandé :

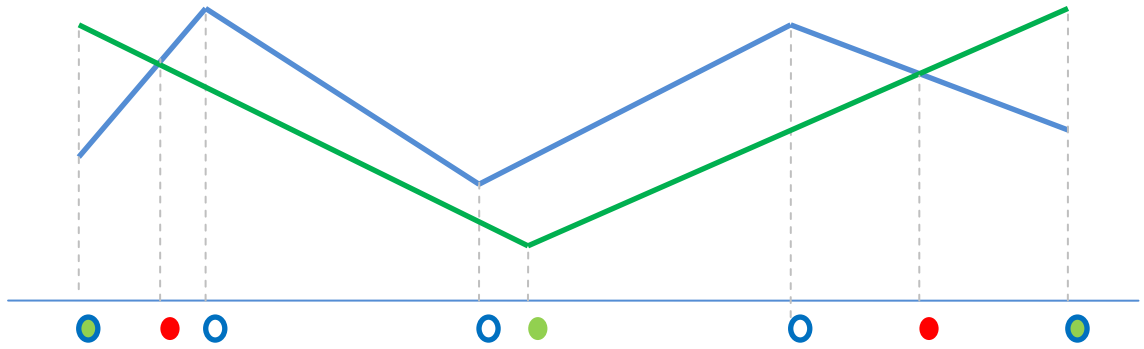
- Identifiant de la série ou de la constantLine (serieId et constantId dans le cas 1, firstSerieId et secondSerieId dans le cas 2)
- Couleur dans le cas où le tracé est au-dessus (colorGreater) ou en dessous (colorLess) de l'autre tracé.

Ces nœuds sont lus par la classe ***TimePlotNode*** en instanciant une classe de type ***FillsNode*** qui alimente à son tour une instance de classe ***FillSerieConstant*** ou ***FillSerieSerie*** en fonction du type de remplissage demandé.

Le tracé de la zone de remplissage est réalisé en premier pour un panel afin de figurer en arrière-plan sur le graphique produit (voir méthode *drawFills* de la classe ***TimePlot***).

Afin de factoriser l'algorithme de tracé, aucune différence n'est faite entre la coloration entre une constante et une courbe ou entre 2 courbes : Dans le cas d'une constante on génère une courbe fictive constituée de deux points ayant même ordonnée et startTime / stopTime de l'autre courbe en abscisse (voir méthode *drawFills*).

L'algorithme de tracé de ces zones de remplissage est le suivant (en considérant 2 courbes) :



- Dans un même tableau temporel, fusionner, trier et supprimer les doublons des abscisses des points constituants les deux courbes (points verts et bleus),
- Pour chaque intervalle de temps constitué par le tableau produit (points verts et bleus) ; rechercher les éventuelles intersections (points rouges) et les ajouter au tableau temporel.
- Parcourir le tableau des temps, pour chaque intervalle, déterminer les coordonnées des deux segments et tracer la zone qui les sépare en utilisant la couleur appropriée en fonction de la position relative (en dessus ou en dessous) de ces derniers.

3.2.5.7 Tracé des « AdditionalObjects »

Les « AdditionalObjects » sont des objets tracés dans une plot area, mais qui ne dépendent pas d'un « Parameter ».

Ils sont utilisés afin d'ajouter des informations supplémentaires à un plot.

3.2.5.7.1 Principe général

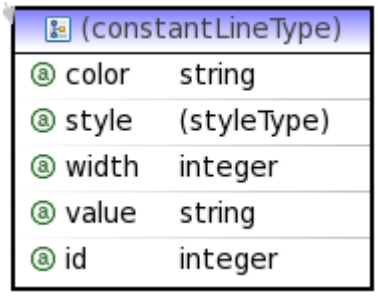
A ce jour, les « additional objets » pris en charge par AMDA NG sont :

- Les « constant lines » horizontales ou verticales rattachées à des axes,
- Les « text plots » permettant d'afficher un texte à l'intérieur de la plot area
- Les « curve plots » supportant le tracé des couirbes paramétrées sur un plot

Ces objets, s'ils sont définis dans la requête sont tracés après le tracé des axes et des séries, mais avant le tracé des légendes (Cf. 3.2.5.8 Tracé d'une légende associée à un plot)

3.2.5.7.2 Objet « Constant line»

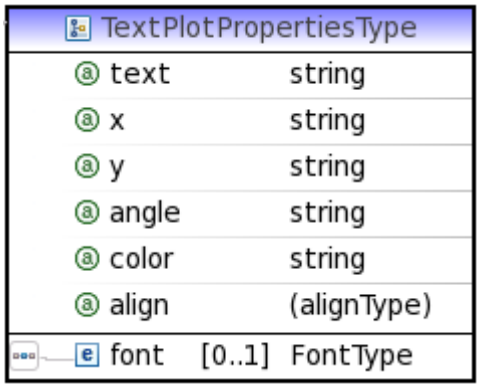
Ces objets sont rattachés aux axes définis pour la requête. Ils disposent d'attributs de style (couleur, style, épaisseur) mais aussi d'un attribut « value » qui définit l'abscisse ou l'ordonnée de la ligne constante tracée



Les objets « constant lines » sont utilisés pour le tracé de zones de remplissage entre une courbe et une valeur constante (Cf. 3.2.5.6 Tracé des zones de remplissage (Fills)), dans ce cas particulier, l'attribut « id » de l'objet permet d'identifier la constant line servant de référence au remplissage.

3.2.5.7.3 Objet « Text »

Ces objets permettent de tracer un texte quelconque dans la plot area en jouant sur un nombre important d'attributs.



La figure suivante présente un exemple des possibilités offertes. Il est important de noter le « crop » effectué sur le tracé pour que le texte ne sorte pas de la plot area.

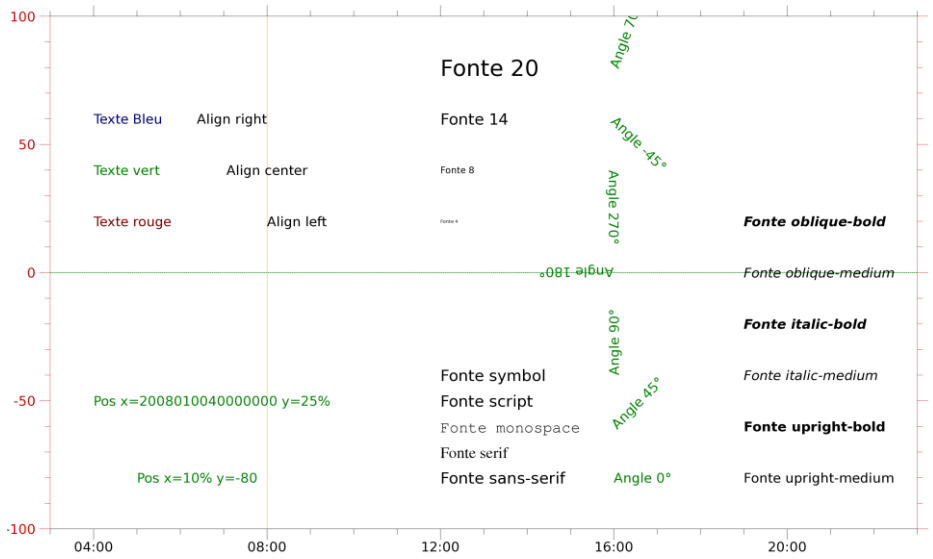


Figure 20 - Exemple d'utilisation de text plots

3.2.5.7.4 Objet « Curve »

L'objet « Curve » désigne une courbe paramétrée ne pouvant s'appliquer qu'à un plot de type « XYPlot ».

Les propriétés d'un tel objet sont portées par la classe « CurvePlot » contenant notamment :

- Un rattachement à l'identifiant d'une série existante du plot, afin de faire le lien vers les axes de tracé de la courbe paramétrée,
- Des propriétés de tracé : la couleur, l'épaisseur et le style du trait.

Une liste d'instances de cet objet peut être définie au niveau d'un « Panel ». Ils sont créés et initialisés au chargement d'un nœud de type « curvePlot » depuis le fichier XML définissant la requête.

Un nœud « function », rattaché à un nœud « curvePlot », est également lu. Il permet de définir le nom de la fonction paramétrée à utiliser et une liste d'attributs utilisés par la fonction désignée. Ces informations sont stockées par une instance d'un « CurveFunctionWriter », rattachée à l'instance de la classe « CurvePlot » associée.

Le tracé d'un tel objet se base sur l'appel de la fonction « getPointList » d'une instance d'un « CurvePlot » qui fournit la liste des points de la courbe à tracer. Pour se faire, elle appelle la fonction « call » du « CurveFunctionWriter » qui, en fonction du nom de la fonction à appliquer, récupère le process de type « CurveFunctionProcess » correspondant, et l'exécute.

L'ensemble des « CurveFunctionProcess » sont définis dans le plugin « CurvePlot ».

3.2.5.8 Tracé d'une légende associée à un plot

Des légendes liées à un plot peuvent être définies au niveau de la requête.

Elles sont portées par un nœud « legends » et se déclinent :

- Sous forme d'un nœud « paramsLegend » définissant une légende dont la responsabilité est de décrire les paramètres qui sont tracés par le plot. Ce type de légende peut être tracé à l'intérieure de la plot area, ou à l'extérieure. Le contenu même de la légende est automatiquement déterminé en fonction de ce qui est tracé dans le plot.
- Sous forme d'un (ou plusieurs) nœud(s) « textLegend », situé à l'extérieur de la plot area, et pouvant être positionné aux quatre points cardinaux. Ce type de légende est associé à un texte à afficher.

Ce type de légende contient des informations sur les séries tracées dans un plot. Ces informations sont écrites lors de l'appel à la fonction virtuelle « addSerieToParamsLegend » pouvant être surchargée en fonction du type de plot considéré.

Les propriétés de tracé sont contenues dans une instance de la classe « ParamsLegendProperties », portée par la classe « Panel ».

Enfin, son tracé est effectué par la fonction « drawParamsLegend » du « PanelPlotOutput ».

3.2.5.9 Gestion des layout

AMDA NG permet de disposer plusieurs panels au sein d'une même page.

L'emplacement de ces panels sur la page peut être définit :

- manuellement au niveau de la requête en spécifiant pour chacun des panels sa position dans la page (on parlera ici de layout manuel),
- en utilisant un layout qui se charge de positionner automatiquement les panels sur la page (layout auto ou vertical).

Afin d'optimiser l'arrangement des panels sur la page, la notion de contrainte a été introduite. Elle définit la préférence de positionnement d'un panel au sein de la page. A ce jour, deux contraintes ont été identifiées :

- MaxWidth : définit la contrainte d'un panel qui souhaite occuper la largeur maximale sur la page (cas des TimePlots et des TickPlots)
- Square : définit la contrainte d'un panel qui souhaite avoir la même largeur que hauteur (cas des XYPlot et InstantPlot).

En fonction du layout utilisé pour la page, les panels sont automatiquement organisés en fonction de leurs contraintes respectives :

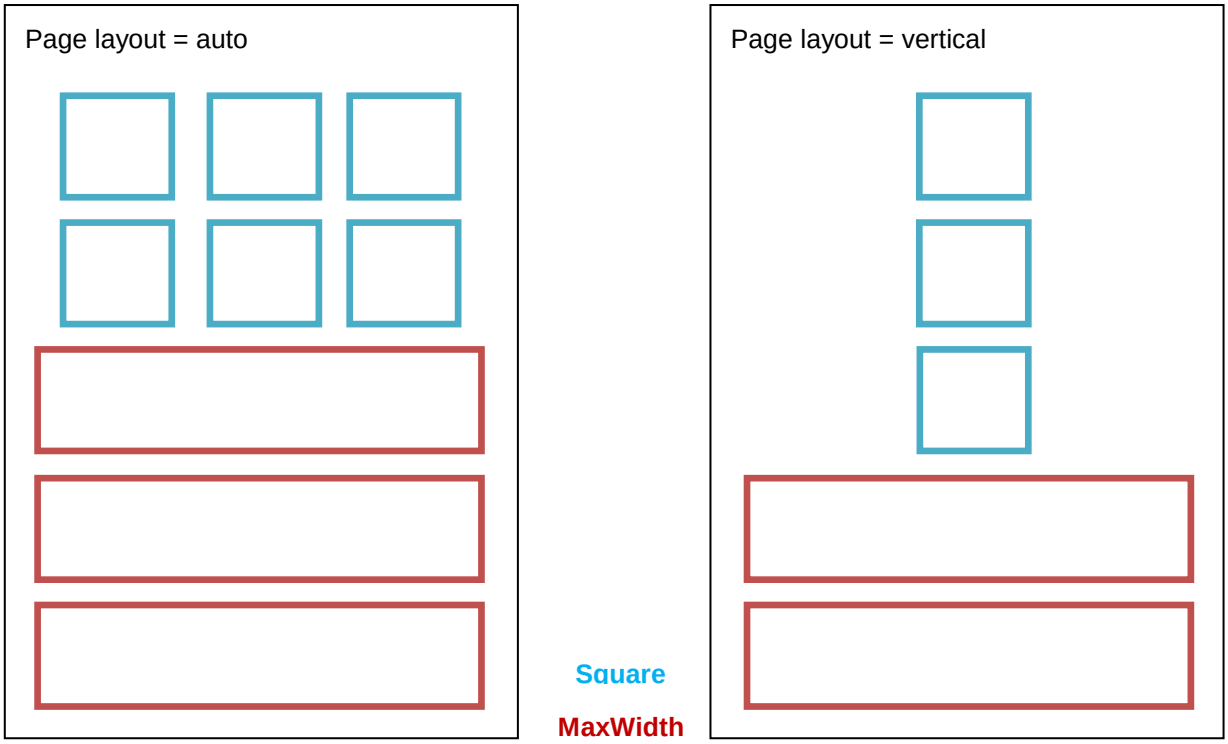


Figure 21 - Organisation des panels en fonction de leurs contraintes

Dans le cas d'un layout auto, les panels ayant une contrainte de type MaxWidth (en rouge) sont empilés les uns sur les autres, les panels ayant une contrainte de type Square (en bleu) sont disposés au-dessus en maintenant une largeur et hauteur identiques. Plusieurs panel (square peuvent figurer sur une même ligne)

Dans le cas d'un layout vertical, tous les panels sont empilés les uns sur les ordres dans l'ordre d'apparition dans la requête de plot. Les panels ayant pour type de contrainte Square sont centrés sur la page.

Différents paramètres permettent d'affiner l'organisation des panels sur la page pour les layout de type auto ou vertical :

- panelHeight : ce paramètre définit la hauteur par défaut des panels[MaxWidth], ainsi que la hauteur et largeur par défaut des panels[Square]
- panelSpacing : ce paramètre définit l'espacement souhaité entre les panels de tous types.
- expand : ce paramètre indique que l'on souhaite que la taille des panels s'adapte automatiquement pour occuper une surface maximale sur la page.
- firstPanelFactor : ce paramètre définit un facteur multiplicateur pour la hauteur du premier panel ayant pour contrainte MaxWidth sur la page

Toutes les dimensions, hauteur par défaut ou espacement précisés pour le layout sont exprimées dans un intervalle [0..1] à l'intérieur des marges de la page

Comme nous l'avons vu au paragraphe (3.2.5.1 Présentation des notions de page, panel et plotArea), la zone de plot area s'adapte en fonction de la taille du panel, du nombre d'axes, des légendes…).

L'alignement automatique des panels au sein de la page pour les layout de type auto ou vertical ne sous-entend donc pas forcément l'alignement des plots area à l'intérieur de ces panels. Le problème est similaire pour l'alignement des légendes sur l'axe Y des timeplots.

Un mécanisme a donc été mis en place afin d'homogénéiser à la fois la taille des plots area et la position des titres pour les panels ayant une contrainte de type MaxWidth.

Le diagramme de séquence suivant détaille le principe de pré-calcul de la position des panels et ce mécanisme d'alignement.

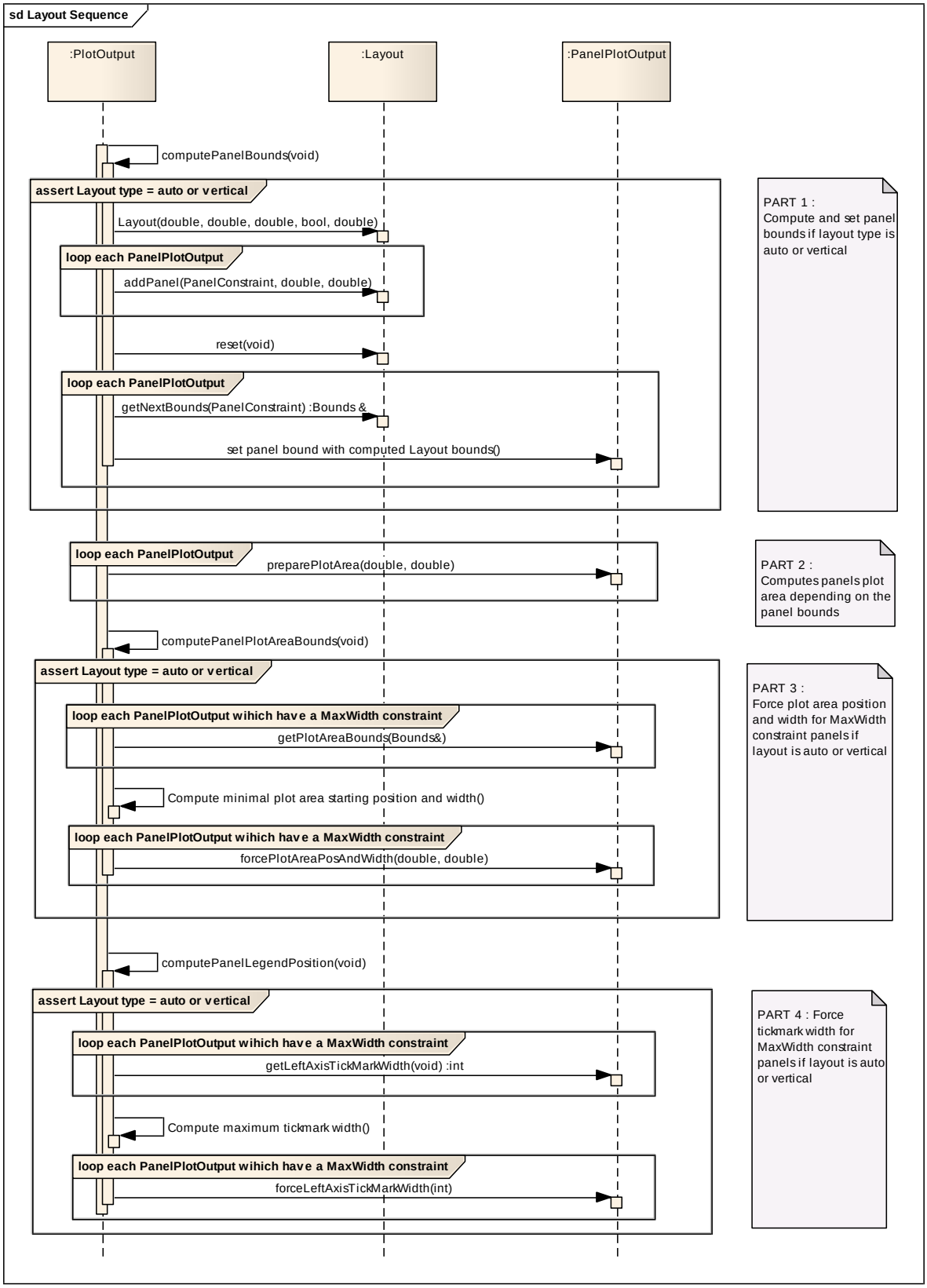


Figure 22 - Gestion des layouts

La gestion de la position des panels à l'intérieur d'un layout auto ou vertical est mis en œuvre au travers de deux classes ***LayoutAuto*** et ***LayoutVertical*** héritant de la classe ***Layout*** :

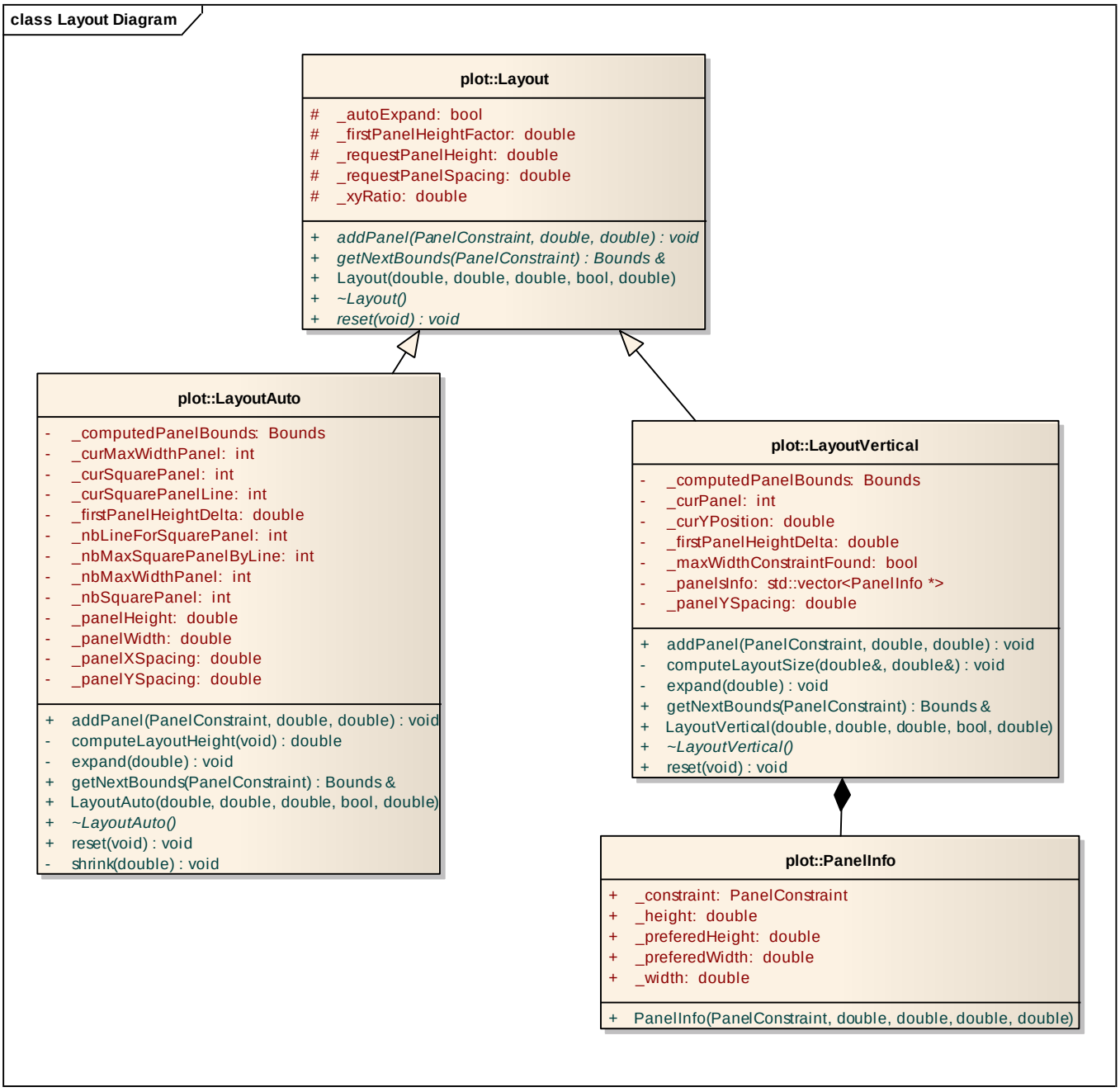


Figure 23 - Classes relatives à la gestion des layouts

La classe ***Layout*** définit trois méthodes virtuelles pures mises en œuvre (spécialisées) par les classes filles :

- La méthode *addPanel* permet d'ajouter des panels au layout en précisant leur contrainte et éventuellement des dimensions préférées (utilisé par les layout de type vertical)

- La méthode *reset* est appelée pour initialiser le layout avant appels successifs à la méthode *getNextBounds,*
- La méthode *getNextBounds*.est appelé pour chacun des panels préalablement ajoutés, elle renvoie les dimensions (Bounds) du panel calculées par le layout.

Pour les layout de type vertical, il est possible de préciser pour chacun des panels la hauteur et la largeur souhaitée. Ces « souhaits » sont mémorisés dans la classe ***PanelInfo*** associée à la classe ***LayoutVertical*** lors de l'appel à la méthode *addPanel*, la fonction de calcul des dimensions du panel (*getNextBounds)* prend en considération ces paramètres pour tenter de satisfaire ces dimensions.

3.2.5.10 Instant plot

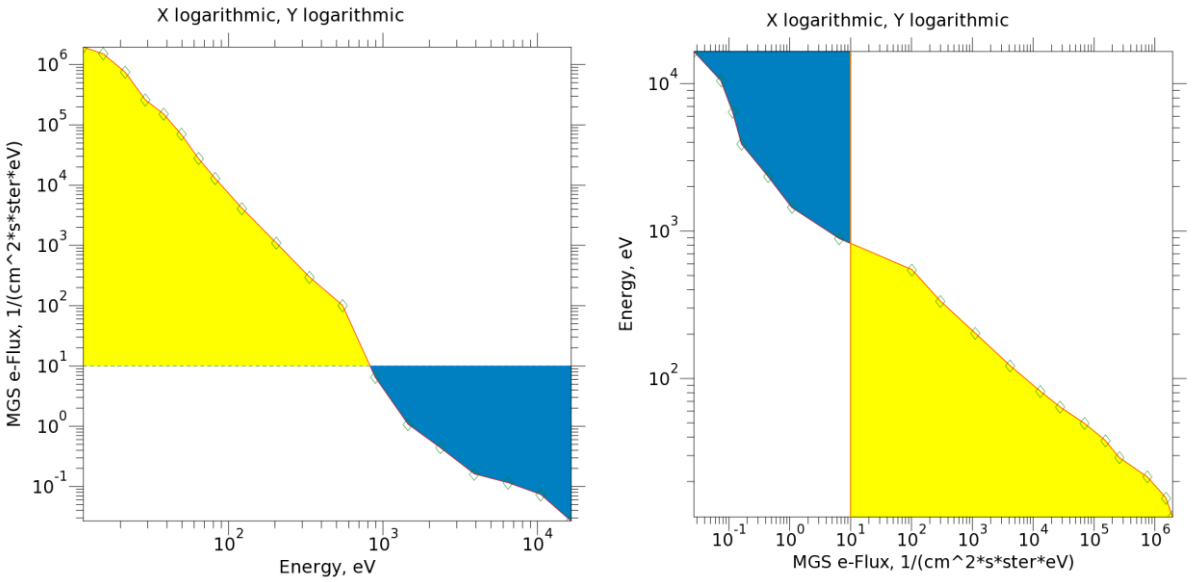
Ce nouveau type de plot a été ajouté afin de permettre le tracé :

- De données de type « Tab1D » à un instant donné, par le biais de la définition d'une « iserie »,
- De données de type « Tab2D » à un instant donné, par le biais de la définition d'un « ispectro ».

3.2.5.10.1 Présentation générale

La classe ***InstantPlot*** s'appuie exactement sur les mêmes principes que ceux utilisés par les classes ***TimePlot*** et ***XYPlot***, elle dérive elle aussi de la classe ***PanelPlotOutput.***

L'InstantPlot supporte également l'ajout de constantes verticales ou horizontales au tracé, ainsi que la définition de zones de remplissages entre une constante et la série tracée (la zone de remplissage ne peut donc s'appliquer que pour une « iserie »).



Ne pouvant pas supposer de la monotonie des valeurs du paramètre, nous avons considéré que la table (ou l'index) associée à ce dernier l'était. De ce fait, le remplissage se fait :

- par rapport à une constante de type Y = cste dans le cas où la table (ou l'index) est sur l'axe X,
- par rapport à une constante de type X = cste dans le cas où la table (ou l'index) est sur l'axe Y.

3.2.5.10.2 Cas d'une « iserie »

Ce type de tracé s'applique à des données de type « Tab1D ».

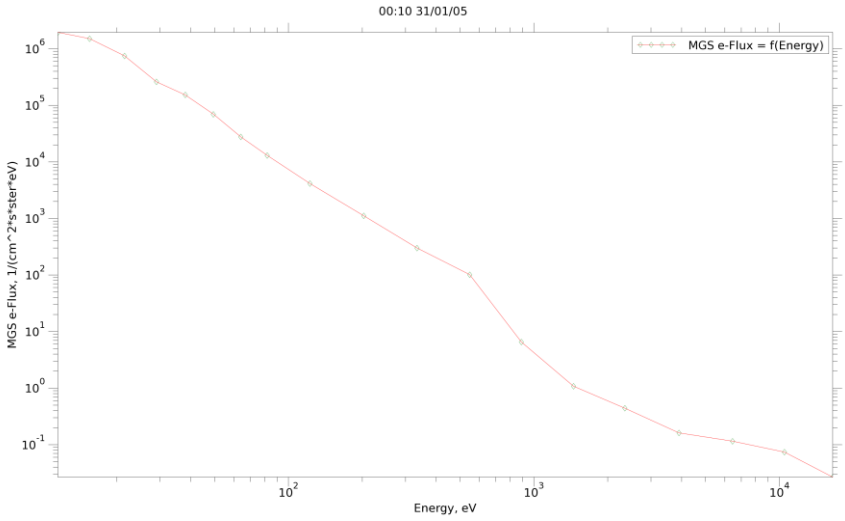


Figure 24 - Exemple de tracé d'une "iserie"

La notation « ***iserie »*** (pour instant série) a été introduite au niveau de la requête pour définir une série associée à un ***InstantPlot***. Les propriétés ce cette « iserie » sont gérées par la classe ***InstantSeriesProperties*** qui hérite de ***SeriesProperties.***

Pour un instant donné, les valeurs affichées par défaut en en X représentent la table associée au paramètre et l'axe Y la valeur du paramètre pour les valeurs de la table.

L'attribut « tableOnXAxis » de la requête permet d'inverser les axes X et Y pour adapter la représentation à des besoins spécifiques.

Le temps de coupe dessiné est précisé en tant qu'attribut dans la requête, dans le cas contraire, le milieu de l'intervalle de la requête est dessiné.

Dans le cas où le aucune table n'est associée au paramètre observé, les valeurs de la table sont remplacées par leur index.

3.2.5.10.3 Cas d'un « ispectro »

Ce type de tracé s'applique à des données de type « Tab2D ».

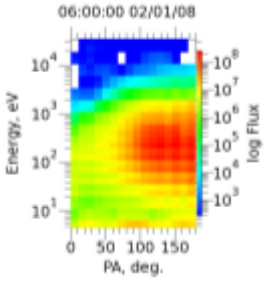


Figure 25 - Exemple de tracé d'une "ispectro"

La notation « ispectro***»*** (pour instant spectro) a été introduite au niveau de la requête pour définir un spectro associé à un ***InstantPlot***. Les propriétés ce ce « isectro » sont gérées par la classe ***InstantSpectroProperties*** qui hérite de ***SectroProperties.***

Pour un instant donné, les valeurs affichées par défaut en X représentent la table associée à la première dimension du paramètre, alors que l'axe Y porte les valeurs de la table associée à la deuxième dimension du paramètre.

L'attribut « dimOnXAxis» de la requête permet d'inverser les axes X et Y pour adapter la représentation à des besoins spécifiques.

Le temps de coupe dessiné est précisé en tant qu'attribut dans la requête, dans le cas contraire, le milieu de l'intervalle de la requête est dessiné.

Dans le cas où aucune table n'est associée à l'une des dimensions du paramètre observé, les valeurs de la table sont remplacées par leurs indexes.

3.2.5.11 *Tracé des plot d'orbite*

AMDA_Kernel supporte le tracé de plot d'orbites comme le montre la figure suivante :

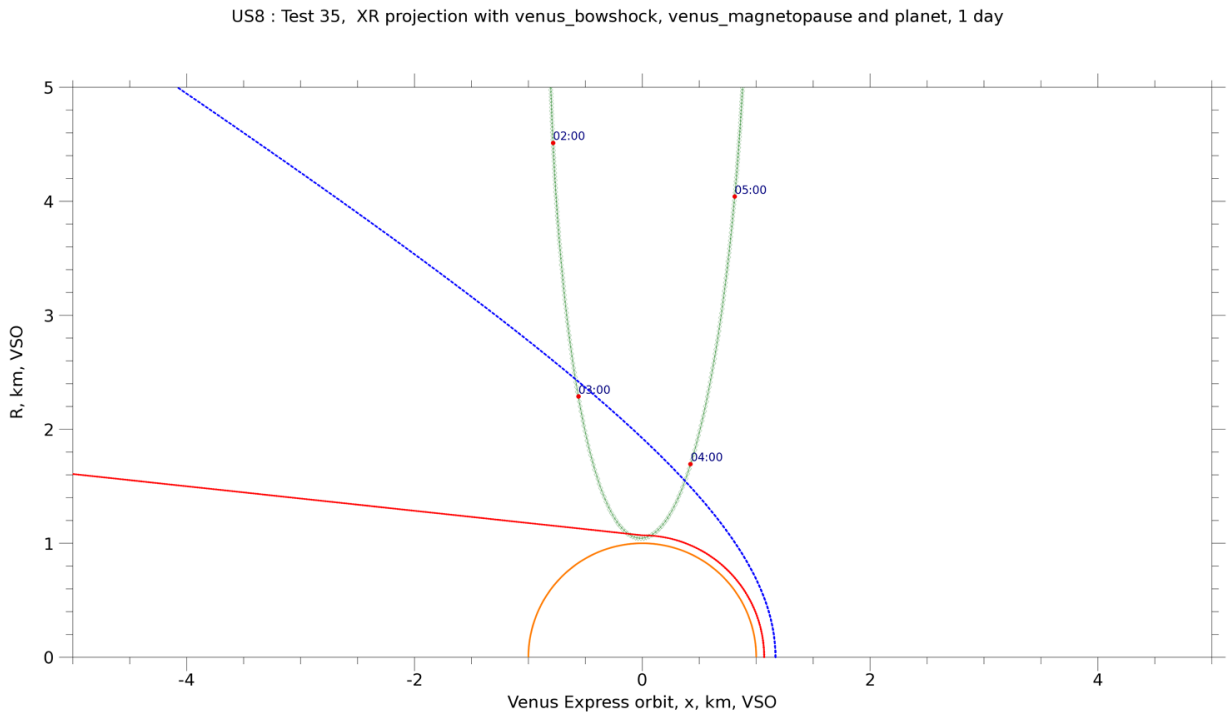


Figure 26 - Exemple de tracé de plot d'orbite

Afin d'arriver à ce résultat, la série ***orbitserie*** a été définie au niveau du xyPlot. Ce type de série définit un type de tracé pour lequel les attributs spécifiques au plot d'orbite sont définis :

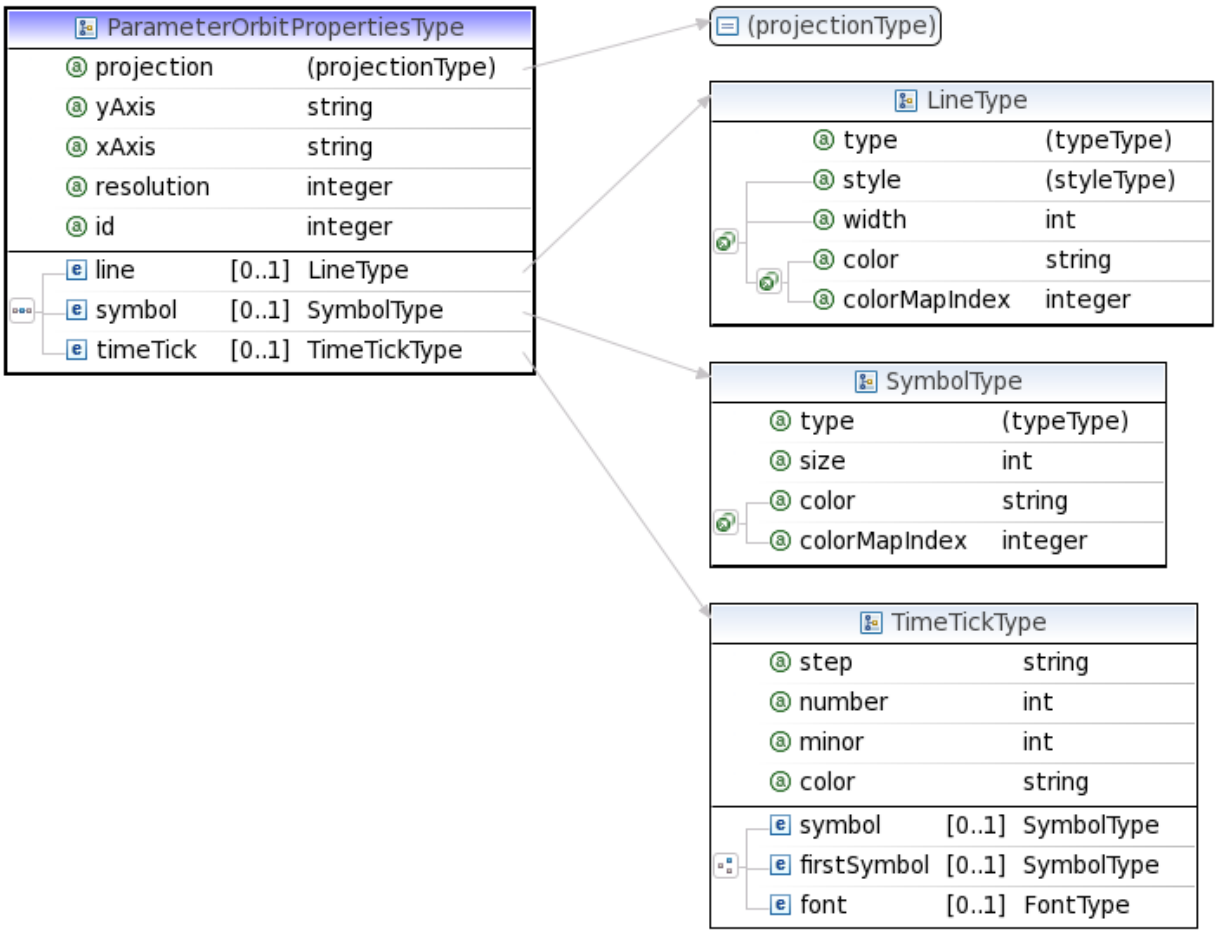
- Le type de projection (XY, XZ, YZ, XR, YR et ZR)
- Les axes de rattachement (xAxis et yAxis)
- La résolution éventuelle de tracé,
- L'id de l'orbitsérie (utilisé par les curvePlot)

Le tableau suivant représente les valeurs portées par les axes X et Y en fonction du type de projection pour un vecteur v [3]:

Type de projection	Valeur associées à l'axe X	Valeur associées à l'axe Y
XY	**V[0]**	**V[1]**
XZ	**V[0]**	**V[2]**
YZ	**V[1]**	**V[2]**

Type de projection	Valeur associées à l'axe X	Valeur associées à l'axe Y
XR	**V[0]**	**Sqrt (V[1]* V[1]+ V[2]* V[2])**
YR	**V[1]**	**Sqrt (V[0]* V[0]+ V[2]* V[2])**
ZR	**V[2]**	**Sqrt (V[0]* V[0]+ V[1]* V[1])**

L''élément orbitserie et les 6 types de projection autorisés sont définis au niveau du schéma plot.xsd dans la définition du ParameterOrbitPropertiesType. Les éléments de ce type sont autorisés au niveau de l'élément <param> d'un xyPlot (Cf. xyPlot.xsd).



La mise en œuvre au niveau d'un xyPlot consiste lors de la lecture d'un élément de type orbitserie à créer dynamiquement une instance de SeriesProperties (axe Y) et une instance de XSeriesProperties (axe X) et à positionner pour chacune de ces instances l'index correspondant à la composante du vecteur observé. Lorsque la projection fait intervenir la composante « R », l'index est positionné à -1 et la formule de calcul est renseigné pour l'axe Y (Cf. méthode setComputeExpression).

Lors de la création des paramètres nécessaires au tracé du xyPlot (méthode createParameters), si une expression est définie pour l'axe des Y, un nouveau paramètre est ajouté correspondant à l'expression définie (Cf. getOneParameterFromExpression).

3.2.5.12 Tracé des plot de type errorBar

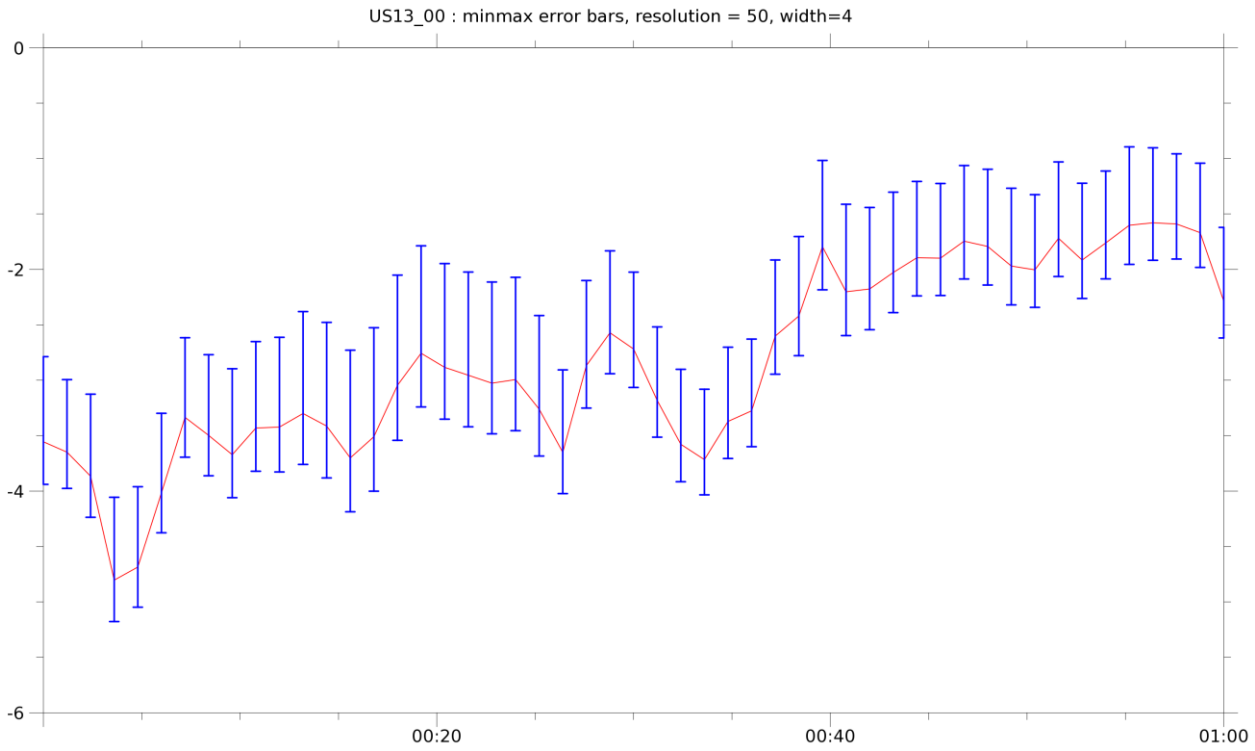
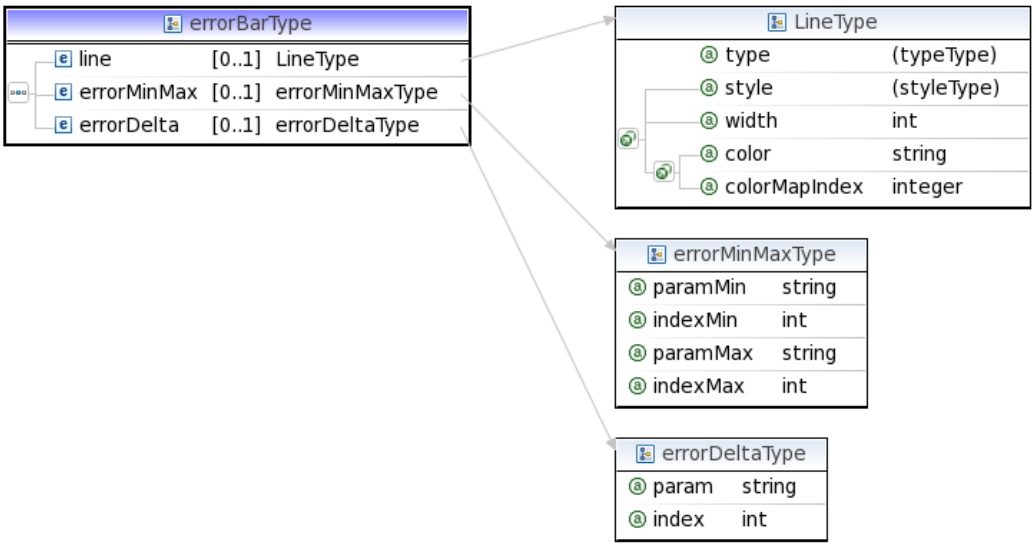


Figure 27 - Exemple de tracé d'error bar

AMDA permet le tracé d'error bar sur un plot. Ces barres d'erreur peuvent représenter les écarts min et max par rapport à une série ou bien un delta para rapport à cette série.

Dans le premier cas un élément de type errorMinMax sera définit au niveau de la requête, dans le deuxième cas l'élément sera de type errorDelta. La représentation (partielle) du schéma XSD de la figure suivante précise les paramètres autorisés pour les errors bar.



La mise en œuvre des erorrs bar est réalisée dans le module plot d'AMDA en effectuant les opérations spécifiques suivantes :

- Lecture d'un nœud de type errorBar au niveau de la ySerie de la requète et alimentation des _errorBarProperties (structure identique pour les 2 types d'erreur errorMinMax et errorDelta).
- Lors de la création des paramètres requis pour le plot, si les error bars properties sont définies pour la séries, création de 2 paramètres à partir des expression « param – paramMin » et « param + paramMax
- Lors du tracé d'un time plot, tracé des paramètres calculés à partir des 2 expressions construites précédemment en utilisant la fonction erry (Draw y error bar) de plplot

3.2.5.13 Tracé des « DecoratorPlot »

Un « DecoratorPlot » consiste en un tracé de type « timePlot » pouvant être :

- Indépendant : Un panel lui est alloué, dans lequel il aura la responsabilité de tracer son propre axe de temps,
- Dépendant d'un autre « timePlot » : Il viendra s'insérer dans le panel du « timePlot » auquel il est associé et partagera le même axe de temps.

3.2.5.13.1 Tracé des « TickPlot »

Ce tracé a été implémenté lors de la deuxième prestation du projet (cf. [R3]).

La classe « TickPlot » hérite dorénavant de la classe « DecoratorPlot ».

3.2.5.13.2 Tracé des « StatusBar »

La classe « StatusBar » hérite d'un « DecoratorPlot ».

Ce tracé a pour vocation de représenter un paramètre sous forme de couleurs représentants des états.

Figure 28 - Exemple de tracé d'un StatusBar

3.2.5.14 Tracé d'un « epochPlot »

Ce type de plot utilise un « epochAxis » afin de représenter un événement par rapport à un temps de référence.

Les événements et les temps de référence sont fournis par un fichier catalogue définis au niveau du nœud « times » d'une requête.

Ce type de tracé, avec l'option « superpose » activée au niveau de la page (cf. §3.2.5.2.2), permet d'obtenir un plot de type « epoch superposition » tel que présenté ci-dessous :

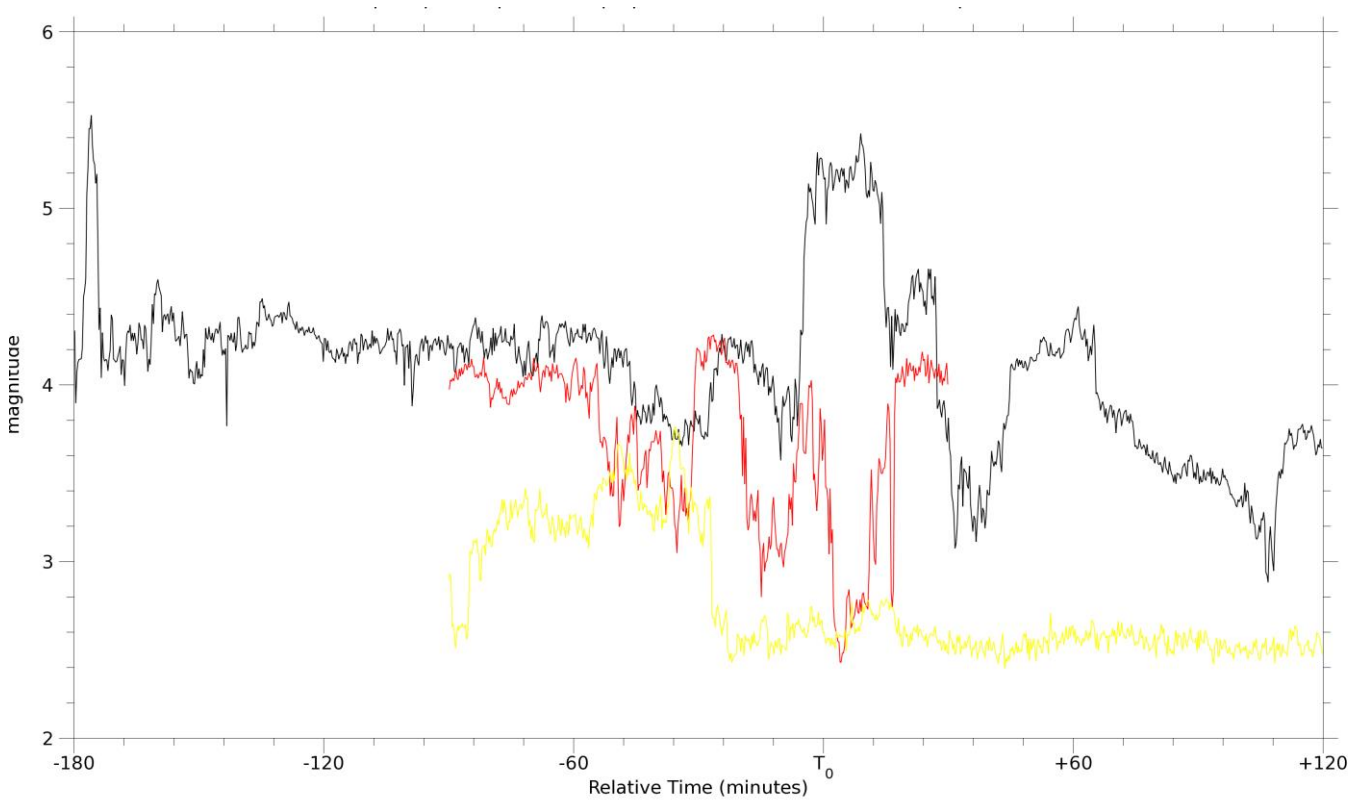


Figure 29 - Tracé de type "epoch superposition"

L'option « normalized » d'un axe « epochAxis » permet de normaliser le tracé de chacun des événements.

3.2.6 Module TimeTableCatalog

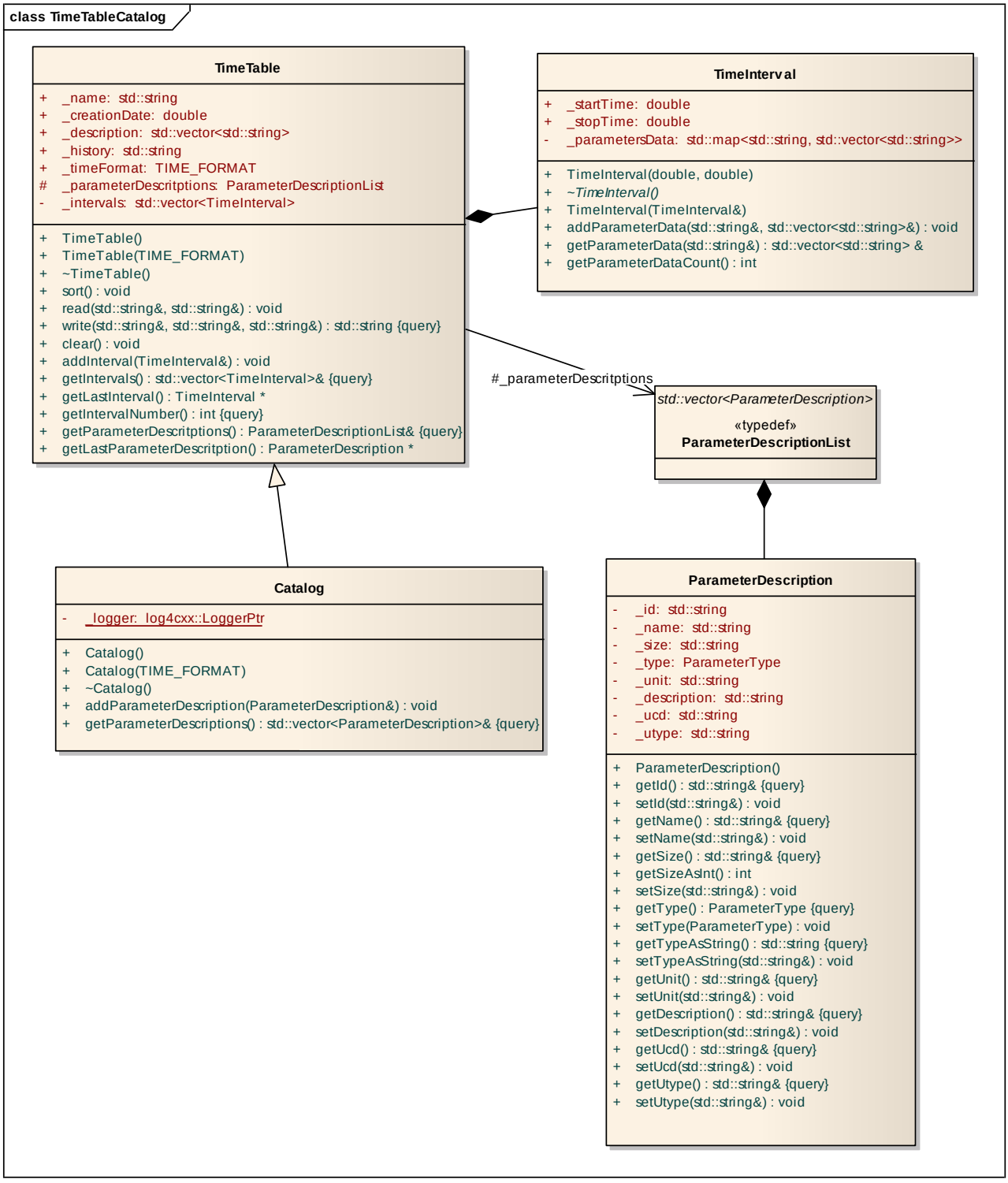


Figure 30 - Module TimeTableCatalog

Le module **TimeTableCatalog** (anciennement nommé module **TimeTable**) contient toutes les fonctionnalités déjà présentes pour les **TimeTable** (Cf. [R3]) auxquelles ont été ajoutées les fonctionnalités spécifiques aux catalogues.

Partant du fait qu'un Catalogue n'est rien d'autre qu'une **TimeTable** à laquelle on ajoute des paramètres pour chaque intervalle de temps, il nous a semblé logique de définir la classe **Catalog** héritant de **TimeTable** et proposant la seule fonction d'écriture des **ParameterDescription** (*addParameterDescription*). En effet, la liste des **ParameterDescription** est protégée dans la classe **TimeTable** et aucun « setter » n'est proposé pour cette classe, ce qui rend impossible la définition de **ParameterDescritption** au niveau d'une **TimeTable**.

Les données associées aux paramètres ajoutés au catalogue (*addParameterData*) sont positionnées dans la classe **TimeInterval** grâce à la méthode *addParameterData*. Ces données sont fournies sous la forme de 2 paramètres :

- Nom du paramètre ajouté
- La ou les valeurs associées au paramètre (sous forme d'une liste de string)

Le nombre de valeurs associées au paramètre ajouté doit correspondre à l'attribut size définit pour le **ParameterDescriptionj** associé.

Les classes d'écriture et lecture des **TimeTable** ou **Catalog** ont un diagramme de classe symétrique (voir ci-après).

Dans ces classes, rien ne distingue une time table d'un catalogue, les méthodes de lecture et d'écriture abstraites implémentées dans les classes filles n'acceptent d'ailleurs qu'une TimeTable (ou objet dérivé) en paramètre d'appel.

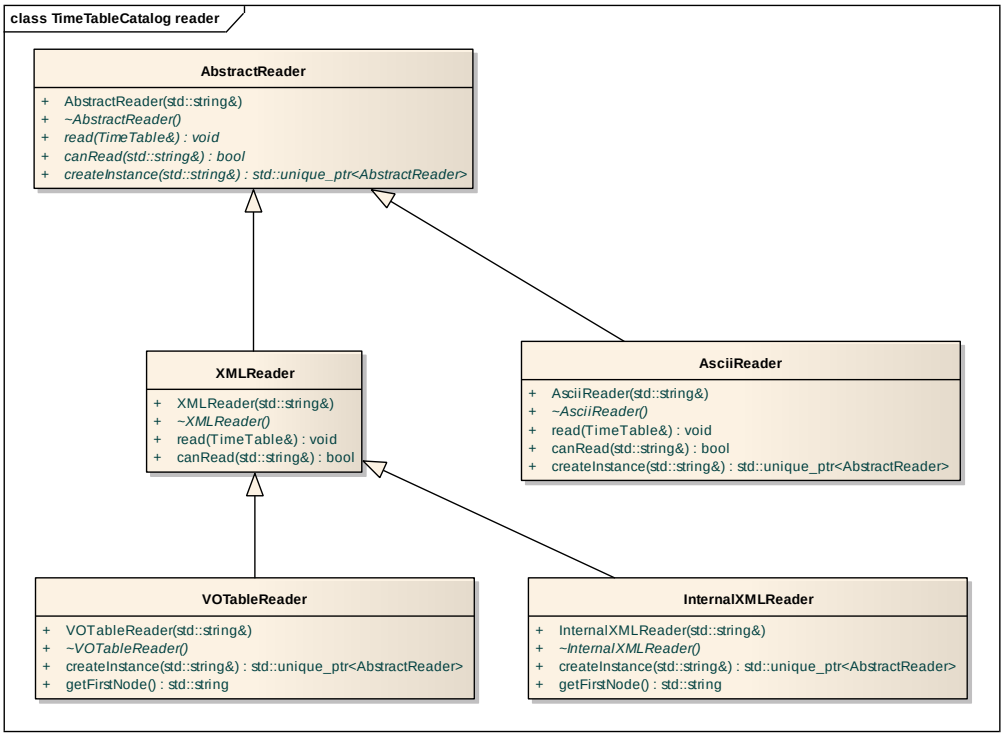


Figure 31 - Classes de lecture TimeTable et Catalog

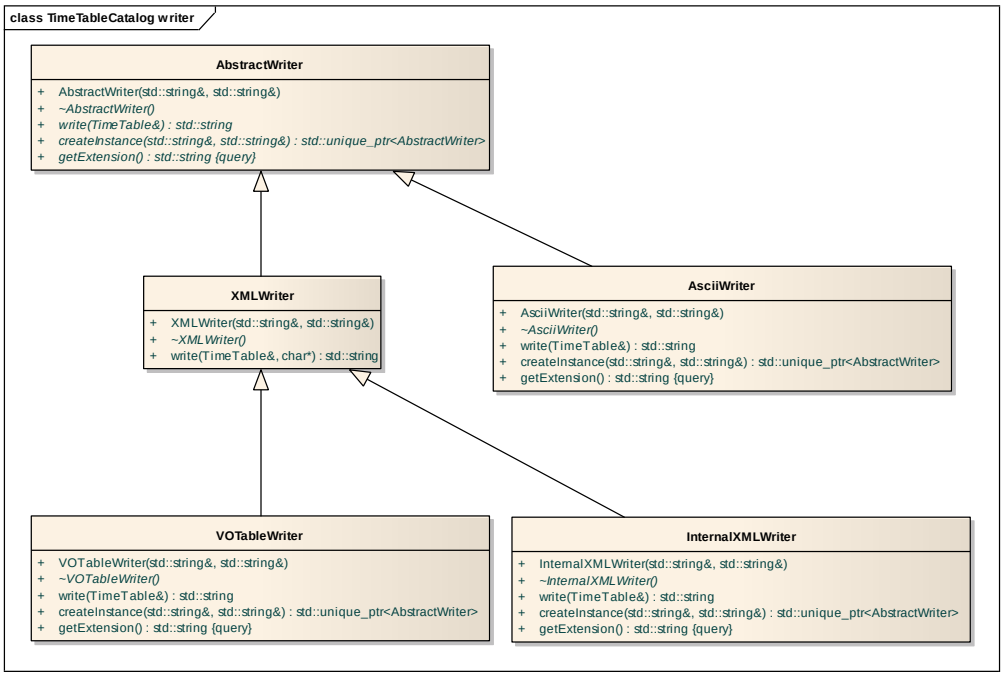


Figure 32 - Classes d'écriture TimeTable et Catalog

3.2.7 Module ParamGetLocalFile

3.2.7.1 Base de données des fichiers locaux

La structure de la base de données des fichiers locaux est contenue dans un fichier XML décrit dans [R4].

Dans ce fichier :

- Une liste de nœud « vi » est disponible constituant une liste de VirtualInstrument,
- Un VirtualInstrument contient une liste de fichiers contenant la même structure de définition des paramètres et définissant les données de ces paramètres pour des intervalles de temps.

Cette base est chargée et est rendue accessible à l'application via la classe « VirtualInstrumentBaseParser ».

3.2.7.2 Définition des paramètres locaux

Le fichier de définition d'un paramètre local est un fichier XML décrit dans [R4].

Un nœud « localvi » au niveau du nœud « get » du paramètre aura pour effet de solliciter la lecture de ce nœud via la classe « GetLocalFileNode » afin d'instancier un « ParamGetLocalFile » (cf. §3.2.7.3) et l'associer au « Parameter ».

3.2.7.3 Implémentation « ParamGetLocalFile » d'un « DataWriter »

La classe « ParamGetLocalFile » est une implémentation d'un « ParamGet », qui est lui-même un « DataWriter » (cf. [R2]), dont la responsabilité est de récupérer des données dans une base de données de fichiers locaux et de les écrire dans une instance d'un « ParamData ».

Cette classe utilise, à l'instar du module « ParamGetDDBase » (cf. [R2]) deux mécanismes fondamentaux :

- Le premier mécanisme est porté par la classe « ParamFlow » dont la responsabilité est de fournir un flot de données stockées dans une instance de « LocalParamDataPacket » au « ParamGetDDBase »,
- Le second mécanisme est porté par la classe « Pusher » dont la responsabilité est de, à partir d'une instance de « LocalParamDataPacket », remplir le « ParamData » du Parameter.

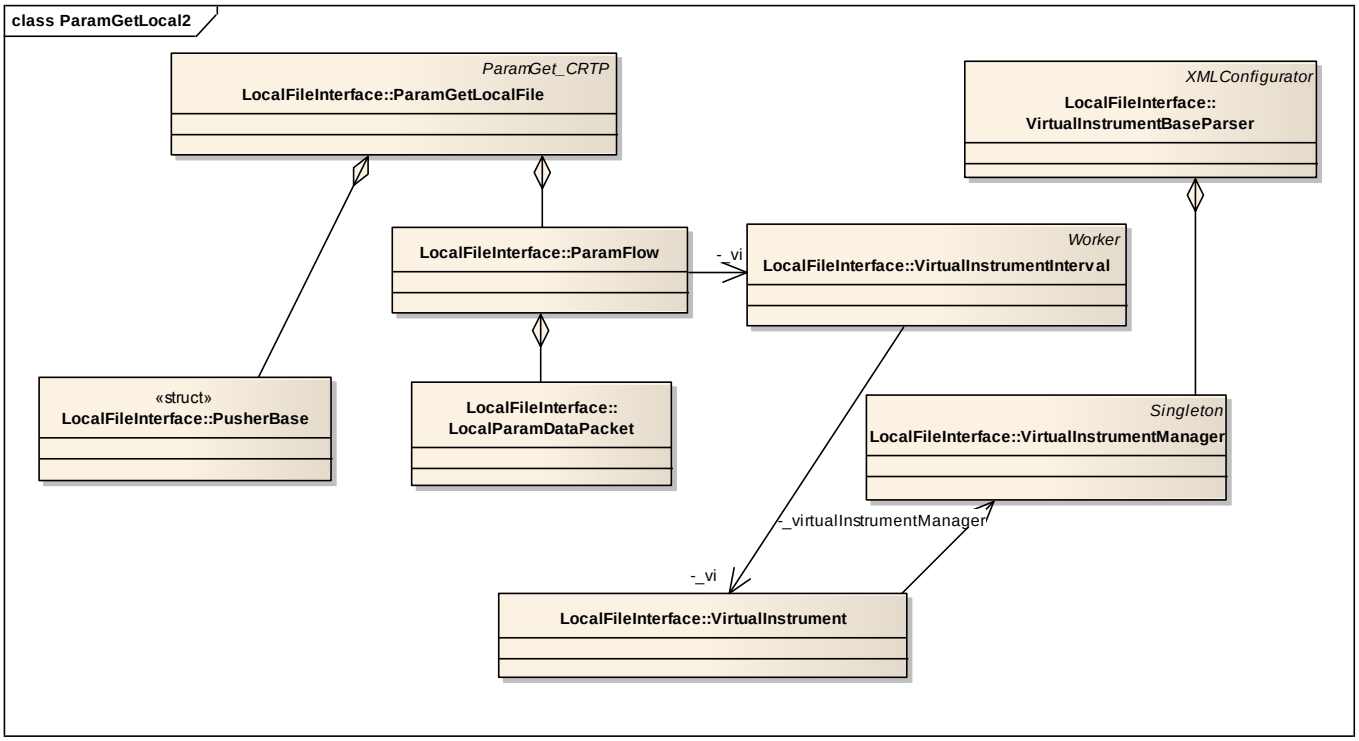


Figure 33 – Diagramme de classe simplifié du module ParamGetLocalFile

Une classe « VirtualInstrument » permet :

- D'instancier le « FileReader » en fonction du format des fichiers associés au VirtualInstrument (fonction « createFileReader »),
- D'instancier un « Pusher » en fonction du type et de la dimension du paramètre (fonction « createPusher »),
- De récupérer, et si nécessaire d'instancier, le « ParamFlow » associé à une liste d'intervalle de temps (fonction « getParamFlow »),

L'unicité des « VirtualInstrument » est assurée par le « VirtualInstrumentManager » qui utilise le « VirtualInstrumentBaseParser » afin de récupérer la définition du VirtualInstrument dans le fichier de définition de la base locale (cf. §3.2.7.1).

3.2.7.4 Implémentation des « FileReader »

La classe abstraite « FileReaderAbstract » définit l'interface de lecture des données des fichiers de la base locale.

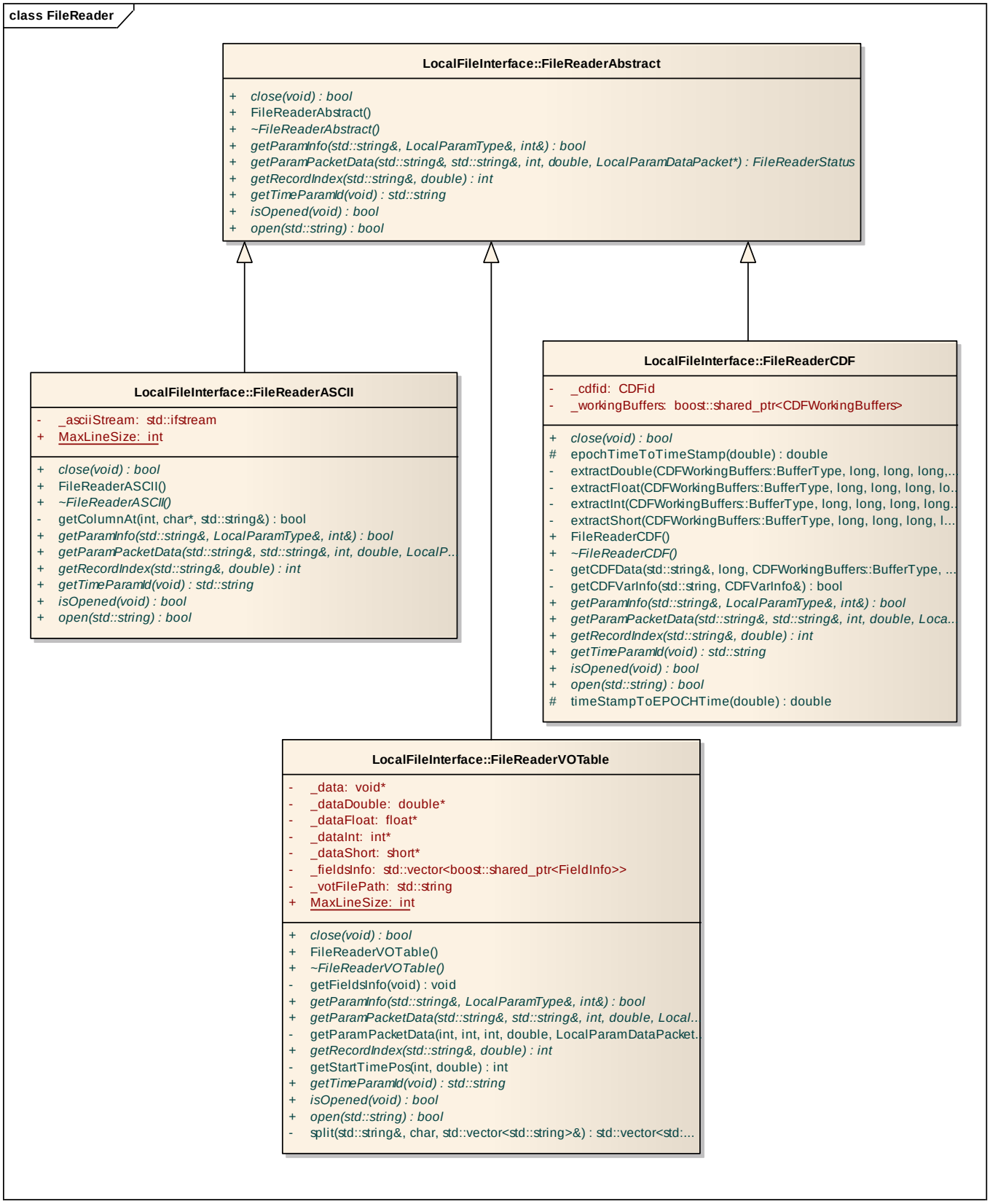


Figure 34 - Implémentation des FileReader

4 DOCUMENTS APPLICABLES ET DE REFERENCE (A/R)

A/R	Référence	Titre
[R1]	CDPP-AR-32500-382-SI	Dossier d'architecture du noyau d'AMDA-NG
[R2]	CDPP-CD-32500-436-SI	Dossier de conception du noyau d'AMDA-NG
[R3]	CDPP-CD-32500-457-CS	Dossier de conception – Noyau AMDA-NG – 2nde partie
[R4]	CDPP-IF-32500-504-SI	Dossier de contrôle des interfaces du noyau AMDA-NG (3ème partie)

5 GLOSSAIRE ET ABREVIATIONS

5.1 GLOSSAIRE

Terme	Définition

5.2 ABREVIATIONS

Abréviation	Nom détaillé
AMDA	Automated Multiple Dataset Analysis
COTS	Commercial off-the-shelf
GNU	GNU Compiler Collection
IRAP	Institut de Recherche en Astrophysique et Planétologie
XML	eXtended Markup Language
XSD	XML Schema Definition