NTIC

Projet : Evolutions et intégration AMDA-NG

NOTES TECHNIQUES POUR LA PRESTATION 'EVOLUTIONS ET INTEGRATION AMDA-NG'

Rédigé par : Benjamin Renard	Diffusé à : CDPP / 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
01.0	07/10/15		Création du document

SOMMAIRE

1 INTRODUCTION

La prestation « Evolutions et intégration AMDA-NG » s'est déroulée du lundi 20 juillet 2015 au vendredi 9 octobre 2015 (50 jours de charge, pour un ETP AKKA) dans les locaux de l'IRAP.

Cette prestation concernait un ensemble de tâches diverses, toutes liées au projet AMDA du CDPP / IRAP, référencées dans la proposition commerciale [R0]. Il est à noter que des tâches supplémentaires ont été traitées au cours de la prestation, alors que d'autres ont été modifiées, en accord avec l'équipe technique du CDPP / IRAP.

Ce document constitue un ensemble de notes techniques couvrant l'ensemble des activités réalisées lors de cette prestation.

2 INTEGRATION DU NOUVEAU MODULE « PLOT » DANS AMDA

L'implémentation du nouveau noyau AMDA a amené un ensemble de nouvelles fonctionnalités devant être rendues accessibles aux utilisateurs depuis « AMDA_IHM ». Ainsi, une refonte complète du module « Plot » a été réalisée.

Le module « AMDA_Integration » a également évolué afin de prendre en compte les modifications réalisées au niveau de la définition d'une requête de « Plot ».

2.1 CONCEPTION GENERALE DU MODULE « PLOT » DANS LE PROJET « AMDA_IHM »

La nouvelle implémentation du module « Plot » dans le projet « AMDA_IHM » s'intègre dans la conception existante du projet. Un nettoyage de l'ancien module « Plot » a également été effectué afin de supprimer le code devenu obsolète.

La conception du projet « AMDA_IHM » se base sur une architecture de type « Modèle-vue-contrôleur », nous retrouvons donc ces trois types d'objets dans la conception du module « Plot ».

2.1.1 Les « modèles » du module « Plot »

Un « modèle » représente un modèle de données. Dans le cas du module « Plot », nous retrouvons :

- Les données de définition d'une requête de « Plot », contenant également des fonctionnalités de traitement de ces données (lecture/écriture au format JSON pour communication AJAX avec le serveur, test de modification de la requête, initialisation avec les valeurs par défaut, etc.). Ces données sont portées par la classe principale « PlotRequestObject » et par un ensemble de classes filles par le biais d'associations. Toutes ces classes sont regroupées dans « js/app/models/PlotObjects/ ».
- Les données de définition d'un nœud « Requête de Plot », portées par la classe « PlotNode » héritant d'un « ExecutableNode » et représentant une requête enregistrée par l'utilisateur dans son espace. Ce « nœud » fait un lien vers un objet de type « PlotRequestObject » contenant l'ensemble des données de définition de la requête. Ces nœuds sont utilisés pour l'affichage de l'arbre des requêtes sauvegardées par un utilisateur dans le module « Explorer » d'AMDA. Lors de la définition d'une nouvelle requête de « Plot », un nœud temporaire de ce type est créé. Le nœud en cours d'édition par un utilisateur est nommé « linkedNode ».
- Les données de définition d'un nœud « Elément de Requête », portées par la classe abstraite « PlotTreeNode ». Les différentes implémentations de ce type de nœud représentent chacune un type d'élément de définition d'une requête. Ces nœuds sont utilisés pour un affichage d'une requête sous forme d'un arbre hiérarchique dans le module « Plot ».

Il est à noter que l'ensemble des valeurs par défaut pour les données de définition d'une requête de « Plot » ont été regroupées dans la classe unique « PlotObjectConfig ».

2.1.2 Les « vues » du module « Plot »

Une « vue » représente une IHM sur laquelle l'utilisateur va interagir. Dans le cas du module « Plot », nous retrouvons :

- La vue « PlotUI », représentant la fenêtre principale de définition d'un « Plot » par l'utilisateur. Cette fenêtre utilise un ensemble de « sous-composants » définis dans « js/app/views/PlotComponents/ ».
- La vue « PlotTabResultUI », représentant une fenêtre dédiées à l'affichage d'une page résultante de l'exécution d'une requête de type « Plot » dans le mode interactif.
- La vue « PlotPreviewUI », représentant une fenêtre dédiée à l'affichage d'une page résultante d'une action effectuée depuis une vue de « PlotTabResultUI » dans le mode interactif.

2.1.3 Le « contrôleur » du module « Plot »

Le contrôleur du module « Plot » est porté par la classe « PlotModule » héritant de la classe « InteractiveModule ».

Ses responsabilités se limitent à :

- Construire et afficher la fenêtre de définition d'une requête de « Plot » (correspondant à la vue « PlotUI »),
- Faire le lien avec l'objet de définition d'un nœud « Requête de Plot », nommé « linkedNode », actuellement édité par l'utilisateur dans la fenêtre de définition d'une requête de « Plot »,
- Dans le cas du mode interactif, construire, afficher et mettre à jour les différentes fenêtres dédiées à l'affichage des résultats (correspondant aux vues « PlotTabResultUI » et « PlotPreviewUI »).

2.2 PRESENTATION GENERALE DE L'IHM DE DEFINITION D'UNE REQUETE DE TYPE « PLOT »

La fenêtre de définition d'une requête de type « Plot » est définie par la vue « PlotUI ». Cette fenêtre est décomposée de la manière suivante :

- Une zone dans laquelle une requête est représentée de manière hiérarchique, sous forme d'un arbre. Deux types de représentations de cet arbre sont disponibles et accessibles à partir du checkbox « Simplified View » (dans le mode simplifié, l'affichage est similaire à celui de l'ancienne version du module « Plot », en revanche toutes les options de tracé ne sont pas accessibles). Cette zone contient également la définition des « tabs », c'est-à-dire l'ensemble des « sous-requêtes » produisant une page et constituants la requête de « Plot » (utilisé notamment dans le cas du « Multi-Plot »). Les paramètres peuvent être « glissés-déposés » dans cette zone afin de définir les paramètres à afficher.
- Une zone dédiée à l'affichage et à l'édition des options disponibles pour l'élément sélectionné dans l'arbre hiérarchique.
- Une zone dédiée à l'affichage des options de sortie de la requête, s'appliquant à l'ensemble des « tabs » constituant la requête de « Plot ».
- Une zone de définition de l'intervalle de temps. Deux cas de figure se présentent :

- Si un « tab » lié au mode « Multi-Plot » est sélectionné, la zone de sélection de l'intervalle de temps est commune à tous les « tabs » liés à ce mode.
- Si un « tab » non lié au mode « Multi-Plot » est sélectionné, la zone de sélection de l'intervalle de temps ne s'applique qu'à ce « tab ».

- Une zone contenant les boutons des actions associées à la requête.

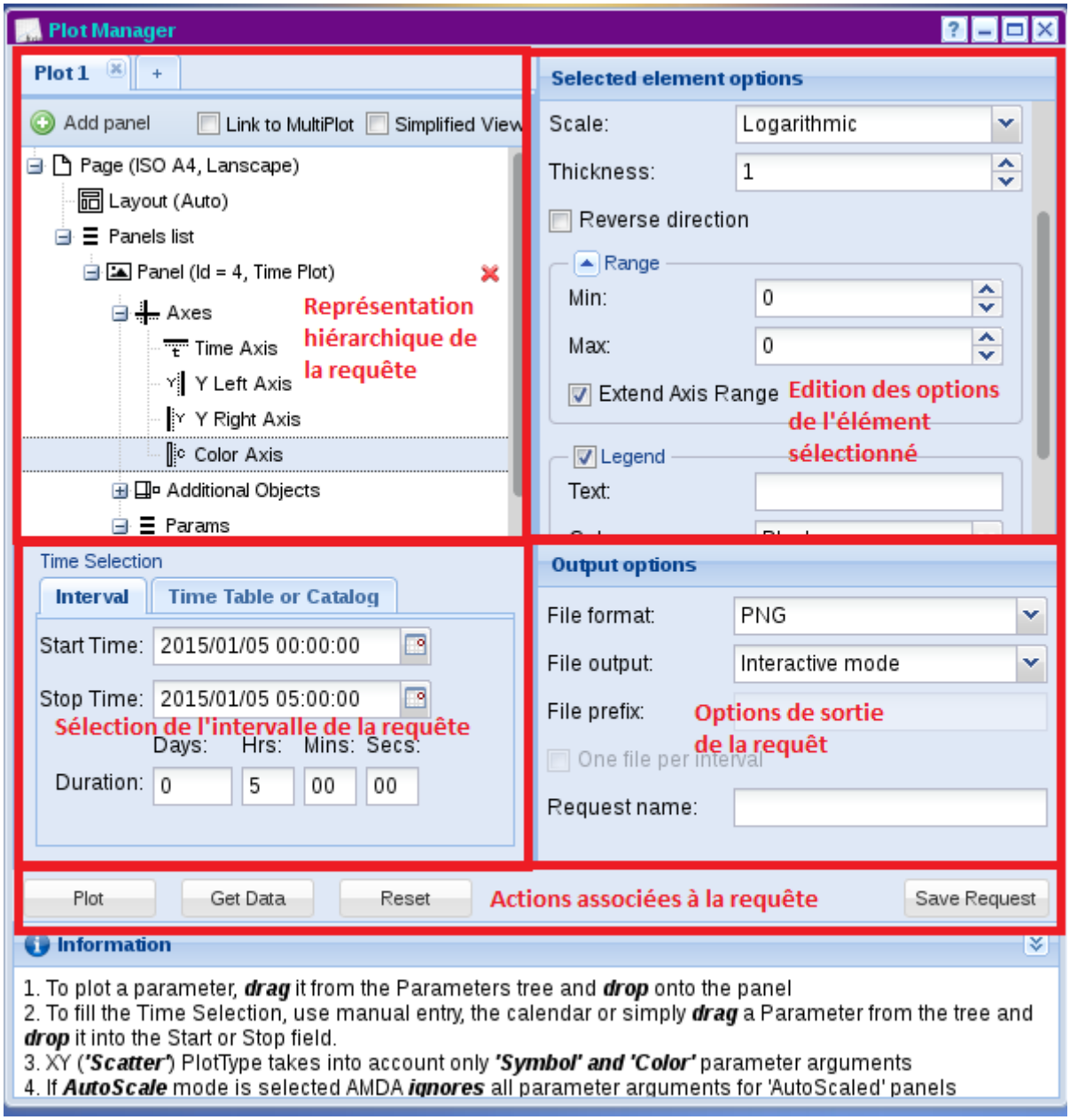


Figure 1 - Zones de la vue "PlotUI"

2.3 PRESENTATION DU MODE INTERACTIF DU MODULE « PLOT »

Après l'exécution d'une requête de « Plot », le mode interactif est automatiquement activé lorsque les options de sortie sont à :

- « PNG » pour le « File format »,
- « Interactive mode » pour le « File output ».

Dans ce mode, le résultat de chaque « tab » constituant la requête est présenté dans une vue de type « PlotTabResultUI ».

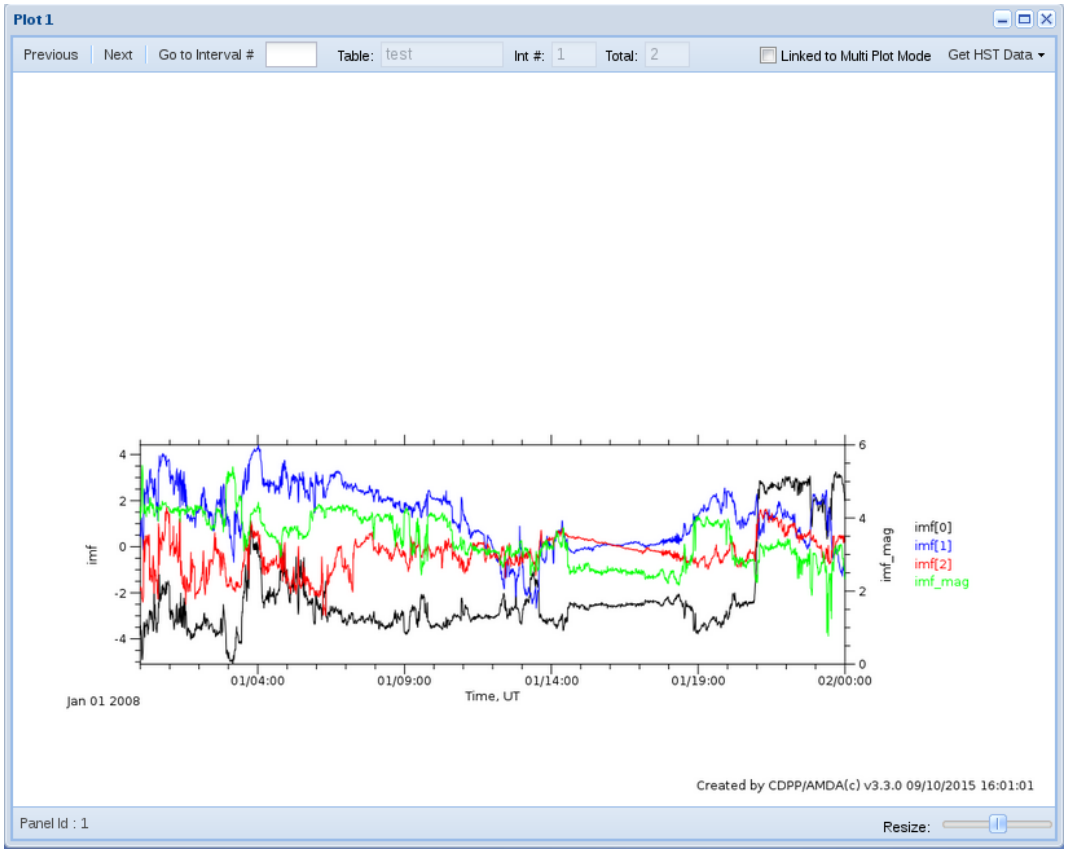


Figure 2 - Aperçu d'une vue "PlotTabResultUI"

2.3.1 Informations de contexte d'une requête de type « Plot »

A une page tracée lors d'une requête de type « Plot » correspond un ensemble de paramètres décrivant le tracé réalisé. Nous nommerons ces paramètres de tracé « Informations de contexte » dans la suite de ce document.

Ces « Informations de contexte » regroupent notamment la dimension de la page, son orientation, la position des différents « panels », les axes, les échelles etc.

Dans le module « AMDA_Kernel », l'attribut « writeContextFile » a été ajouté au niveau de la définition du nœud « plot » dans le schéma XML « plot.xsd » de définition d'une requête de « Plot ».

Lorsque cet attribut prend la valeur « true », un fichier « XML », nommé « PREFIX_context.xml », contenant les informations de contexte est produit en parallèle du tracé de la page.

Ce fichier est ensuite « parsé » dans le module « AMDA_Integration » (cf. « src/InputOutput/IHMImpl/Tools/IHMPlotContextFileClass.php ») et son contenu relayé jusqu'à la vue de type « PlotTabResultUI » du module « AMDA_IHM ».

Ces informations de contexte sont ensuite utilisées par la vue afin de redimensionner la fenêtre, d'afficher les coordonnées du point survolé par la souris de l'utilisateur, etc. en fonction du tracé réalisé.

2.3.2 Affichage des coordonnées du point survolé par la souris de l'utilisateur

Lorsque un utilisateur survole un point contenu dans une « plot area » d'un panel, les coordonnées réelles de ce point sont affichées dans la « status bar » de la vue « PlotTabResultUI ».

Le type d'affichage d'une coordonnée est désigné par le type d'axe concerné.

2.3.3 « Zoom », « Extend » et « Shift »

Les actions de « Zoom », « Extend » et « Shift » sont accessibles depuis le menu contextuel associé au clic droit de la souris, lorsque l'utilisateur survole un panel.

En fonction du panel survolé, une action de zoom est proposée pour chaque axe disponible.

2.3.4 Sauvegarde directe d'une image d'un « Plot »

L'action de sauvegarde directe d'une image d'un « Plot » est accessible depuis le menu contextuel associé au clic droit de la souris, lorsque l'utilisateur survole la page.

2.3.5 Prise en charge de l'«instant cut » interactif

L'action d'«instant cut » interactif est accessible depuis le menu contextuel associé au clic droit de la souris, lorsque l'utilisateur survole un panel contenant un « spectro ».

Cette action produit un plot de type « instantPlot », avec pour temps de coupe la valeur survolée au moment de l'ouverture du menu contextuel. Le résultat de ce « Plot » s'affiche dans une vue de type « PlotPreviewUI ».

2.3.6 Mode « Multi-Plot »

Le mode « Multi-Plot » permet de synchroniser le paramètre temps entre plusieurs « tabs » d'une même requête de type « Plot ».

Un « tab » est initialement lié au mode « Multi-Plot » lorsque le checkbox « Link to MultiPlot » est coché dans l'interface de définition de la requête.

L'utilisateur peut néanmoins « lier » / « délier » momentanément ses « tabs » dans le mode interactif, depuis les différentes vues « PlotTabResultUI » du plot en cours, en cochant / décochant le checkbox « Linked to Multi Plot Mode » :

- Lorsque l'utilisateur décoche ce checkbox, la fenêtre résultat courante se retrouve détachée du mode « Multi-Plot ». Une navigation sur cette fenêtre n'aura aucune influence sur les autres fenêtres résultats.
- En revanche, s'il coche ce checkbox, la fenêtre résultat courante se rattachera au mode « Multi-Plot » et se synchronisera avec le temps courant lié au mode « MultiPlot ».

2.4 MISE A JOUR DE L'INTEGRATION DU MODULE « PLOT » DANS LE PROJET « AMDA_INTEGRATION »

Le module « AMDA_Integration » a évolué afin de prendre en considération les modifications dans la définition d'une requête de « Plot » au niveau du module « AMDA_IHM », et prendre en charge l'ensemble des options de tracé disponibles.

En revanche, sa conception n'a pas été modifiée (cf. [R2]).

Les modifications se sont concentrées sur :

- La classe d'interface pour le module Plot « IHMInputOutputParamsPlotClass », qui a été entièrement réécrite
- L'interface unifiée contenue dans «src\Request\ParamsRequestImpl\Nodes\Requests\ », pour laquelle toutes les options de tracé ont été ajoutées.

3 LIBRAIRIE « DDCLIENT », ET MODIFICATION DE LA PROCEDURE D'INSTALLATION DU MODULE « AMDA_KERNEL »

Le projet « AMDA_Kernel » contenait les sources C et C++ de la librairie « DDClient », ainsi que celles destinées à la compilation des exécutables « DDLogin », « HtmlLogin » et « Check_User ».

Ces sources étaient également contenues dans le projet « DDServer ».

Enfin, une ancienne version de ces sources était présente dans un repository CVS « AMDAcvs ».

Il était donc devenu indispensable de créer un projet propre à la librairie « DDClient » et d'y merger les sources de ces différentes implémentations.

3.1 ORGANISATION DU REPOSITORY GIT DU PROJET « DDCLIENT »

Le repository GIT « DDClient » a été créé afin de recevoir les sources de « DDClient » (https://gitlab.irap.omp.eu/CDPP/DDClient.git).

Ce projet contient :

- Les sources de la librairie en C (cf. « src/DDClientLibC »).
- Les sources de la librairie en C++ (cf. « src/DDClientLibCpp »)
- Les sources de plusieurs exécutables, utilisant la librairie DDClient, et pouvant être utilisé par plusieurs projets (cf. « src/TOOLS »).

3.2 COMPILATION ET INSTALLATION DE DDCLIENT

La compilation et l'installation de DDClient s'effectue en utilisant cmake.

Les commandes à effectuer sont décrites dans le fichier « README » du projet.

3.3 MODIFICATION DE LA PROCEDURE D'INSTALLATION DU PROJET « AMDA_KERNEL »

Dorénavant, les sources de « DDClient » ont été supprimées du projet « AMDA_Kernel ».

Ainsi, l'installation de « DDClient » devient un prérequis à l'installation du projet « AMDA_Kernel » (cf. [R1]).

Afin d'assurer la compatibilité avec l'exécution des différents scripts de tests, et d'éviter à avoir à modifier les jobs jenkins du projet, nous imposons son installation dans le répertoire suivant : « /opt/local ».

Les commandes à exécuter pour sa compilation et son installation deviennent donc :

- cmake -E make_directory build
- cmake -E chdir build cmake -DCMAKE_BUILD_TYPE=Release -DCMAKE_INSTALL_PREFIX=/opt/local ..
- make -C build install VERBOSE=1

4 OPTIMISATION DE LA GESTION DE L'ACCES CONCURRENTIEL AU NIVEAU DU PROJET « DDSERVER »

4.1 PRESENTATION RAPIDE DU MECANISME DE CACHE DE DDSERVER

Les fichiers de données accédés par DDServer sont compressés au format « gzip » sur le disque.

Au fur et à mesure du traitement d'une requête de récupération de données pour un « Virtual Instrument » (noté VI_ID), « DDServer » décompresse successivement les fichiers de données qui doivent être accédés.

Les fichiers de donnés décompressés sont enregistrés auprès du fichier « VI_ID_cache.nc ».

Le nombre maximum de fichiers pouvant être décompressé à un instant donné est fixé par la dimension (actuellement fixée à 20) partagée par les paramètres « name » (destiné à recevoir les noms des fichiers décompressés) et « time » (destiné à recevoir le « timestamp » désignant la date de décompression du fichier) du fichier de cache.

Lorsque le nombre maximum de fichier décompressé pour VI_ID est atteint, DDServer détermine le plus ancien fichier décompressé à partir du paramètre « time » du fichier de cache, le compresse et utilise l'emplacement ainsi libéré pour décompresser le nouveau fichier de données.

4.2 PRESENTATION DE LA PROBLEMATIQUE

« DDServer » est en mesure de gérer le traitement simultané de plusieurs requêtes provenant de différents clients.

La connexion d'un client à un « Virtual Instrument » de « DDServer » (noté VI_ID) provoque la création, d'une part, d'un socket qui sera utilisé pour la communication entre le serveur et le client et, d'autre part, d'un processus « forké » de DDServer destiné à traiter les différentes requêtes reçues de la part du client (cf. fonction « Serv » de « DD_Server.c »).

Imaginons la situation suivante :

- Deux clients accèdent simultanément à VI_ID,
- Le nombre maximum de fichier décompressé est atteint,
- Le premier client est en train de lire les données du fichier dont la décompression est la plus ancienne,
- Le second client a besoin d'accéder à un fichier qui n'est pas décompressé.

Dans cette situation, le second client va commander la compression du fichier dont la décompression est la plus ancienne, c'est-à-dire le fichier qui est actuellement lu par le premier client, et ainsi donc provoquer une exception.

Cette situation permet d'illustrer la problématique, mais il est à noter que des situations différentes peuvent également provoquer l'échec de la récupération des données par un client, notamment dû au fait qu'un très léger décalage peut exister entre l'état compressé/décompressé sur le disque et l'état indiqué dans le fichier de cache.

En effet, DDServer réalise d'abord la compression/décompression sur le disque et ne met qu'ensuite à jour l'état dans le fichier de cache.

La probabilité d'apparition de ce type de situations est d'autant plus élevée que le nombre de clients interrogeant le même Virtual Instrument simultanément est important, ce qui peut être le cas lors de l'organisation d'un atelier par les scientifiques de l'IRAP, par exemple.

4.3 DESCRIPTION DU MECANISME DE GESTION DE L'ACCES CONCURRENTIEL MIS EN PLACE

Le mécanisme de gestion de l'accès concurrentiel mis en place (cf. « DD_Cache.c ») s'appuie sur l'utilisation d'une instance d'une structure de type « CacheData » placée dans un espace mémoire partagé par l'ensemble des processus « forkés » de DDServer.

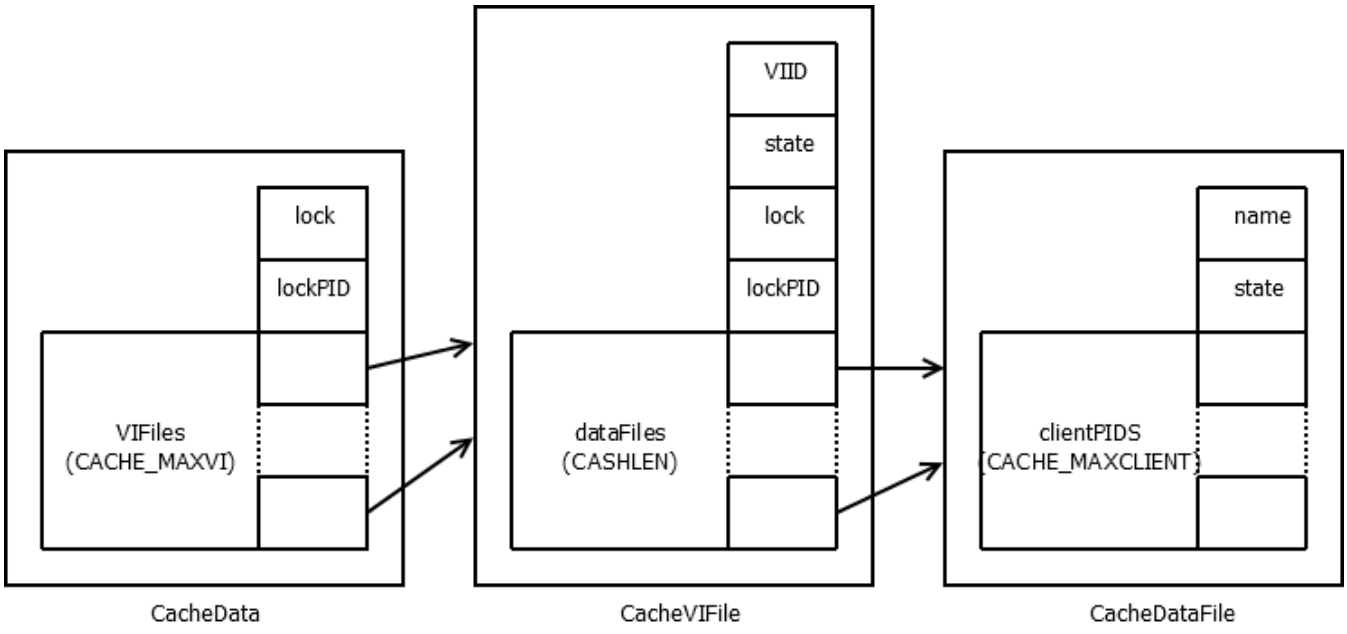


Figure 3 - Schématisation de la structure "CacheData"

La structure « CacheData » contient :

- Un mutex « lock » permettant de réserver l'accès à l'instance de la structure à un seul client à la fois,
- Un entier « lockPID » contenant le « Process ID » du processus responsable du lock de l'instance de la structure. Cette indication permet de s'assurer que le processus est toujours en vie et, le cas échéant, permettre de forcer le déverrouillage du mutex (cette situation peut se produire lorsqu'un processus est sorti en exception, ce qui ne devrait normalement jamais arriver).
- CACHE_MAXVI éléments de type « CacheVIFile » (cf. ci-dessous).

Une instance d'une structure de type « CacheVIFile » contient :

- Une chaine de caractères « VIID » contenant l'identifiant du Virtual Instrument concerné.
- Un entier « state » précisant l'état de l'instance (« 0 » pour non utilisé pour un Virtual Instrument, et « 1 » pour utilisé).
- Un mutex « lock » permettant de réserver l'accès à l'instance de la structure à un seul client à la fois,
- Un entier « lockPID » contenant le « Process ID » du processus responsable du lock de l'instance de la structure. Cette indication permet de s'assurer que le processus est toujours en vie et, le cas échéant, permettre de forcer le déverrouillage du mutex (cette situation peut se produire lorsqu'un processus est sorti en exception, ce qui ne devrait normalement jamais arriver).
- CASHLEN éléments de type « CacheDataFile » (cf. ci-dessous).

Une instance d'une structure de type « CacheDataFile » contient :

- Une chaine de caractère « name » précisant le nom de fichier de données concerné.
- Un entier « state » précisant l'état de l'instance (« 0 » pour non utilisé pour un fichier de données, et « 1 » pour utilisé).
- CACHE_MAXCLIENT éléments de type entier précisant la liste des « Process ID » des processus qui sont en cours de lecture du fichier de données concerné.

Le mécanisme implémenté utilise les quatre fonctions principales suivantes :

- « Cache_Init » : cette fonction est appelée par le processus parent, afin d'allouer et d'initialiser l'instance de type « CacheData » dans un espace mémoire partagé,
- « Cache_Free » : cette fonction est appelée par le processus parent, afin de libérer l'instance de type « CacheData »,
- « Cache_RequestDataFileAccess » : cette fonction est appelée par les processus enfants afin de réclamer l'accès à un fichier de données (cf. séquence ci-dessous). Après l'appel à cette fonction, le processus est assuré que le fichier est décompressé et qu'il le restera.
- « Cache_ReleaseDataFileAccess » : cette fonction est appelée par les processus enfants afin de préciser qu'il n'utilise plus un fichier de données (cf. séquence ci-dessous).

La séquence mise en œuvre lors de l'appel à la fonction « Cache_RequestDataFileAccess » peut être résumée de la manière suivante :

- Verrouillage de l'accès à l'instance de type « CacheData »,

- Parcours de la liste des éléments de type « CacheVIFile » pour trouver celui qui correspond au Virtual Instrument concerné :
    - S'il est trouvé, on verrouille l'accès à l'instance de l'élément de type « CacheVIFile » correspondant et on déverrouille l'accès à l'instance de type « CacheData »,
    - Sinon, on cherche un élément de type « CacheVIFile » non utilisé :
        - Si on en trouve un, on le réserve pour ce Virtual Instrument, on verrouille l'accès à l'instance de l'élément de type « CacheVIFile » correspondant et on déverrouille l'accès à l'instance de type « CacheData »,
        - Sinon, on procède à un nettoyage de la structure en libérant les éléments de type « CacheVIFile » qui ne sont plus utilisés par aucun processus, et relançons la procédure de recherche d'un élément de type « CacheVIFile » pour ce Virtual Instrument.
- A ce niveau, nous sommes assuré que l'instance de l'élément de type « CacheVIFile » correspondant au Virtual Instrument est verrouillé par notre processus,
- Synchronisation de la structure « DD_cash_t » avec le contenu du fichier cache du Virtual Instrument,
- Parcours de la liste des éléments de type « CacheDataFile » pour trouver celui correspondant au fichier de données concerné :
    - S'il est trouvé, on ajoute le « Process ID » dans la liste des processus accédant à ce fichier,
    - Sinon, on cherche le plus ancien fichier de données qui n'est plus accédé par aucun processus (à noter que les emplacements vides portent le « timestamp » 0. Ces emplacements sont donc forcément prioritaires par rapport aux emplacements déjà utilisés par des fichiers de données) :
        - Si on en trouve un, on compresse l'ancien fichier de données utilisé par cet élément (s'il était utilisé), on décompresse notre nouveau fichier de données, on ajoute le « Process ID » dans la liste des processus accédant à ce fichier,
        - Sinon, on nettoie la structure en supprimant les processus morts et on appelle de nouveau la fonction « Cache_RequestDataFileAccess »,
- A ce niveau-là, on est assuré que le fichier de données est décompressé et que notre « Process ID » a été ajouté dans la liste des processus accédant à ce fichier. Par ailleurs, la structure « DD_cash_t » est mise à jour pour refléter la nouvelle situation du cache du Virtual instrument,
- Synchronisation du fichier cache avec le contenu de la structure « DD_cash_t »,
- Libération du verrou de l'instance de type « CacheVIFile » correspondant au Virtual Instrument.

La séquence mise en œuvre lors de l'appel à la fonction « Cache_ ReleaseDataFileAccess » peut être résumée de la manière suivante :

- Verrouillage de l'accès à l'instance de type « CacheData »,

- Parcours de la liste des éléments de type « CacheVIFile » pour trouver celui qui correspond au Virtual Instrument concerné :
    - S'il est trouvé, on verrouille l'accès à l'instance de l'élément de type « CacheVIFile » correspondant et on déverrouille l'accès à l'instance de type « CacheData »,
    - Sinon, on cherche un élément de type « CacheVIFile » non utilisé :
        - Si on en trouve un, on le réserve pour ce Virtual Instrument, on verrouille l'accès à l'instance de l'élément de type « CacheVIFile » correspondant et on déverrouille l'accès à l'instance de type « CacheData »,
        - Sinon, on procède à un nettoyage de la structure en libérant les éléments de type « CacheVIFile » qui ne sont plus utilisés par aucun processus, et relançons la procédure de recherche d'un élément de type « CacheVIFile » pour ce Virtual Instrument.
- A ce niveau, nous sommes assurés que l'instance de l'élément de type « CacheVIFile » correspondant au Virtual Instrument est verrouillé par notre processus,
- Synchronisation de la structure « DD_cash_t » avec le contenu du fichier cache du Virtual Instrument,
- Parcours de la liste des éléments de type « CacheDataFile » pour trouver celui correspondant au fichier de données concerné
    - S'il est trouvé, parcours de la liste des processus pour supprimer notre « Process ID »,
    - Sinon, il n'y a rien à faire,
- Libération du verrou de l'instance de type « CacheVIFile » correspondant au Virtual Instrument.

5 MISE A JOUR DU WEBSERVICE AMDA

Les fonctionnalités du WebService AMDA sont portées par la classe « WebServer » du module « AMDA_IHM ».

Le WebService n'était plus fonctionnel puisque les fonctions « getPlot », « getParameter », « getOrbites » et « getDataset » utilisées encore l'implémentation de l'ancien noyau AMDA.

Cette tache consistait donc à modifier ces fonctions pour qu'elles utilisent le nouveau noyau.

5.1 PRISE EN CHARGE DU NOUVEAU NOYAU

Les fonctions « getParameter », « getOrbites » et « getDataset » du WebService réalisent toutes les trois un appel à la fonction « doDownloadRequest » afin de réaliser une requête de type « Download ».

La fonction « getPlot » réalise de son côté une requête de type « Plot ».

Ainsi, seules les fonctions « getPlot » et « doDownloadRequest » ont été modifiées.

Les modifications ont consisté à construire les objets PHP similaires à ceux reçus par « AmdaAction » pour le traitement de ces deux types de requêtes, et de réaliser ensuite un appel de la fonction « runIHMRequest » du module « AMDA_Integration ».

5.2 IMPLEMENTATION D'UNE CLASSE CLIENTE ET D'UN SCRIPT DE TEST

Afin de mettre en place une procédure de test, une implémentation d'une classe cliente « WSClientSOAP » a été réalisée.

Elle est utilisée par le script « testWebServer.php » qui exécute une succession d'appel au WebService afin de le tester.

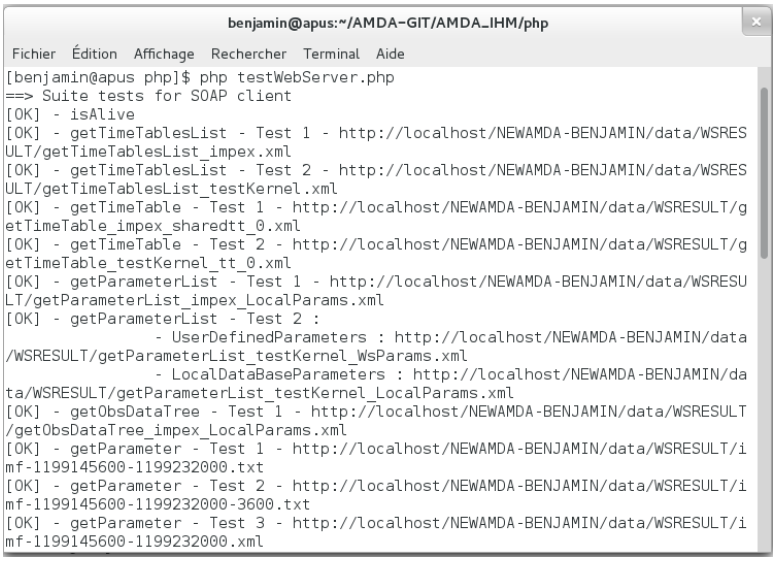


Figure 4 - Aperçu de l'exécution du script de test du WebService

6 PRISE EN COMPTE DES TABLES DANS LE MODULE « AMDA_KERNEL »

6.1 DEFINITION D'UN PARAMETRE AVEC TABLES VARIABLES

L'attribut « variable » a été ajouté au niveau du groupe d'attributs « TableGroup ».

Ce groupe est commun à tous les éléments de type « TableDef » qui sont utilisés pour définir une table.

Lorsque l'attribut « variable » n'est pas défini, ou lorsqu'il prend la valeur « false », les paramètres utilisés pour la définition d'une table sont des constantes et sont lus à partir d'une information de type « clb » (information de calibration lié au « ParamGet ») ou de type « clbManual » (information de calibration décrite dans le fichier de définition du paramètre).

En revanche, lorsque l'attribut « variable » prend la valeur « true », les paramètres utilisés pour la définition d'une table désignent des paramètres définis dans la base de données des paramètres du noyau.

6.1.1 Exemple de définition d'un paramètre avec table non variable

Le paramètre « e_mgs_omni1 » est un exemple de définition d'un paramètre avec table non variable contenu dans la base de test :

```
<?xml version="1.0" encoding="UTF-8"?>
<param xml:id="e_mgs_omni1">
 <info>
     <name>MGS electron flux</name>
     <short_name>MGS e-Flux</short_name>
     <components></components>
     <units>1/(cm^2*s*ster*eV)</units>
     <coordinates_system></coordinates_system>
     <tensor_order>0</tensor_order>
     <si_conversion></si_conversion>
     <table>
             <boundsTable name="Energy" boundsName="energy" units="eV"/>
     </table>
     <fill_value>-1.0e+31</fill_value>
     <ucd></ucd>
     <dataset_id></dataset_id>
 </info>
 <get>
 <vi name="mgs:er:omni">
     <baseParam name="e_flux">
```

```
                <clb name="energy"/>
        </baseParam>
    </vi>
    </get>
    <process/>
    <output/>
</param>
```

La table est de type « boundsTable » et ne contient pas l'attribut « variable ». Le paramètre de définition de la table désigné par l'attribut « boundsName » est donc une information de calibration.

L'information de calibration « energy » correspond à une information liée au « paramGet » définit par le nœud « clb ».

6.1.2 Exemple de définition d'un paramètre avec table variable

Le paramètre « ros_mip_surv » est un exemple de définition d'un paramètre avec table variable contenu dans la base de test :

```
<?xml version="1.0"?>
<param xml:id="ros_mip_surv">
  <info>
      <name>Rosetta MIP Survey</name>
      <short_name>MIP Survey</short_name>
      <components></components>
      <units>dB</units>
      <coordinates_system></coordinates_system>
      <tensor_order>0</tensor_order>
      <si_conversion></si_conversion>
      <table>
                <centerAutoTable variable="true" name="Frequency" units="kHz" centerName="ros_mip_surv_frequency"
log="true"/>
      </table>
      <fill_value>-1e+31</fill_value>
      <ucd></ucd>
      <dataset_id></dataset_id>
  </info>
  <get>
    <vi name='ros:mip:surv'>
      <baseParam name="SurveyFull"/>
```

```
    </vi>
    <amdaParam name="ros_mip_surv_frequency"/>
  </get>
  <process/>
  <output/>
</param>
```

La table est de type « centerAutoTable » et contient l'attribut « variable ». Le paramètre de définition de la table désigné par l'attribut « centerName » est donc un paramètre contenu dans la base de données des paramètres du noyau.

Il est à noter que le paramètre « ros_mip_surv_frequency » est défini dans le nœud « get » afin de s'assurer qu'il soit automatiquement initialisé lors de l'initialisation du paramètre « ros_mip_surv ».

La définition du paramètre de fréquence « ros_mip_surv_frequency » est la suivante :

```
<?xml version="1.0"?>
<param xml:id="ros_mip_surv_frequency">
  <get>
    <vi name='ros:mip:surv'>
      <baseParam name="Frequency_table_idx" useNearestValue="true">
        <clb name="Frequency"/>
      </baseParam>
    </vi>
  </get>
  <process description="Rosetta MIP Survey Frequency">#getClbInfoByIndex($ros_mip_surv_Frequency_table_idx;Frequency;8)</process>
  <output/>
</param>
```

6.2 MODIFICATION DE LA CLASSE « PARAMTABLE » ET DE SES DIFFERENTES IMPLEMENTATIONS

La classe abstraite « ParamTable » du module « Info » a évolué de manière à prendre en charge la définition d'une table variable :

- L'attribut « _variable » précise s'il s'agit d'une table variable ou non,

- La méthode abstraite « getBound » contient l'argument supplémentaire « paramsTableData ». Cet argument est utilisé uniquement dans le cas d'une table variable et contient les données des paramètres de définition d'une table pour le temps étudié.

6.3 PRISE EN COMPTE AU NIVEAU DU « PARAMOUTPUT » DE « PLOT »

6.3.1 Plot de type « timePlot » avec tracé de type « spectro »

Les modifications effectuées se sont limitées à :

- Dans la fonction « createParameters » de la classe « PanelPlotOutput » : Lorsque la résolution de tracé impose un resampling du paramètre, le même resampling est appliqué sur l'ensemble des paramètres de la table variable. De plus, la liste des paramètres de la table variable est ajoutée dans la liste des paramètres à initialiser.
- Dans la fonction « configureSpectroAxis » de la classe « TimePlot » : Le calcul du « range auto » de l'axe portant la table prend dorénavant en compte le cas d'une table variable,
- Dans la fonction « drawSpectro » de la classe « TimePlot » : La définition de la grille de type « MatrixGrid » utilisée pour le tracé du spectro prend en compte le cas d'une table variable.

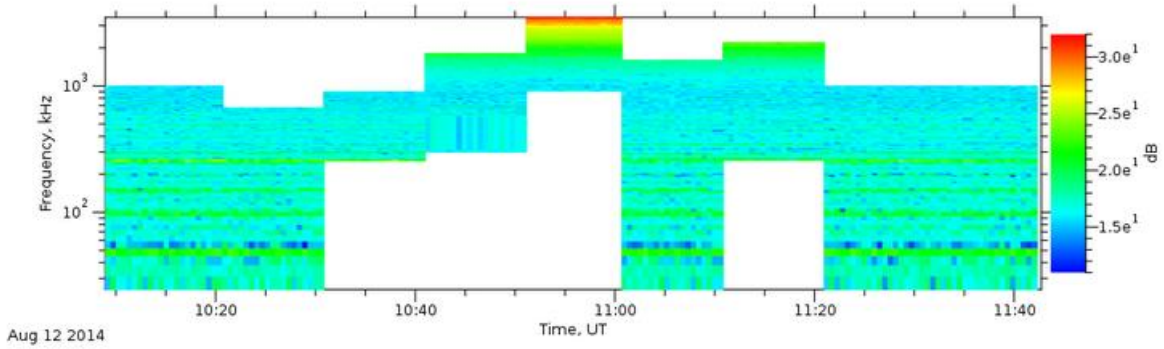
6.3.2 Plot de type « instantPlot »

Les modifications effectuées se sont limitées à :

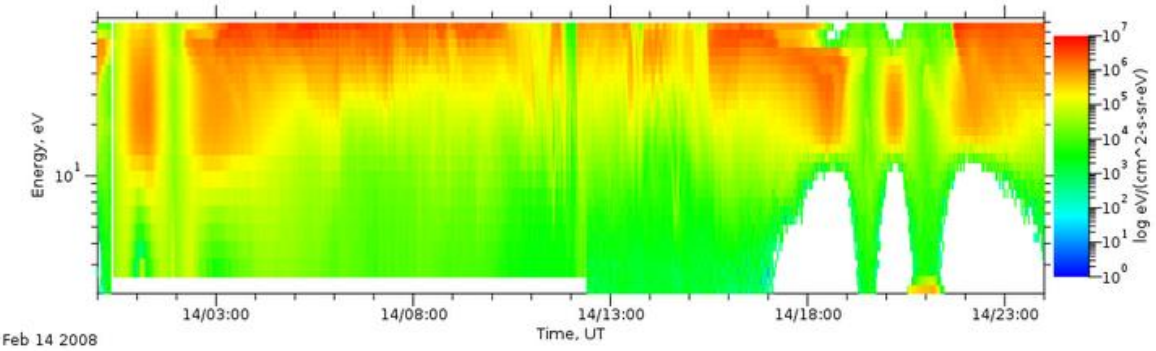
- Dans la fonction « createParameters » de la classe « InstantPlot » : La liste des paramètres des tables variable est ajoutée dans la liste des paramètres à initialiser.
- Dans la fonction « getSerieParameterValues » de la classe « InstantPlot » : La récupération des données de la série à tracer prend en compte le cas d'une table variable,
- Dans la fonction « configureTableAxis » de la classe « InstantPlot » : Le calcul du « range auto » de l'axe portant la table prend dorénavant en compte le cas d'une table variable.
- Dans la fonction « drawSpectro » de la classe « InstantPlot » : La définition de la grille de type « MatrixGrid » utilisée pour le tracé du spectro prend en compte le cas des tables variables.

6.4 QUELQUES EXEMPLES DE TRACES

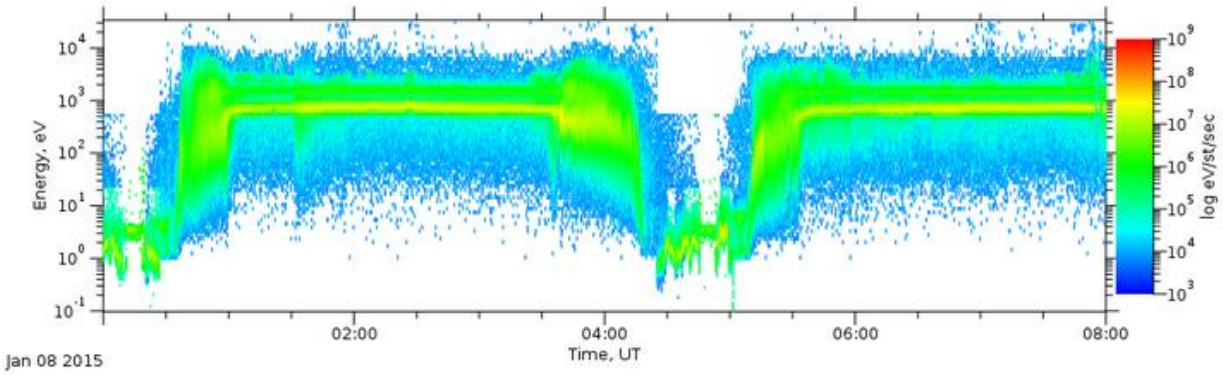
6.4.1 Rosetta MIP Survey



6.4.2 Themis-A PEIR Energy Flux



6.4.3 Maven STATIC C0, Mass 0



7 RESULTATS DES TESTS D'INTEGRATION CONTINUE ET DE QUALITE DU CODE

7.1 RESULTATS DES TESTS D'INTEGRATION CONTINUE

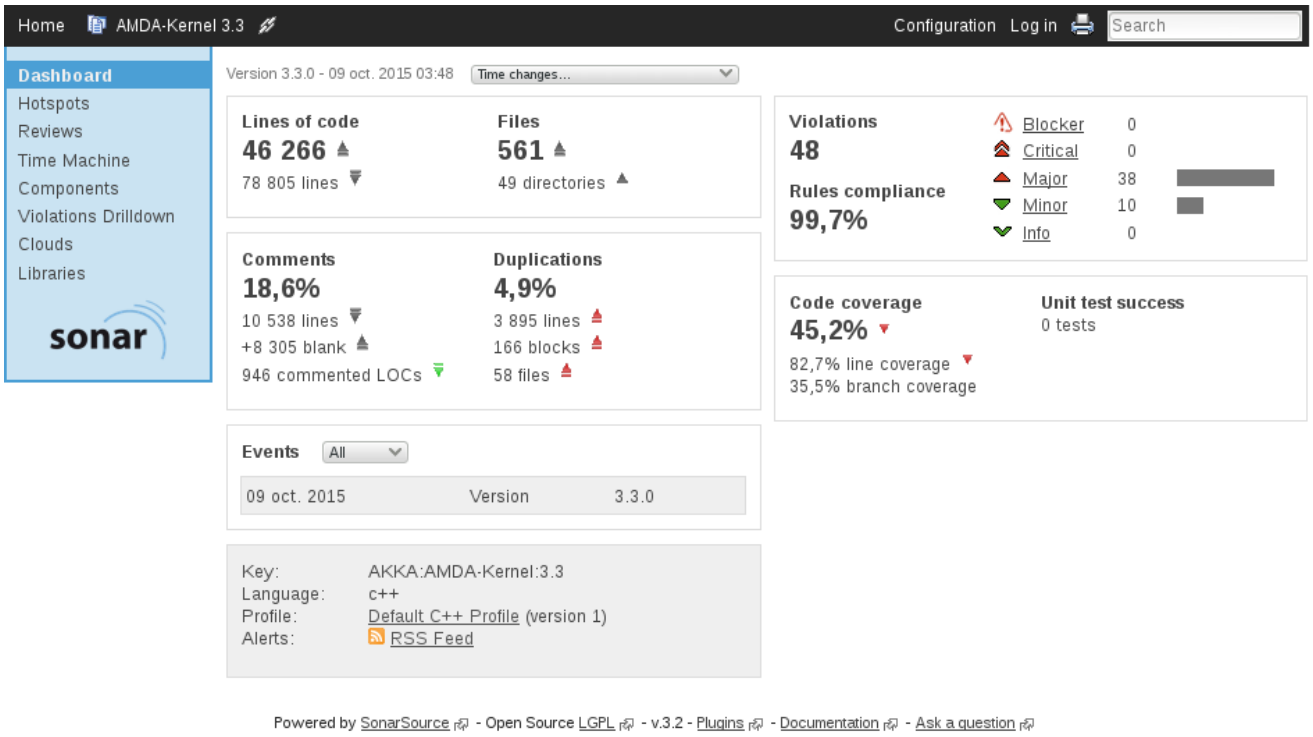
L'ensemble des tests exécutés depuis la plateforme jenkins de l'IRAP sont sortis en succès :

Tous +

S	M	Nom du projet ↓	Dernier succès	Dernier échec	Dernière durée
		AMDA-Kernel_Doxygen	3 mo. 19 j - #3	s. o.	1 mn 23 s
		AMDA_Kernel_Phase3_GCOV	22 h - #43	3 mo. 12 j - #19	1 h 11 mn
		AMDA_Kernel_Phase3_RELEASE	13 h - #11	s. o.	1 h 5 mn
		AMDA_Kernel_Phase3_SONAR	21 h - #14	s. o.	7 h 48 mn

7.2 QUALITE DU CODE

Aucune nouvelle violation n'est apparue :



8 DOCUMENTS APPLICABLES ET DE REFERENCE (A/R)

A/R	Référence	Titre
[R0]	PRC-14421-AMD-AV2	Proposition commerciale – Assistance technique pour des développements sur l'application CDPP/AMDA
[R1]	CDPP-MI-32500-505-SIL	Manuel d'installation du noyau AMDA-NG (3ème partie) et d'intégration avec l'IHM
[R2]	CDPP-CD-32500-502-SIL	Dossier de conception du noyau AMDA-NG (3ème partie) et de son intégration avec l'IHM AMDA.

9 GLOSSAIRE ET ABREVIATIONS

9.1 GLOSSAIRE

Terme	Définition

9.2 ABREVIATIONS

Abréviation	Nom détaillé
IRAP	Institut de Recherche en Astrophysique et Planétologie
CDPP	Centre de Données de la Physique des Plasmas
AMDA	Automated Multi-Dataset Analysis