



Institut Supérieur
d'Informatique de
Modélisation et de
leurs Applications
Complexe des Cézeaux
63173 AUBIÈRE Cedex
FRANCE



Organisation
Européenne
pour la Recherche
Nucléaire
1211 GENÈVE
SUISSE

Stage de deuxième année
Filière Architectures Matérielles et Conception de Circuits

Développement d'un système visualisation des taux du HLT dans le système de supervision PVSS du LHCb

Présenté par : **Jean-François Menou**
Responsable CERN : **Eric van Herwijnen**
Responsable ISIMA : **Emmanuel Mesnard**

Durée : 5 mois
Soutenance :
3 Septembre 2008



Remerciements

Tout au long de ce stage j'ai été en contact avec de nombreuses personnes qui m'ont assisté à différents niveaux de mon travail.

Je tiens donc à remercier Niko NEUFELD et Markus FRANCK pour leur accueil et la visite des installations qui m'a permis de mieux comprendre le contexte de mon travail.

J'aimerais également remercier Juan OTALORA, Bruno SOUZA DE PAULA et Peter SOMOGYI pour leur collaboration et leurs explications sur les logiciels existants.

Je remercie également Clara GASPAR et Carmen BARANDELA pour leur aide quant aux outils utilisés ainsi que Radu STOICA pour son assistance liée aux bases de données et Marco CLEMENCIC pour son aide liée à l'environnement de développement.

J'adresse un grand merci à Loïc BRARDA et Gary MOINE qui ont toujours su me tirer de mes petits problèmes de matériel, me faisant gagner un temps précieux pour la poursuite de mon travail.

Enfin je remercie naturellement Eric VAN HERWIJNEN pour m'avoir donné l'opportunité de découvrir le monde du CERN en m'accordant sa confiance ainsi que Emmanuel MESNARD pour m'en avoir ouvert les portes.

Table des figures

1.1	Architecture du LHCb	3
1.2	Principe du trigger	4
2.1	Logo de PVSS	5
2.2	Architecture d'un système PVSS	6
2.3	Aspect du module PARA	8
2.4	Architecture du framework JCOP	9
2.5	Architecture d'une communication DIM	10
2.6	Exemple de publication DIM	11
2.7	Table des types de MonObjects	12
2.8	Exemple de publication à l'aide de MonitorSvc	13
3.1	Modèle de transfert de données vers PVSS via DIM	18
3.2	Modèle du RateService 1.0	19
3.3	Modèle du RateService 2.0	20
3.4	Modèle du RateService 2.1	21
3.5	Diagramme fonctionnel du RateService	22
3.6	Diagramme UML du RateService	23
4.1	Structure du DPT RateType	29
4.2	Trends PVSS dans un panneau	30
4.3	Exemple de page fwTrending	31
4.4	Arborescence de pages classées par partition	32
4.5	Interface de recherche de services de type taux	36
4.6	Interface de création de pages pour fwTrending	37
5.1	Aspect global des différents composants	39
5.2	Contenu du composant RateCtrl	40
5.3	Tâches et planning	41

Résumé

Dans le cadre de l'expérience **LHCb**, au **CERN**, le groupe Online de l'équipe LHCb est chargé de l'acquisition des données et de la **supervision** de l'expérience. Ces données sont sélectionnées par différents **triggers**, dont le High LevelTrigger (**HLT**). Le HLT est un système logiciel distribué sur une ferme de calcul qui doit réduire le **taux** d'événements jugés satisfaisant de 1 MHz à 2 kHz.

Afin de surveiller le bon fonctionnement du HLT, des taux sont publiés au moyen de **DIM**, aux différents niveaux du HLT. Le but du travail présenté est de fournir un moyen d'afficher et d'archiver ces taux (aussi nommés **rates**) au moyen du logiciel **PVSS**. Ce travail met en œuvre un serveur, le **RateService**, faisant la traduction des données représentant les taux en provenance du HLT, développé en **C++** et un client, le **RateCtrl**, développé sous PVSS et utilisant des composants standards du CERN, chargé de l'affichage et de l'archivage. Le client et le serveur communiquent eux-aussi par DIM.

Le serveur a été intégré dans l'environnement du LHCb à l'aide de CVS et de CMT. Le client a été installé et testé avec les logiciels du LHCb et est disponible sous forme d'un **composant** installable.

Mots clefs: LHCb, CERN, supervision, triggers, HLT, taux, DIM, rates, PVSS, RateService, C++, RateCtrl, composant.

Abstract

At **CERN**, the Online group of the **LHCb** team is in charge of data acquisition and of the **control system**. Interesting data are selected by different **triggers**. The **HLT**, for High Level Trigger, is a distributed system which should reduce the rate of selected events from 1 MHz to 2 kHz.

In order to monitor the HLT's work, some **rates** are published at different levels of the HLT using the **DIM** system. The purpose of the presented work is to provide a way to display and archive this rates in PVSS. This work involves a server converting rates published by the HLT into a format that **PVSS** can handle. This server, called **RateService**, is written in **C++**. A client, called **RateCtrl**, developed as a PVSS project, is in charge of displaying and archiving this converted rates using standard components developed by CERN. Communication between the server and the client is also provided by DIM.

The server is now working and has been commissioned in the pit. The client has been installed and tested. It is available as an LHCb Framework **component**.

Keywords: CERN, LHCb, control system, triggers, HLT, rates, DIM, PVSS, RateService, C++, RateCtrl, component.

Table des matières

Remerciements

Table des figures

Résumé

Abstract

Table des matières

Glossaire

Introduction	1
1 Présentation du contexte	2
1.1 Le CERN	2
1.1.1 Présentation du CERN	2
1.1.2 Le LHC	2
1.2 Le LHCb	2
1.2.1 Le but du LHCb	3
1.2.2 Composition du projet LHCb	3
1.3 Le groupe Online et le HLT	3
1.3.1 Le monitoring au groupe Online	4
1.3.2 Le monitoring au HLT	4
2 Présentation des outils	5
2.1 PVSS	5
2.1.1 Présentation de PVSS	5
2.1.2 Déploiement	8
2.1.3 Utilisation au CERN - framework JCOP	9
2.2 DIM	10
2.2.1 Principe général	10
2.2.2 Implémentation	12
2.3 Gaucho	12
2.3.1 MonObject	12
2.3.2 MonRate	14
3 Serveur de traduction des MonRates en taux	15
3.1 Rôle du module RateService dans la chaîne de supervision	15
3.1.1 Module intermédiaire pour PVSS	15
3.1.2 Client/serveur DIM	16
3.1.3 Traducteur de formats	16
3.2 Développement du RateService	17
3.2.1 Modèle de développement de Online	17
3.2.2 Evolutions du RateService	18
3.2.3 Détail de la version actuelle	21
3.2.4 Adaptabilité	24
3.3 Conventions entre le RateService et PVSS	25
3.3.1 Format de diffusion des données	25

3.3.2	Identification des services de type <i>rate</i>	26
4	Composant PVSS	28
4.1	Système d'acquisition des données	28
4.1.1	Structure des données dans PVSS	28
4.1.2	Parcours et sélection des services	29
4.2	Visualisation des données	29
4.2.1	Objectif de la visualisation	30
4.2.2	Composant de visualisation de JCOP	30
4.2.3	Automatisation et interfaçage avec fwTrending	31
4.3	Archivage et récupération de données	34
4.3.1	But de l'archivage des données	34
4.3.2	Système de gestion des archive de PVSS	34
4.3.3	Bibliothèque de manipulation des archives	34
4.4	Interfaces de test	35
4.4.1	Interface de découverte des services	35
4.4.2	Interface de création de pages	36
5	Résultats et discussion	38
5.1	Tests du RateService	38
5.1.1	Installation du composant	38
5.1.2	Tests fonctionnels	38
5.1.3	Performances	39
5.1.4	Modifications à venir	39
5.2	Test du composant PVSS	40
5.2.1	Déploiement du composant	40
5.2.2	Test des éléments	40
5.2.3	Performances	41
5.2.4	Validation par les utilisateurs	41
5.3	Organisation	41
	Conclusion	43
	Bibliographie	
	ANNEXES	
A	Documentation du serveur C++ RateService	I
B	Documentation du client PVSS RateCtrl	V

Glossaire

Algorithm : classe de Gaudi appelé à chaque événement (collision de particules) pouvant tourner en mode *offline* (données simulées) ainsi que *online* ; une option permet d'exécuter un algorithme sans qu'il y ait des données (utilisé par RateService). **BOOST** : bibliothèque C++ utilisée pour la sérialisation des données.

CERN : sigle du Conseil Européen pour la Recherche Nucléaire, conservé par l'Organisation Européenne pour la Recherche Nucléaire.

CMT : système de gestion des dépendances entre composants pour les différentes versions d'un projet mis en place au CERN.

CTRL : langage de programmation de PVSS ; nom des managers PVSS fonctionnant en tâche de fond.

CVS : système de suivi de versions utilisé pour permettre le "retour" à une version antérieure d'un logiciel.

DIM : Distributed Information Management (System), système de partage d'information entre processus.

DNS : DIM Name Server, processus faisant la correspondance entre un service DIM publié et serveur publiant ce service.

DP : Data Point, structure de donnée interne à PVSS (instance en modèle objet).

DPE : Data Point Element, sous partie d'un DP (attribut en modèle objet).

DPT : Data Point Type, type/structure d'un DP (classe en modèle objet).

ETM : société développant PVSS.

Framework : ensemble de bibliothèques, d'outils et de conventions.

Gauche : framework développé par Online pour la surveillance des compteurs, histogrammes etc... publiés par les algorithmes du HLT.

Gaudi : framework C++ du LHCb qui fournit un ensemble de services standards pour l'analyse des données (*offline* et *online* pour le HLT par exemple). **HLT** : High Level Trigger, trigger logiciel réduisant le taux d'événements à traiter de 1 MHz à 2 kHz.

JCOP : Joint Control Project, framework du CERN dirigeant la réalisation des système de supervision.

L0 : trigger électronique réduisant le nombre d'événements traités de 40 millions à 1 millions par seconde.

LHC : Large Hadron Collider, l'accélérateur de particules du CERN.

LHCb : LHC beauty, expérience du LHC basé sur l'étude du quark *beauty*.

Manager : processus d'un projet PVSS affecté à une tâche (affichage, archivage, exécution de tâche de fond, etc...).

MonObject : Monitoring Object, classe encapsulant des données en provenance d'une tâche Gaudi (ex: HLT) ; utilisé pour la publication sur le réseau à l'aide de Boost (sérialisation) et de DIM (publication de services).

MonRate : type de MonObject contenant principalement des compteurs d'événements.

MonitorSvc : service Gaudi (créé par Gauche) qui assume la publication sur le réseau des MonObjects.

Online : groupe de travail de l'équipe LHC chargée de l'acquisition des données et de la supervision de l'expérience.

Offline : contexte d'exécution des processus pour l'analyse des données (après l'expérience).

Online : contexte d'exécution des processus durant l'acquisition des données.

PARA : module de PVSS permettant la gestion (création, édition, suppression et visualisation) des DPs.

PVSS : Prozessvisualisierungs- und Steuerungs-System, système de supervision et de contrôle de proces-

sus ; logiciel de supervision (SCADA) utilisé pour la surveillance et le contrôle des expériences du LHC.

RateService : algorithme développé afin de convertir des données au format MonObject du HLT en un format utilisable par PVSS.

SCADA : Supervisory Control And Data Acquisition, logiciel de supervision et d'acquisition de données.

TCP/IP : Transmission Control Protocol over IP (Internet Protocol), protocole de transport de données sur réseau.

Trend : Widget permettant l'affichage de l'évolution du valeur dans le temps dans PVSS.

UI : User Interface, interface utilisateur.

UTGID : identifiant des processus tournant sur le HLT.

Trigger : système chargé de sélectionner ou rejeter des événements selon leur pertinence.

Widget : élément d'une interface graphique (bouton, texte, champs de saisie, trend, etc...).

Introduction

La recherche dans le domaine de la physique des particules est un défi relevé quotidiennement depuis près d'un siècle par des scientifiques du monde entier. Aujourd'hui, à Genève, au CERN, un nouvel outil, le LHC, Large Hadron Collider, est sur le point d'être mis à la disposition des physiciens pour les aider à percer les secrets de la matière et comprendre les lois de l'Univers.

La mise en œuvre du LHC a autant impliqué ingénieurs en génie civil et en informatique que physiciens. Le LHC est en effet accompagné de différentes expériences, dont le LHCb, dont le rôle est de détecter les collisions entre particules puis analyser les données ainsi récoltées. Le LHCb comprend donc de nombreux détecteurs situés au plus près des collisions générant un flot d'événement important et des systèmes de sélection chargés de réduire la fréquence des événements de 40 MHz à 2kHz. Le *HLT* est le dernier maillon de la chaîne de sélection. Un système de supervision est chargé de surveiller le fonctionnement des différents systèmes et de piloter les processus de sélection du HLT.

Cette étude a pour but de créer, dans le système de supervision, un nouveau composant permettant l'affichage et l'archivage des taux de sélection diffusés par le HLT. La surveillance de ces taux doit permettre de vérifier le bon fonctionnement de l'expérience. Pour réaliser cet outil, un système intermédiaire entre le HLT et le système de supervision doit être également développé afin d'interfacer ces deux entités. Le langage utilisé pour système intermédiaire est C++, accompagné de nombreuses bibliothèques et de composants développés au CERN. Le système de supervision en lui-même est PVSS, et le composant réalisé utilise les outils de développement de PVSS et les composants précédemment développés au CERN.

Nous nous intéresserons donc tout d'abord à une présentation plus détaillée du contexte de travail et des outils utilisés. Les deux composants réalisés (le système intermédiaire d'interfaçage et le composant de visualisation et d'archivage) seront ensuite décrits quant à leur développement et à leurs fonctionnalités. Enfin, nous présenterons les résultats obtenus et les limites de ce travail à travers l'application en conditions réelles d'utilisation des composants réalisés.

Chapitre 1

Présentation du contexte

1.1 Le CERN

1.1.1 Présentation du CERN

Le CERN, Organisation Européenne pour la Recherche Nucléaire issue du Conseil Européen pour la Recherche Nucléaire (d'où le sigle), est un centre de recherche regroupant 20 pays membres fondé en 1954 avec comme domaine principal la physique des particules à haute énergie. Une de ses vocations est la découverte des constituants et des lois de l'Univers.

Fondé il y a plus de 50 ans, le CERN possède depuis l'été 2008 le plus puissant accélérateur de particules en fonctionnement, le LHC, Large Hadron Collider, le grand collisionneur de hadron.

1.1.2 Le LHC

Le Large Hadron Collider sera, à sa mise en fonctionnement prévue maintenant pour Septembre 2008, le plus grand et le plus puissant accélérateur de particules ayant jamais fonctionné. L'anneau principal de l'accélérateur le long duquel sont installées les quatre expériences principales : ATLAS, ALICE, CMS et LHCb, fait plus de 27 kilomètres de circonférence. Les particules accélérées seront des protons (des noyaux d'atomes d'hydrogène), voyageant par "paquets" de quelques millions à une vitesse proche de celle de la lumière, pour atteindre une énergie de l'ordre du TeV (10^{12} électron-volt soit environ 10^{-7} joules).

Ces énergies (infiniment faibles à l'échelle humaine mais incomensurables à l'échelle subatomique) doivent permettre la création, lors des collisions, de particules dont les différentes expériences seront chargées de détecter la présence et d'analyser le comportement afin de faire avancer la compréhension des modèles physiques en place.

1.2 Le LHCb

Le LHCb [1] est l'une de ces expériences. Il est situé à la toute frontière franco-suisse sur la commune de Prévessin.

1.2.1 Le but du LHCb

LHCb est destinée à l'étude de la violation de CP et à la recherche de désintégrations rares. L'analyse des données portera largement sur les mésons *beauty* (contenant un quark b ou un anti-quark b) [2].

1.2.2 Composition du projet LHCb

Du point de vue matériel, LHCb est un spectromètre à un bras dirigé vers l'avant. En effet les simulations à l'origine de la conception du détecteur ont montré que les paires de quarks - antiquarks b sont émises au point de collision avec un angle petit par rapport aux faisceaux. La région d'intérêt pour détecter ces particules correspond donc au cône formé par l'instrument (voir figure 1.1).

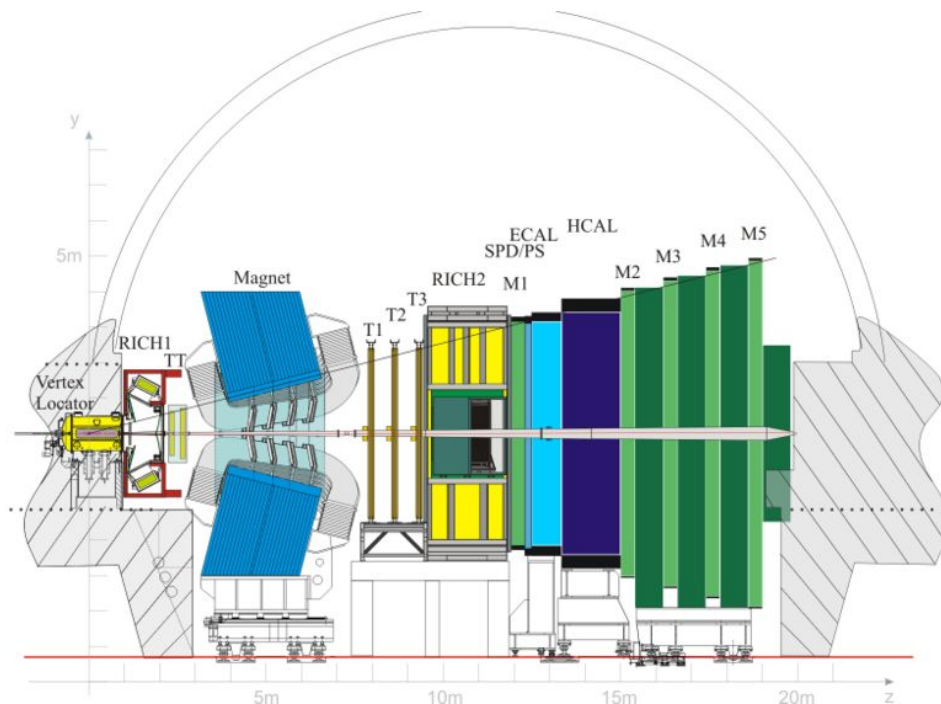


FIG. 1.1 – Le détecteur LHCb composé de ses différents sous-détecteurs.

Les données en provenance des millions de canaux électroniques des différents sous-détecteur du LHCb sont destinées à être analysées et sélectionnées afin de ne retenir que les plus intéressants événements, représentatifs des théorie à vérifier. Ces données sont archivées et destinées à être analysées par la suite.

D'autres données sont quant à elles générées afin de surveiller le bon fonctionnement du système. Ces données suivent un chemin différent et sont destinées à être visualisées et archivées dans le cadre du programme de monitoring de l'expérience. La charge de ce travail de monitoring revient au groupe Online au LHCb.

1.3 Le groupe Online et le HLT

Le groupe Online est, entre autre, en charge du monitoring du LHCb. Ceci signifie qu'il doit développer et maintenir les applications nécessaires à la surveillance du bon fonctionnement des installations (capteurs, alimentations, systèmes de refroidissement, etc...).

1.3.1 Le monitoring au groupe Online

Le groupe Online est composé de physiciens et d'informaticiens développant les différentes applications de *monitoring* du LHCb. Le monitoring consiste à surveiller l'évolution de valeurs représentatives de l'état du système. Ces valeurs peuvent être affectées par différents facteurs liés aux conditions de l'expérimentation, mais aussi, par exemple, à la suite d'incidents tels que des coupures de courant mettant le système dans un état indéterminé.

1.3.2 Le monitoring au HLT

Le HLT est le dernier trigger de la chaîne de sélection des événements. Son rôle est de réduire la quantité de données à traiter par la suite en ne gardant que les événements les plus intéressants d'un point de vue physique. Comme les autres systèmes qui composent le LHCb, il doit être surveillé pour vérifier son bon fonctionnement. Le bon fonctionnement du HLT est entre autre défini par le taux de sélection qu'il applique aux événements qui lui sont soumis.

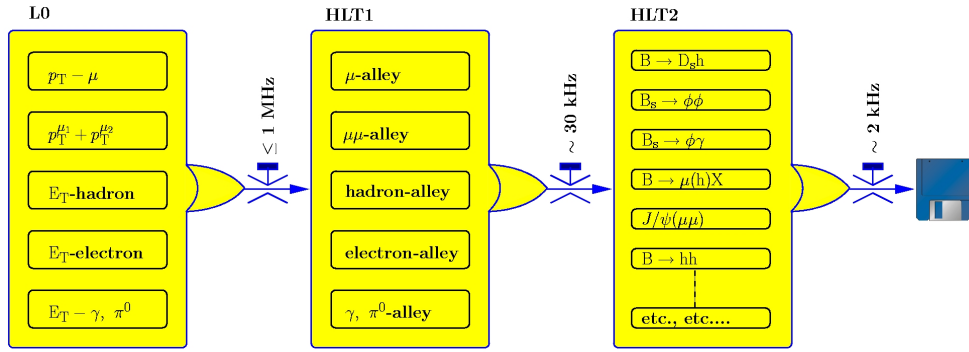


FIG. 1.2 – Le HLT réduit le taux d'événements sélectionnés de 1MHz à 2kHz à la suite du trigger L0 [3].

Le HLT est un ensemble de processus de sélection déployés sur une ferme de calcul d'un millier de CPUs ou nœuds. Chacun de ces nœuds exécute environ huit de ces processus. Le HLT a une structure arborescente. À chaque niveau de cette arborescence, des processus spéciaux, les *Adders*, comptent les événements sélectionnés et d'autre processus en font la somme pour la communiquer au niveau suivant. À la racine de cette arborescence on retrouve donc un décompte total des événements sélectionnés.

Les différents décomptes aux différents niveaux du HLT permettent donc de mesurer son taux de sélection. En théorie, le HLT doit recevoir des événements à une fréquence de 1 MHz et réduire ce taux à 2 kHz, comme l'explique la figure 1.2.

Conclusion

Les différents systèmes mis en place au LHCb nécessitent des nombreux outils pour être exploités et surtout supervisés. Ce sont ces outils que nous allons étudier par la suite.

Chapitre 2

Présentation des outils

Le développement de systèmes, tant matériels que logiciels est une constante au CERN. Ces systèmes sont basés sur des composants extérieurs ou développés en interne dont certains, comme DIM, peuvent trouver une application au-delà des laboratoires du CERN [7].

2.1 PVSS

Dans le cadre du CERN, les expériences menées prennent des proportions industrielles. Le complexe du LHC comprend entre-autre, rappelons-le, quatre détecteurs de taille gigantesque comprenant non seulement des millions de capteurs mais aussi des systèmes d'alimentation en énergie et de refroidissement nécessitant un contrôle à distance (les zones d'expérience sont soumises à des températures, des champs magnétiques et des taux de radiation rendant impossible leur accès lors de leur utilisation).



FIG. 2.1 – Logo de PVSS [5].

2.1.1 Présentation de PVSS

PVSS est un logiciel développé par la société autrichienne ETM (figure 2.1). PVSS signifie Prozessvisualisierungs- und Steuerungs-System c'est-à-dire système de visualisation et de contrôle de processus. C'est un logiciel de type SCADA : un logiciel de supervision et d'acquisition de données et est conçu pour permettre à l'utilisateur de créer son propre système de supervision. PVSS est donc un outil plus qu'un système de supervision à part entière, et est à la fois l'environnement de développement et d'exécution des systèmes créés [4].

Architecture de PVSS

PVSS permet de créer des systèmes de tailles variées de par son architecture. Il est en effet lui-même conçu selon une architecture modulaire qui permet, au moment de l'exécution du système créé, de charger les différents composants du système (possibilité d'activer

et de désactiver des modules). Les quantités de données manipulées au CERN sont, en outre, considérables. Le CERN et ETM collaborent pour adapter PVSS aux besoins particuliers du CERN afin d'assurer la continuité du service tout au long de son utilisation dans les décennies à venir.

Un système PVSS est composé de processus communicant par TCP/IP appelés *managers*¹ tels que représentés sur la figure 2.2.

On trouve des managers qui forment le cœur d'un système dont le *event manager* et le *database manager*. Ces managers assurent respectivement la communication entre les autres managers et la persistance des données.

Un système PVSS contient généralement des *driver managers* qui assurent la communication à travers la plupart des protocoles de communication industriels (Profibus, IEC, Modbus, Applicom, etc...). Ces managers permettent donc d'établir la communication (envoyer des signaux de contrôle, lire des valeurs sur des capteurs, etc...) avec des composants tels que des alimentations électriques ou des capteurs utilisant ces protocoles standards dans l'industrie.

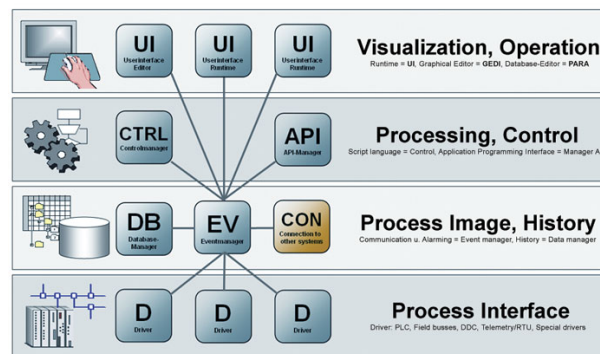


FIG. 2.2 – Architecture d'un système PVSS [5].

Les systèmes PVSS ne sont cependant pas uniquement en communication avec des composants matériels. Dans le cadre de mon travail il est notamment nécessaire de communiquer avec d'autres systèmes logiciels. PVSS permet le développement de managers spécialisés tels que, dans le cas présent, un client pour le protocole DIM (voir 1.2.3). Ces managers sont les *API managers*.

Enfin, afin de contrôler et visualiser les processus à superviser, PVSS comprend deux autres classes de managers : les *CTRL*² managers et les *UI*³ managers. Les premiers sont dédiés à l'exécution de scripts en tâche de fond, les seconds à l'affichage d'interfaces utilisateurs. Ces interfaces peuvent être produites à l'aide d'un éditeur graphique et leur comportement face à l'utilisateur scripté à l'aide du même langage utilisé dans les CTRL managers.

CTRL

CTRL (control) est le langage de script utilisé par les CTRL managers et les interfaces. Il est fortement inspiré du langage C pour sa syntaxe. Il est donc de ce fait facilement assimilable par tout développeur ayant une connaissance de ce langage (informaticien ou

1. Nombre de termes sont d'origine anglaise et sont utilisés dans le langage courant, au CERN, dans leur version originale quelle que soit la langue parlée ; aussi les utilisera-t-on ici, parfois accordés.

2. control

3. user interface (interface utilisateur)

physicien au CERN).

CTRL se caractérise par un comportement proche des langages orientés objet pour la manipulation des widgets, les objets graphiques des interfaces (appel de méthode sur des références vers ces widgets par notation pointée, exemple : `textArea.text = "My text"`). Il existe également des types particuliers comme les tableaux de données dynamiques qui se caractérisent notamment par le fait commencer à l'indice 1 au lieu de 0 (de ce point de vue, CTRL est un peu perturbant dans les premières heures d'utilisation pour un informaticien habitué au C). Le tout visant à simplifier le développement en se détachant des détails techniques propres aux langages de programmation tels que le C.

Il est à noter que CTRL est un langage interprété, ce qui signifie que les erreurs de syntaxe et de sémantique ne sont détectables qu'au moment de l'exécution, ce qui peut parfois gêner le développement en masquant des erreurs qui n'arrivent que dans des cas particuliers.

Sauvegarde des données

La sauvegarde des données dans PVSS est possible à moyen et long terme à travers deux mécanismes que nous allons décrire.

a. Sauvegarde interne

Dans PVSS, les données nécessitant d'être stockées à moyen terme (d'une utilisation sur l'autre du système) sont sauvegardées dans une base de données interne. Cette base de données contient des types de données et des instances de ces types. Une donnée structurée est appelée un *DataPoint* (DP). Un DP est d'un certain type défini par les éléments qui le constituent ; les types sont appelés *Datapoint Type* (DPT). Les éléments composant un DP sont les *DataPoint Elements* (DPE). Les DPEs sont soit des types du langage CTRL (entier, réel, chaîne de caractères, tableau dynamique des différents types, etc..) soit des nœuds (subdivision du DP contenant d'autres DPEs).

L'outil privilégié pour gérer les DPs est le module *PARA* de PVSS. *PARA* permet de créer, supprimer, éditer les DPs et les DPTs (voir figure 2.3). Toutes ces opérations sont cependant également disponibles à travers le langage CTRL.

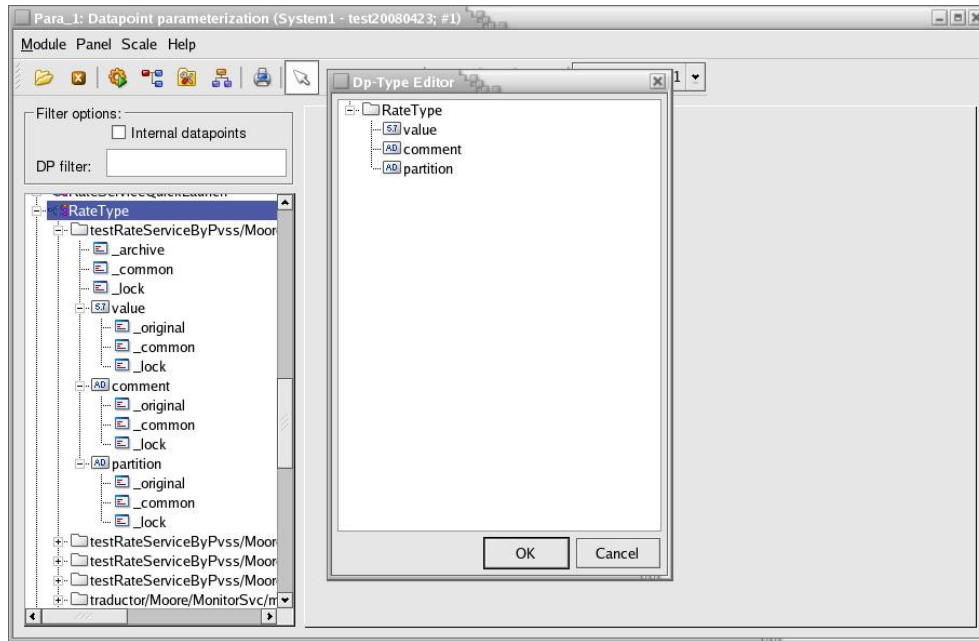


FIG. 2.3 – Aspect du module PARA, d'un DP de type *RateType* (avec des différents DPs et configs) et de l'éditeur de DPT montrant les DPEs de *RateType*.

En tant que SCADA, PVSS permet d'ajouter aux données des caractéristiques (appelées *configs*) permettant de gérer les plages de valeurs d'un DPE, les alarmes, etc... Ces *configs* permettent également de définir les règles d'archivage (config *_archive* sur la figure 2.3) des DPs en base de données.

b. Bases de données

La sauvegarde des DPs en base de données est déterminée par le paramétrage de la config *_archive* pour les DPs ou DPEs concernés. La config *_archive* détermine un manager qui se charge de sauvegarder les données selon des critères paramétrables (plage de valeur neutre pour une valeur, limite dans le temps, etc...).

Gestion des événements

PVSS est fortement orienté événements. Par exemple, dans le cadre d'une communication DIM, une fois une donnée mise à jour dans un service souscrit, le DPE correspondant est automatiquement mis à jour.

PVSS permet également l'utilisation d'un mécanisme de *callback* associant une fonction à exécuter à un DPE à surveiller. Quand le DPE change de valeur, la fonction est exécutée.

Cette gestion des événements permet de n'avoir aucune surcharge de transfert d'information entre les managers (communiquant par TCP/IP rappelons-le) en ne transmettant les données que lorsque cela est nécessaire.

2.1.2 Déploiement

Afin d'assurer la meilleure intégration possible de ses systèmes, PVSS fournit plusieurs modes de déploiement :

- *système simple* : une collection de managers tournant sur une même machine phy-

sique.

- *système éclaté* : de même que pour le système simple mais les managers peuvent se trouver sur des machines différentes ; permet d'exécuter un projet en utilisant les interfaces sur une machine distante par exemple.
- *système distribué* : un système distribué est un ensemble de systèmes communiquant entre eux à l'aide d'un manager spécifique (un dist manager) ; pour chaque système les managers peuvent évidemment être exécutés sur n'importe quelle machine, il s'agit donc en fait d'un ensemble de systèmes éclatés.

Un système simple est donc identifié par son event manager, car il y a un et un seul event manager par système. Chaque système, qu'il soit isolé ou compris dans un système distribué possède un nom. Ce nom permet d'identifier les DP's d'un système dans le cas d'un système distribué par exemple.

2.1.3 Utilisation au CERN - framework JCOP

Au CERN, l'ensemble des expériences du LHC utilisent PVSS pour la supervision de leurs différents systèmes physiques et logiciels. Dans un souci d'unité au sein des systèmes d'un même groupe scientifique, et même à l'échelle du CERN, un ensemble de règles de programmation et de composants (voir figure 2.4) a été édité sous le nom de framework JCOP, pour Joint Controls Project.

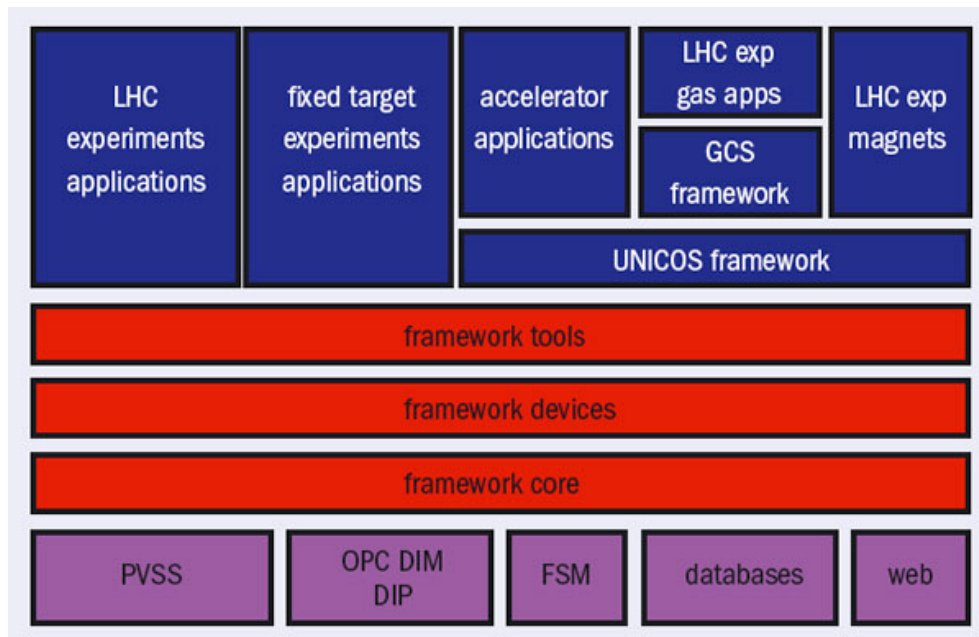


FIG. 2.4 – Architecture du framework JCOP mêlant composants achetés (PVSS, base de données Oracle, etc...) et produits au CERN (DIM, FSM, etc...) [6].

L'initiative JCOP date de 1997. Précédemment, au CERN, chacun des groupes chargés du développement du système de contrôle (DCS : detector control system) d'une expérience procédait indépendamment. La nécessité d'une factorisation est apparue à cause de la taille croissante des systèmes mis en place ainsi que de l'augmentation de leur complexité face aux ressources limitées en hommes et en moyens [6].

Pour les expériences du LHC, le CERN et les différentes équipes de développement travaillent ensemble aux systèmes de contrôle des détecteurs, selon une approche conjointe.

Le but fixé et atteint était l'élimination des développements redondants, la favorisation l'échange d'expériences et des connaissances et la mise en place un support centralisé.

JCOP se traduit dans PVSS par l'existence de bibliothèques de scripts et d'interfaces adaptées aux différentes expériences (du fait de la convergence des moyens utilisés au niveau des détecteurs).

Pour le développement des systèmes PVSS, un certain nombre de lignes de conduite sont également édictées par le framework, pour faciliter la lisibilité et l'utilisation des interfaces produites par exemple.

Le framework JCOP définit également l'usage d'autres composants tels que DIM.

2.2 DIM

DIM (Distributed Information Management System) est un système de communication inter-processus développé au CERN. Les principales caractéristiques de DIM sont d'être basé sur le modèle client-serveur et de fournir un support pour de nombreuses plateformes (Windows et Linux principalement).

2.2.1 Principe général

DIM permet le transfert d'informations, principalement des nombres et des chaînes de caractères, entre *serveurs* et *clients* en gérant les différences possibles de représentation des valeurs (un entier peut être codé binaires de façons différentes entre le client et le serveur, néanmoins la valeur représentée reste la même à la suite du transfert par DIM).

DIM a une architecture particulière qui fait intervenir trois interlocuteurs dans l'établissement d'une communication (voir figure 2.5).

Dans le système DIM, on trouve des serveurs qui publient des services, des clients qui s'abonnent à des services et une entité qui répertorie les services publiés : le DNS, pour DIM Name Serveur.

Le rôle du DNS est de répertorier les services publiés par les serveurs pour permettre aux clients d'entrer en communication avec la machine qui publie un service recherché. Avec DIM on ne sait pas qui publie un service, mais le nom du service recherché suffit à le trouver s'il existe.

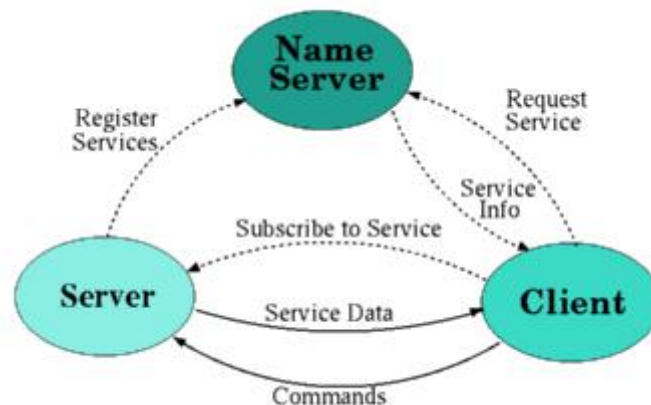


FIG. 2.5 – Architecture d'une communication DIM [7].

DIM permet de publier des données en fournissant un format. Un service contient donc une (service *scalaire*) ou plusieurs (service *structuré*) données. Pour publier un service scalaire on associe (en C++) une variable de type scalaire (entier, réel, etc...) à un service; pour publier un service structuré on associe une structure (au sens C du terme) à un service comme le montre l'exemple ci-dessous (figure 2.6) :

```
/* déclaration d'un service "scalaire" */
DimService * mySimpleService;
double myDouble;

/* le service créé s'appellera "DOUBLE_EXAMPLE_SERVICE" et contiendra la va-
leur
   la variable myDouble.*/
mySimpleService = new DimService("DOUBLE_EXAMPLE_SERVICE", (double*)&myDouble;

/* déclaration d'un service "structuré" */
DimService * myStructuredService;
struct data_t{
    float myFloat;
    char myChar[20];
};
int sizeofStruct = sizeof(float) + 20 * sizeof(char);
data_t myStructure;
myStructuredService = new DimService("STRUCTURED_EXAMPLE_SERVICE", "F:1;C",
                                   (void*)&myStructure, sizeofStruct);

:

/* la mise à jour des service peut être forcée de cette manière.
   on peut également fournir une autre variable pour la mise à jour
   que celle utilisée pour la déclaration du service.*/
mySimpleService->updateService(myDouble);
myStructuredService->updateService((void*)&myStructure, sizeofStruct);
```

FIG. 2.6 – Déclaration et mise à jour de services scalaires et structurés dans DIM.

Lorsqu'un client s'abonne (souscrit) à un service, il précise la façon dont il veut être informé des changements de valeur du service.

Le client a la possibilité de spécifier une valeur de temps au bout de laquelle il sera informé de la valeur courante du service. Le client est également informé lors de tout changement de façon automatique si le serveur est configuré pour prévenir ses clients (une fonction permet d'informer tous les clients d'un service pour un service donné).

DIM se caractérise également par sa gestion des incidents. Si un serveur "meurt", tous ses clients sont informés de l'événement et tenteront de se reconnecter aussitôt que le serveur sera à nouveau opérationnel (c'est-à-dire, aussitôt qu'un serveur publiant le service

perdu se mettra à émettre). Ceci permet de récupérer facilement des situations de crash, mais aussi de gérer la migration d'un service d'une machine sur une autre [7].

2.2.2 Implémentation

Il existe de nombreuses implémentations de DIM à l'heure actuelle au CERN. On trouve des implémentations en C, C++, Java, Fortran et Python pour Windows et Linux permettant le développement d'application communicant par DIM.

Il existe également un module PVSS (sous forme d'un API manager, voir 2.1.1), remplissant les fonctions de client et de serveur DIM.

L'implémentation sur de nombreuses plateformes et environnements, et *a fortiori* sur toutes les plateformes utilisées dans le cadre des expériences concernées, donne à DIM sa propriété de couche intermédiaire liant toutes les applications et assurant la continuité des données, de la ferme de calcul aux écrans de contrôle.

2.3 Gaucho

Le groupe Online développe actuellement un composant nommé *Gaucho* [8], ayant pour objectif le formatage des informations transitant entre les différentes applications. *Gaucho* signifie "GAUdi Component Helping Online", en effet, le framework Gaudi [9] est dédié aux processus offline. Gaucho permet de conserver les interfaces de programmation de Gaudi pour un contexte d'exécution online. Gaucho introduit un concept particulier pour la transmission des données relative au *monitoring* au LHCb. Comme on la vu dans la partie 1.3.2 le monitoring du HLT implique la transmission de données d'un niveau à l'autre de l'arborescence. Afin de rendre ces données plus facilement manipulables dans les programmes, un format, appelé MonObjects a été créé.

2.3.1 MonObject

Les MonObjects sont une classe d'objets ayant pour but d'encapsuler tout type de donnée susceptible de transiter à des fins de monitoring. Le but du MonObject est d'être échangé entre processus. Comme on l'a vu, le système DIM est utilisé pour le transfert d'information interprocessus au LHCb.

Les MonObjects se déclinent sous plusieurs formes pour représenter les différentes données échangeables comme le montre le tableau suivant (figure 2.7) :

Type C++	MonObject	Préfixe	Type C++	MonObject	Préfixe
$\emptyset$	MonObject	MonObj	long	MonLong	MonL
-	MonH1F	MonH1F	bool	MonBool	MonB
-	MonH1D	MonH1D	pair<double, double>	MonPairDD	MonPDD
-	MonH2F	MonH2F	pair<int, int>	MonPairII	MonPII
-	MonH2D	MonH2D	pair<double, int>	MonPairDI	MonPDI
-	MonProfile	MonP1	-	MonHitMap2D	MonHM2D
int	MonInt	MonI	-	MonStatEntity	MonSE
double	MonDouble	MonD	vector<int>	MonVectorI	MonVI
float	MonFloat	MonF	vector<double>	MonVectorD	MonVD
char*, string	MonString	MonS	ensemble de compteurs	MonRate	MonR

FIG. 2.7 – Les différents types de MonObjects correspondant aux type du C++ à diffuser et les préfixes utilisés pour nommer les services DIM.

Afin d'échanger ces données sous forme de `MonObject`, il existe un composant nommé *MonitorSvc*. Le *MonitorSvc* sert d'adaptateur pour DIM. Afin, par exemple, de transmettre une données entière avec DIM sous forme de `MonInt` (la forme `MonObject` des entiers) on se sert du *MonitorSvc* pour déclarer un service associé à cette donnée. La figure 2.8 nous donne un exemple de publication à l'aide du *MonitorSvc* :

```
int myInteger;

/* déclaration d'un service pour une variable entière à l'aide du
   MonitorSvc : MonInt, le service sera nommé MY_MONITORING_OBJECT et
   le MonInt aura le commentaire "This is a MonInt service." */
declareInfo("MY_MONITORING_OBJECT", myInteger, "This is a MonInt service");

int myCounter1, myCounter2, myCounter3;

/* déclaration de compteurs devant être encapsulés dans un MonRate.
   le premier paramètre donne l'identifiant du compteur (counter1Id),
   le second donne la variable à surveiller pour mettre à jour le
   MonRate (myCounter1), la troisième donne le commentaire à associer
   au compteur (My first counter).
   Le service sera nommé d'après le UTGID du processus exécutant ce code. */
declareInfo("COUNTER_TO_RATE[counter1Id]", myCounter1, "My first counter");
declareInfo("COUNTER_TO_RATE[counter2Id]", myCounter2, "My second counter");
declareInfo("COUNTER_TO_RATE[counter3Id]", myCounter3, "My third counter");

:

/* les compteurs et autres valeurs publiées peuvent être modifiées librement,
   le MonitorSvc se charge de la mise à jour des services pour nous. */
myCounter1++;
myCounter2++;
myCounter3++;
myInteger = 5214;
```

FIG. 2.8 – Déclaration d'un entier et de compteurs avec *MonitorSvc* : un service *MonInt* est créé pour l'entier et un service *MonRate* (pouvant déjà exister) est utilisé pour publier les compteurs.

Transport des `MonObjects` dans DIM

Comme on l'a vu dans la partie 2.2, DIM ne permet que deux type de transferts : le transfert de données scalaire et de structures de données. Les `MonObjects` sont des *structures* trop complexes pour pouvoir être transférées par DIM. Il faut donc pouvoir les transformer de façon à les rendre transportables par DIM.

Cette opération est la *sérialisation*. La *sérialisation* consiste à transformer un objet complexe tel qu'un `MonObject` en une suite binaire. On utilise pour cela la bibliothèque Boost⁴. Boost propose une fonctionnalité d'*archivage* réalisant la sérialisation désirée. La suite binaire obtenue peut ensuite être considérée comme une chaîne de caractères (car le langage C++ permet de changer le type d'une données de façon très souple et donc de

4. www.boost.org

considérer toute suite binaire comme une suite de caractères).

DIM est donc chargé par le MonitorSvc de transférer cette chaîne de caractères. Le service peut être souscrit par un client pour récupérer le MonObject. Une classe particulière, *DimInfoMonObject* écrite par Juan OTALORA, permet de s'abonner à un service de type MonObject et d'en extraire la valeur. Si le MonObject en question est un MonInt, la classe permet de récupérer ce MonInt. La manipulation du MonInt est ensuite laissée à la discrétion du client.

Nous verrons par la suite que cette classe n'est pas utilisée dans le RateService car elle ne permet pas de définir de callback sur le service surveillé (ici un service contenant un MonRate). La classe développée est décrite dans la partie 3.2.3 : DimInfoMonRate.

2.3.2 MonRate

Les MonObjects se déclinent également sous un type particulier appelé *MonRate*. Les *MonRates* sont les objects qui transportent les compteurs à travers le HLT (voir partie 1.3.2). Ils contiennent donc deux types d'informations qui vont nous intéresser par la suite. Premièrement, les MonRates contiennent une liste de compteurs, repérés par un identifiant (permettant de déterminer la quantité mesurée par ce compteur, voir figure 2.8). Ces compteurs mesurent le nombre d'événements sélectionnés par les différents niveaux du HLT.

On trouve deuxièmement, dans les MonRates, des informations temporelles. En effet, la mesure des événements sélectionnés dans le HLT s'effectue sur une période de temps donnée pour chaque nœud. Les mesures de temps permettent d'assurer la cohérence des mesures (les compteurs) au moment d'ajouter ces données au niveaux des Adders (les processus effectuant la somme des Adders d'un niveau sur l'autre).

Ces deux informations doivent permettre de calculer les taux de sélection aux différents étages du HLT.

Conclusion

L'utilisation des différents outils présentés dans cette partie pour la réalisation du RateService et des composants PVSS doit normalement assurer, par leur bonne utilisation, un fonctionnement et une maintenance optimale dans le contexte du groupe Online.

Par ailleurs, une bonne maîtrise des différents concepts s'est avérée indispensable pour réaliser le travail demandé.

Chapitre 3

Serveur de traduction des MonRates en taux

Au LHCb, le HLT publie, grâce à DIM, des compteurs permettant de surveiller son activité et son bon fonctionnement. On a vu également que le système utilisé pour la supervision de l'expérience est PVSS. La transmission des données produites par le système de monitoring du HLT à PVSS est donc essentielle pour pouvoir assurer sa supervision.

3.1 Rôle du module RateService dans la chaîne de supervision

Afin de transmettre les informations en provenance de la ferme de calcul vers PVSS, en vue de leur affichage et de leur sauvegarde, il y avait donc besoin au LHCb d'un programme permettant la traduction des données dans un format compréhensible par PVSS. En effet, les données de type *taux* en provenance de la ferme sont empaquetées et diffusées dans des objets de type *MonRate*.

Les MonRates, en tant que services DIM nécessitent un traitement particulier pour pouvoir être compris par un client. Le client DIM de PVSS n'ayant pas été conçu pour réaliser ce traitement, il est nécessaire d'intercaler un élément entre le HLT et PVSS pour assurer cette traduction de format.

Le serveur fournissant les données à PVSS, développé dans ce sens, est appelé le *RateService*. Ce serveur réalise deux tâches pour remplir son rôle : découvrir les services MonRate pour extraire les taux de leurs différents compteurs d'une part et publier ces taux vers PVSS d'autre part.

3.1.1 Module intermédiaire pour PVSS

Pour PVSS le support de DIM est tel qu'il permet uniquement l'association d'un service scalaire à un DPE ou d'un service structuré à un DP. La modification du client DIM de PVSS pour lui permettre d'effectuer les traitements réalisés par RateService a été jugée trop complexe par rapport à la réalisation du RateService. Le RateService permet de conserver le client généraliste développé pour PVSS (souscription à des services scalaires ou structurés uniquement) en adaptant les services émis par le HLT sous forme de MonObjects sérialisés sous forme binaire.

Techniquement, le RateService est un *Algorithm* dans la terminologie Gaudi. Un *Algorithm* peut être chargé dans un programme (*Gaudi*) et exécuté sur une machine cible (voir

3.2.1).

Le nombre de RateService n'est pas encore déterminé mais dans les conditions actuelles il devrait y avoir une seule instance de RateService pour l'ensemble de la ferme de calcul.

3.1.2 Client/serveur DIM

Etant développé au CERN pour le CERN, le RateService utilise les composants standards du CERN, parmi lesquels DIM. Les données traitées par le RateService et celles qu'il émet sont toutes diffusées à l'aide de ce composant. Le RateService se sert donc des fonctionnalités de recherche, abonnement et déclaration de services.

La nature du RateService est de convertir les données du format MonRate au format scalaire compréhensible par PVSS. Pour se faire il faut que tout changement de valeur d'un service entrant soit répercuté dans le service sortant équivalent.

Afin de répercuter un tel changement il faut que les traitements soient réalisés dans un temps relativement court (*relativement* signifie court par rapport au temps entre deux changements de valeur par exemple). Il serait possible avec DIM de demander une mise à jour du service entrant à une fréquence régulière et estimée suffisante pour pouvoir observer tous les changements et ainsi modifier les services sortants en conséquence. Ce genre de comportement est appelé *polling*.

Il est également possible, et c'est cette technique qui est employée par le RateService, d'utiliser la fonctionnalité de *callback* de DIM (une fonction est automatiquement exécutée lors du changement de valeur d'un service, voir 2.2.1).

Les avantages et les inconvénients du polling et du mécanisme de callback sont liés : le polling ne permet pas de voir tous les changements mais il permet de maîtriser la charge de travail du programme qui traite les données ; le callback, quant à lui, permet de traiter tous les changements de valeur, mais à condition que tous les traitements soient effectués, c'est-à-dire que le système ne soit pas surchargé par de trop nombreux changements de valeurs.

Ainsi, en se reposant principalement sur le système mis en place par DIM, on assure la découverte des nouveaux compteurs du MonRate, la diffusion des données extraites et la robustesse au problème de service MonRate *décédé*¹.

3.1.3 Traducteur de formats

Le RateService est un traducteur de données. Il s'abonne à des services DIM contenant des données ayant un sens (logique) et un format donné et diffuse des services contenant la même valeur logique dans un format différent, compréhensible par PVSS.

Dans le cas le plus avancé du développement, les données entrantes sont des MonRates (des objets de monitoring contenant des compteurs) sérialisés à l'aide de la bibliothèque Boost et diffusés à l'aide de DIM.

Le principe de la traduction des données est simple car les mêmes composants sont utilisés dans les différents programmes de Online pour la diffusion des données. Ici les objets MonRates sont sérialisables et désérialisables² dans une archive Boost, et les archives Boost

1. Le processus fournissant le service peut *mourir*, entraînant un arrêt de la diffusion du service.

2. La sérialisation permet de représenter un ensemble de données comme une suite de valeurs élémentaires (valeurs binaires) pouvant être stockées sur un support ou transmises à un élément capable de les reconvertir (désérialiser) pour retrouver les données originelles.

sont diffusables et lisibles en tant que services DIM (sous forme de chaînes de caractères). Il convient d'utiliser correctement les composants mis à notre disposition pour assurer la continuité des données à travers les différents étages de la chaîne de traitement.

Le fait que les mêmes composants sont utilisés pour la diffusion et la récupération des données, ainsi que l'architecture spécifique des programmes de Online permet également de supporter efficacement les mises à jours (utilisation de nouvelles versions de composants pour le projet en développement) des composants (voir 2.3), ce qui rend la maintenance d'autant plus facile.

3.2 Développement du RateService

Au CERN comme dans tous les groupes de développement, on trouve des normes et des habitudes qui dirigent la façon de développer les applications. Dans le groupe Online, ces directives sont complétées par la structure du framework mis en place depuis des années pour rationaliser le développement et la maintenance des applications.

3.2.1 Modèle de développement de Online

Le groupe Online a en effet mis en place un framework structuré autour d'un type d'application : *Gaudi*. Ce modèle repose sur le prédicat que tous les programmes du type du RateService seront développés sous la forme d'un algorithme qui sera chargé dans un programme dont le but est de le *dérouler*.

Le concept d'algorithme a déjà été développé en 1.3 et 2.3 mais il doit en ressortir que, indépendamment de la fonction attendue du programme, il y a un comportement commun à tous les algorithmes qui leur permet d'être gérés de la même façon par les structures de contrôle adaptées.

Les algorithmes comportent un état et peuvent réagir à des commandes (démarrer, mettre en pause, arrêter...), ce qui les rend manipulables aisément. Les algorithmes sont également associés à un fichier de configuration au moment de leur lancement dans Gaudi, ce qui les rend adaptables à leur environnement de fonctionnement. Dans notre cas, le fichier de configuration peut, par exemple, définir un modèle pour les noms des services à rechercher et traiter (selon un usage qu'il faudra définir par la suite).

Ce fichier peut également contenir des informations quant à la localisation du DNS (voir 2.2.2) ou quant au niveau d'information à afficher sur la sortie standard (simples informations, warnings, erreurs fatales, etc...³).

Afin d'intégrer les différents composants du framework nécessaires au fonctionnement d'un nouvel élément, les projets Online comportent un fichier répertoriant différentes informations telles que le nom du nouveau composant créé dans le projet, la version du composant, la structure du projet et les composants (en indiquant le nom, la version et le répertoire du composant) à utiliser. Ce fichier, nommé *requirements* permet au système CMT [10] d'utiliser les versions voulues des différents composants utilisés par le projet. Le projet du RateService utilise le composant Gauchon dans sa version 6.0 (*v6r0* d'après la nomenclature de Online) et la bibliothèque Boost dans sa version la plus récente disponible.

3. Il existe un système particulier pour afficher des informations depuis un algorithme en spécifiant le niveau d'importance du message.

Ce mécanisme permet lui aussi de faciliter le développement et la maintenance. Sa compréhension n'a toutefois pas été aisée et a pris une partie du temps nécessaire à la découverte des outils mis à ma disposition.

3.2.2 Evolutions du RateService

La maîtrise des concepts mis en place ainsi que des outils (DIM, Gaucho, etc...) est passée par le développement en plusieurs étapes de prototypes de RateServices.

GaudiExample : modèle de diffusion

Afin de prendre en main les techniques utilisées, une première étape a été de travailler sur une copie d'un algorithme nommé *GaudiExample*. Cet algorithme (dans sa version 2.5) génère des séries de valeurs et les diffuse via DIM sous le format reconnu par PVSS. Le premier format choisi pour la diffusion de ces informations est un couple de services DIM contenant, pour l'un, la *valeur* du taux et pour l'autre le *commentaire* associé. Les noms des services étaient alors de la forme suivante : `nomServiceValeur` pour la valeur et `nomServiceValeur/comment` pour le commentaire.

La modification de cet algorithme (principalement en terme de nombre et type de services diffusés) a permis non seulement de comprendre la technique à utiliser pour la déclaration et la mise à jour de services, mais aussi, en parallèle de tester le composant PVSS lors de son développement.

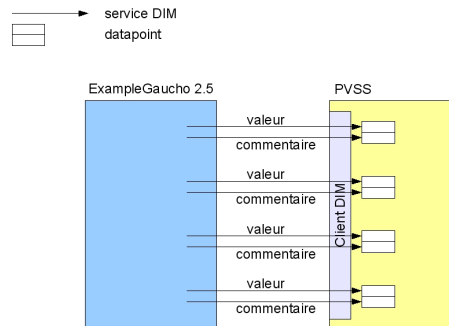


FIG. 3.1 – La communication vers PVSS est à l'origine établie en diffusant un service DIM pour la valeur du taux et un autre pour le commentaire associé.

RateService 1.0

La première véritable version du RateService à avoir été développée n'était pas destinée à son usage final (à savoir la traduction des MonRate). Cette version avait les caractéristiques et le comportement suivants : tout d'abord, cette version (dite 1.0) avait la caractéristique de s'abonner à des services de type *MonDouble*⁴.

4. *MonDouble* : MonObject ayant une valeur de type double (réel), sérialisable comme tout MonObject et de ce fait pouvant être transmis via DIM et reconstruit par la suite dans le programme abonné au service.

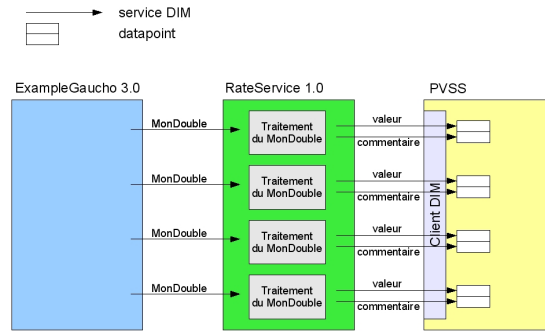


FIG. 3.2 – Le RateService 1.0 s’abonne à des MonDoubles et diffuse des paires de services vers PVSS.

Chacun des services de type MonDouble trouvé donnait lieu à la création d’une unité chargée de traiter les données (valeur et commentaire) à chaque changement de valeur du service (en utilisant le mécanisme de callback) et de diffuser une paire de services (voir 3.3.1). Grâce à DIM la diffusion des données est extrêmement facile à programmer : il suffit de déclarer un service en lui donnant un nom et une variable à surveiller. Quand la variable indiquée change, DIM est chargé de modifier la valeur du service.

Pour fonctionner, la version 1.0 tirait les données à traiter d’un autre algorithme : le *GaudiExample 3.0*. Cet algorithme est en fait le même que le *GaudiExample 2.5* à la différence près qu’il utilise une version plus récente de Gaucho qui introduit la diffusion des valeurs scalaires (entiers, réels, chaînes de caractère, etc...) directement sous la forme de MonObjects (respectivement MonInt, MonD, MonString, etc...) ⁵. C’est ici un exemple de modularité des programmes développés par Online (le changement de version de Gaucho utilisé dans le fichier `requirements` suffit à faire utiliser par le projet la version désirée sans changer le code source).

Une caractéristique de cette version 1.0 était d’utiliser la phase d’exécution ⁶ de l’*algorithm* pour effectuer une nouvelle recherche de services à convertir. Ceci permettait de maintenir à jour une liste de services convertis. Les services en cours de conversion sont susceptibles de disparaître. En effet, le processus les diffusant peut être arrêté, peut supprimer le service ou bien les services peuvent disparaître d’eux-mêmes en cas de disparition du processus.

Dans un tel cas c’est DIM lui-même qui gère cette situation : l’abonné au service ne reçoit plus de mise à jour et réagit comme si le service était tout simplement inchangé, c’est-à-dire dans notre cas en faisant rien. En cas de réapparition du service, le RateService est simplement notifié du changement de valeur du service et réagit en conséquence.

Pour le RateService 1.0, il est à noter que les services diffusés ne le sont pas de la même façon que dans le GaudiExample.

En effet pour pouvoir diffuser les données sous forme de MonObject, le GaudiExample inclus un autre algorithme appelé MonitorSvc chargé de créer et de sérialiser les MonObject.

De son côté, afin de communiquer directement des données scalaires à PVSS, le RateSer-

5. Via le *MonitorSvc*.

6. Un *algorithm* comporte une phase d’*initialisation* et de *finalisation* qui réagissent généralement à des commandes de démarrage et d’arrêt et une phase d’exécution qui est répétée tant qu’il est censé s’exécuter.

vice fait directement appel aux fonctions de création de service de DIM.

RateService 2.0

La version 2.0 (*v2r0* sous Online), diffère de la précédente du fait qu'elle s'abonne non plus à des MonDoubles mais à un MonRate. Les MonRates étant des MonObjects structurés de façon à contenir des compteurs ainsi que diverses informations relatives au contexte de ces compteurs (commentaire associé, date du cycle d'actualisation de ces compteurs, dernier événement apparu lors du dernier cycle, etc...). Le contenu du MonRate est lié à la position du processus qui le produit dans l'arborescence des processus du HLT.

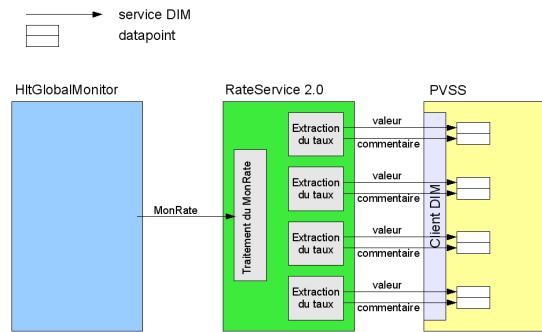


FIG. 3.3 – Le RateService 2.0 s'abonne à un MonRate, calcule des taux à partir des compteurs et diffuse des paires de services.

Dans notre cas précis, le processus produisant un MonRate pour un RateService donné est situé au “sommet” d’une sous-ferme et diffuse les informations collectées dans cette sous-ferme et additionnées par les différents Adders. Ce processus exécute un algorithme nommé *HltGlobalMonitor*.

Dans les phases de développement de la version 2.0, le programme utilisé pour fournir le MonRate exécutait un HltGlobalMonitor qui tirait les données à diffuser sous forme de MonObject d’un générateur d’événements. Cette version n’a pas pu être testée en conditions réelles d’exploitation. En effet en conditions réelles, ce générateur n’était pas accessible (problème lié aux réseaux) et les tests auraient dû être réalisés directement sur les services produits par le HLT (chose impossible car le HLT n’était pas encore prêt). Du fait de cette utilisation du générateur pour tester le RateService, les données utilisées avaient une durée de diffusion relativement courte (de l’ordre de 3 à 4 minutes pour lire et diffuser toutes les données) ce qui a rendu les tests relativement peu représentatifs de la fiabilité du RateService.

Comme l’utilisation du RateService prévoit qu’il soit assigné à un MonRate, il n’y a pas, comme dans la version 1.0, de recherche systématique de nouveaux services lors de la phase d’exécution. En revanche le fichier d’options de l’algorithme se trouve enrichi d’une nouvelle information, à savoir le nom du service à rechercher et à traduire.

RateService 2.1

La version 2.1 du RateService apporte des modifications significatives de comportement, mais l’architecture du code ne s’en est pas trouvé grandement modifiée.

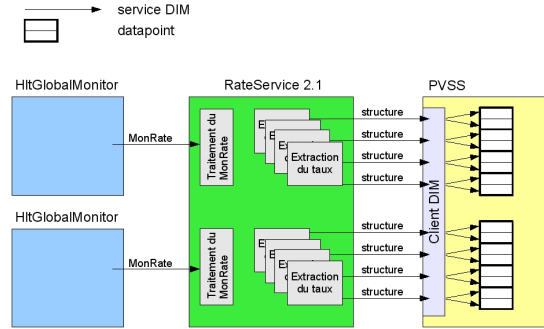


FIG. 3.4 – Le RateService 2.1 s’abonne à des MonRates, extrait les valeurs de leurs compteurs et diffuse les taux sous forme de services DIM structurés.

Dans cette version, il est spécifié qu’un seul RateService pour l’ensemble des sous-fermes sera exécuté. De ce fait, il devra être capable de traiter l’ensemble des MonRates présentés par les différents Adders de chacune des sous-fermes (le Adder final de chaque sous-ferme), ceci étant, il est possible de prévoir l’utilisation d’un RateService pour traduire une partie des MonRates, en spécifiant un format de services à traduire dans le fichier d’options par exemple.

La version 2.1 apporte principalement une modification significative du mode de transmission des données vers PVSS. On passe dans cette version d’un mode scalaire à un mode structuré (voir 3.3.1). Les données de valeur et de commentaire du taux sont ici transmises dans un même service.

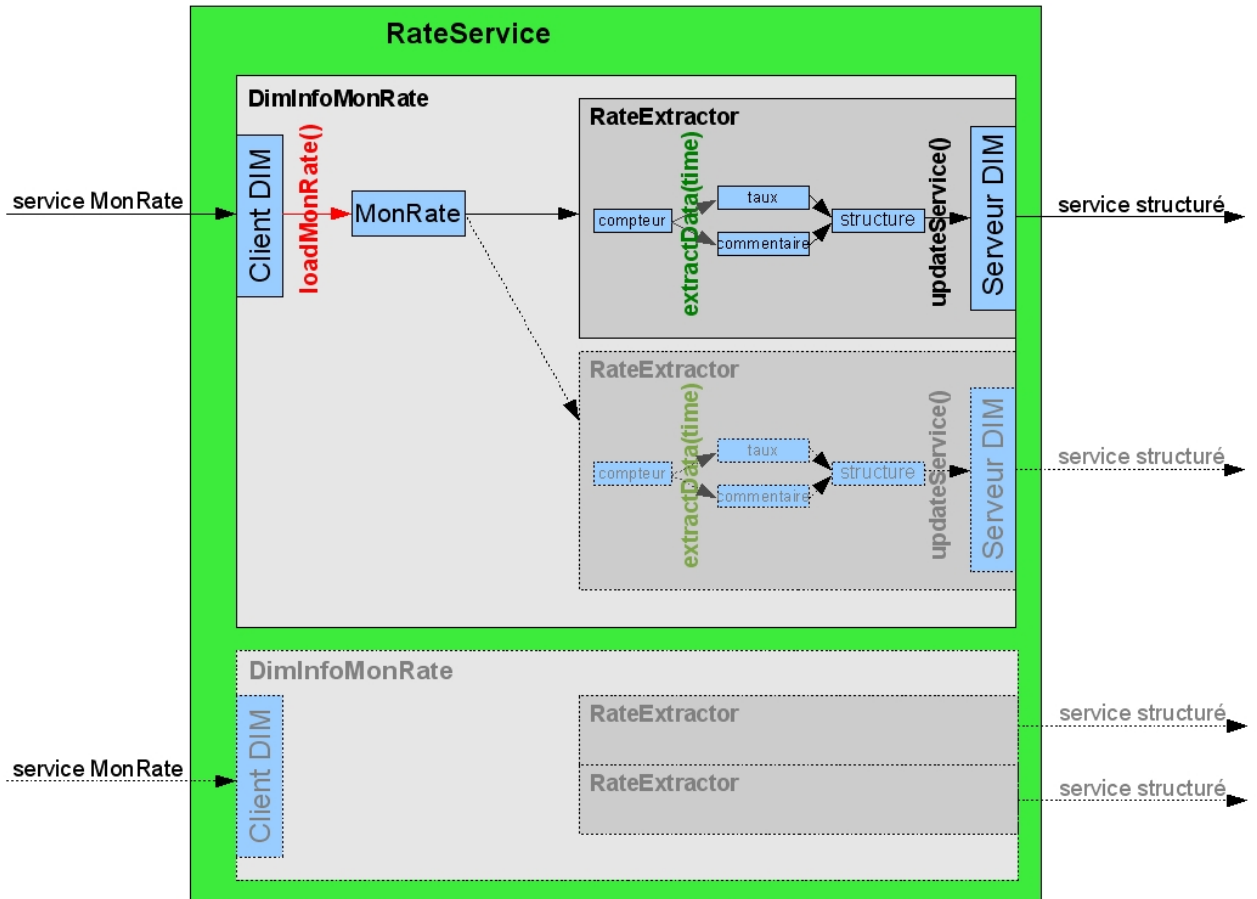
3.2.3 Détail de la version actuelle

Le RateService actuel est un algorithme. Il agit principalement dans sa phase d’exécution. Il est capable de gérer le traitement de plusieurs services MonRate simultanément grâce à l’emploi des callbacks de DIM. Il publie les taux vers PVSS sous forme de services DIM structurés.

Phase d’exécution de l’algorithme

Le RateService est capable de traiter plusieurs services MonRate à la fois. Il lui faut donc rechercher ces services. Pour ce faire, une méthode (`findServices()`) fait appel aux fonctionnalités de DIM pour parcourir les services répondant aux critères de sélection définis dans le fichier d’options.

La méthode de recherche obtient une liste de services susceptibles d’être traités et commence par comparer ces services avec ceux déjà en cours de traitement (dans la liste des *DimInfoMonRates* `m_dimInfoMonRate`, voir figure 3.6). Si des services ne sont pas encore traités (`isHandledService()`) on crée un *DimInfoMonRate* pour chacun de ces services.



loadMonRate() : méthode appelée par callback
extractData(time) : méthode invoquée par le DimInfoMonRate

FIG. 3.5 – Le *RateService* gère un *MonRate* à l’aide d’un *DimInfoMonRate* et chaque compteur est géré par un *RateExtractor* qui publie les données vers PVSS après formatage.

DimInfoMonRate

Afin de traiter un service DIM contenant un *MonRate* il faut un client capable d’interpréter les *MonObjects*. Il existe dans Gauchio une classe ayant le rôle de client DIM pour des services *MonObject*. Cette classe est *DimInfoMonObject* (voir 2.3.1 : *Transport des MonObjects dans DIM*). Elle permet d’obtenir le *MonObject* contenu dans le service DIM par l’appel à sa méthode `loadMonObject()`.

Cette classe aurait pu être utilisée, plutôt que de développer *DimInfoMonRate*, mais elle avait l’inconvénient de ne pas implémenter la fonctionnalité de callback de DIM. En d’autres termes, le *RateService* aurait dû effectuer des `loadMonObject()` à intervalles réguliers pour vérifier le changement du service par un système de boucle. Le fait de se débarrasser de cette boucle apporte une meilleure clarté au code et rend surtout le *RateService* plus réactif aux changements subis par les services *MonRate* provenant du HLT. La classe *DimInfoMonRate* a donc été développée en se basant sur le modèle de *DimInfoMonObject* en la faisant hériter de *DimInfo* (la classe de base des clients DIM) pour bénéficier de la fonctionnalité de callback implémentée par la méthode `infoHandler()` (voir figure 3.6).

Le point d'entrée du fonctionnement de DimInfoMonRate est donc la méthode `infoHandler()`. Dans cette méthode on trouve les éléments suivants :

- Dans un premier temps, sachant que le service a été modifié on tente d'extraire le MonRate du service en appelant `loadMonRate()`, comme le montre la figure 3.5. Afin de se protéger contre toute corruption des données contenues dans le service, il est fait un usage massif du système d'exceptions⁷ dans le RateService. En effet les méthodes appelées, que ce soit celles écrites par moi-même, par les développeurs de Gauchou ou de la bibliothèque standard du C++ ou encore de Boost font usage des exceptions en cas d'erreur. Donc si une exception est détectée lors de l'exécution de `loadMonRate()` on sait qu'il y a une erreur que l'on peut alors notifier à l'utilisateur.
- Un fois le MonRate extrait du service, on invoque la méthode `extractData()` qui est chargée d'analyser le contenu du MonRate. Cette méthode va donc d'abord rechercher des nouveaux compteurs dans le MonRate qui ne seraient pas encore pris en charge. Comme au niveau du RateService, le DimInfoMonRate tient à jour une liste des compteurs qui sont déjà en cours de traitement à l'aide de l'attribut `m_extractorMap` qui associe une instance de *RateExtractor* (voir figure 3.6) à l'identifiant d'un compteur.
- Toujours dans `extractData()`, on effectue ensuite la conversion des compteurs en taux à proprement parler. Il faut premièrement récupérer les mesures de temps contenues dans le MonRate afin de calculer Δt l'intervalle de temps depuis la dernière conversion. La valeur de l'attribut `timeLastEvInCycle` du MonRate est passée comme paramètre à la méthode `extractData(longlong time)` du *RateExtractor* gérant chacun des compteurs. La méthode `extractData(longlong time)` est quant à elle chargée d'effectuer le traitement du compteur que traite le RateExtractor.

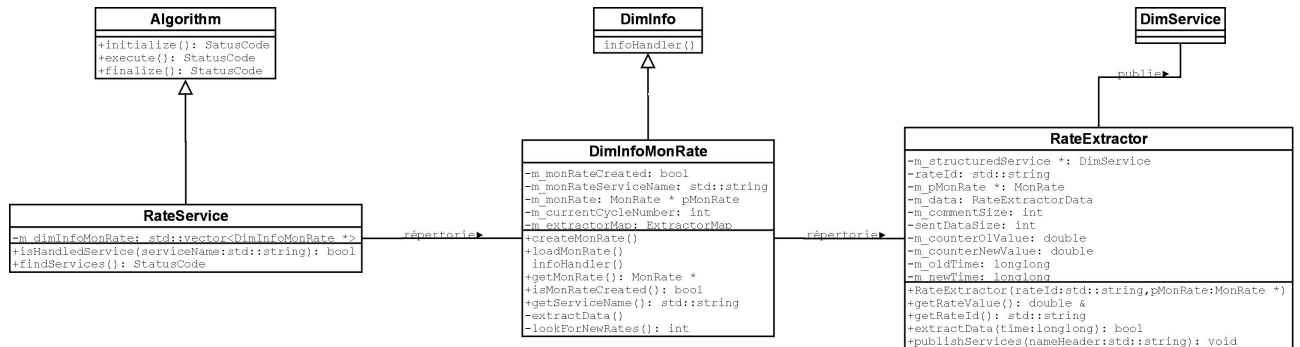


FIG. 3.6 – Les différentes classes interagissant au sein du RateService 2.1.

RateExtractor

Afin de gérer indépendamment chaque compteur et le convertir en taux, j'ai développé la classe RateExtractor. La localisation des fonctions propres à la conversion des compteurs en taux dans une classe doit permettre une plus grande modularité du code et une plus grande facilité de maintenance et de mise à jour.

Le but du RateExtractor est donc, d'une part, de calculer le taux à partir du compteur⁸ et d'extraire le commentaire⁹ et, d'autre part, de publier ce taux et le commentaire sous

7. Système utilisé dans certains langage dont le C++ pour transmettre les messages d'erreur.

8. `m_counterNewValue = m_pMonRate->counter(m_rateId);`

9. `std::string comment = m_pMonRate->counterDescription(m_rateId);`

forme d'un service approprié pour DIM :

- La fonction d'extraction des taux `extractData(longlong time)`, invoquée par `DimInfoMonRate`, permet de calculer le taux en calculant premièrement Δt , l'intervalle de temps depuis la dernière conversion. Pour cela, le `RateExtractor` compare la valeur du paramètre `time` avec sa valeur lors du précédent appel : $\Delta t = time_n - time_{n-1}$. On calcule ensuite Δc , la différence entre la nouvelle valeur du compteur (obtenue en récupérant le compteur dans le `MonRate` à l'aide de l'identifiant du compteur) et l'ancienne valeur sauvegardée par le `RateExtractor`. On obtient donc le taux par le rapport $taux = \Delta c / \Delta t$.
- La construction du service pour PVSS passe par l'affectation du taux et du commentaire à une variable type `RateExtractorData` (structure C contenant un attribut réel `value` et un tableau de caractères `comment`). Afin de forcer la mise à jour du service (ce qui a pour conséquence d'en informer "immédiatement" tous les clients, c'est-à-dire PVSS), on fait appel à la méthode `updateService(void * data, int dataSize)` du serveur DIM (voir figure 3.5) instancié dans le `RateExtractor` (`m_structuredService`) avec la structure et sa taille comme paramètres (voir exemple 2.6).

Le `RateExtractor` est donc subordonné au `DimInfoMonRate` pour l'extraction et la publication des taux. Le `DimInfoMonRate` étant réactif aux changements de valeur du service `MonRate` qu'il traite, la publication des taux est effectuée, elle-aussi, de façon réactive ce qui limite le délai de répercussion de l'information vers PVSS.

3.2.4 Adaptabilité

Un des soucis majeurs dans les expériences du CERN est de pouvoir assurer le fonctionnement des installations (des matériels et des logiciels) pour les durées d'exploitations prévues, qui sont dans le cas du LHC de l'ordre de plusieurs décennies. Il convient donc d'avoir les outils les plus robustes et les plus adaptés possibles aux situations futures. L'avantage du logiciel est de pouvoir être relativement vite remplacé en cas de changement du modèle mis en place (un changement dans l'architecture des fermes de calculs par exemple). Il doit cependant pour ce faire pouvoir être facilement modifié.

L'architecture des systèmes Online permet une répercussion rapide et efficaces des modifications de packages vers les packages utilisateurs. Cependant le cœur du problème est le code en lui-même du `RateService` qui, s'il est semble-t-il adapté aux conditions actuelles d'utilisation, pourrait avoir à subir des modifications de structure radicales en cas de modification du contexte.

Fort heureusement, grâce à l'utilisation des outils du framework, le code de l'algorithme en lui-même ne représente qu'une quantité relativement faible. Le fait de connaître et comprendre le framework de Online permettrait de facilement se focaliser sur la fonction même de l'algorithme s'il était besoin de la modifier.

En effet les particularités fondamentales du `RateService` reposent sur la façon d'extraire les données du `MonRate`, chose qui est soumise à la façon dont elles sont empaquetées dans le `MonRate` et qui est aisément compréhensible par quelqu'un qui aura une certaine connaissance des MonRates ; la deuxième particularité est la façon dont les données

extraites sont présentées à PVSS.

3.3 Conventions entre le RateService et PVSS

Afin d'assurer la communication entre les RateServices et PVSS, on utilise un média commun : DIM. DIM n'est qu'un outil qu'il convient d'utiliser de la façon la plus adaptée à l'usage que l'on veut en faire.

Dans le cas présent, et du fait des caractéristiques du client DIM présent dans PVSS, il a été décidé d'une façon de présenter les informations à transmettre à travers DIM.

3.3.1 Format de diffusion des données

Le format de diffusion des données a évolué au cours des différentes versions du RateService (cette évolution s'est également répercutée dans le composant PVSS). En effet, avec DIM, il est possible de transférer les données de plusieurs façons : sous forme scalaire (un élément de type entier, réel, chaîne de caractères, etc...), ou sous forme structurée.

Dans un premier temps, pour reprendre la convention adoptée par le précédent système (GaudiExample), c'est la forme scalaire qui a été choisie. Avec la forme scalaire, il faut un service DIM par donnée à transférer, c'est-à-dire un pour la valeur du taux et un pour le commentaire.

Imaginons le cas courant d'un taux classique : le commentaire représente la signification du taux et ne change que peu (ou pas du tout) ; la valeur, elle, est susceptible de changer "souvent". Quand la valeur change, on ne transmet sur le réseau que cette donnée là, sans retransmettre le commentaire qui, lui, n'a pas changé. Ce mode de fonctionnement permet donc en théorie des économies de bande passante.

En revanche, le coût en nombre de services est deux fois plus élevé qu'avec une solution structurée.

En effet, en mode structuré, on ne déclare qu'un seul service pour transmettre la valeur et le commentaire du taux.

A l'inverse de la solution scalaire, le changement de la valeur du taux entraîne la transmission de la valeur du service (valeur du taux et commentaire) à tous les clients, mais le gain en nombre de services est appréciable.

Le mode structuré est présent à partir de la version *v2r1* du RateService.

C'est en effet le nombre de services qui est une ressource critique dans le cadre des transmissions de données avec DIM. En effet, le DNS est un système qu'il convient de ne pas surcharger avec de trop nombreux services.

Notons que ce service structuré sous la forme d'une valeur et d'un commentaire peut rappeler les services MonDoubles (objet de monitoring transportant des données de type réel).

On peut se demander pourquoi l'on a pas directement utilisé ce format de données pour transmettre les taux à PVSS. La réponse est que le client DIM de PVSS est fait exclusivement pour associer directement un service de type scalaire à un DPE ou un service structuré à des DPEs d'un DP.

3.3.2 Identification des services de type *rate*

La communication entre les RatesServices et PVSS doit passer par l'établissement d'un protocole permettant d'intenfier les services transportant des données de *taux*.

Cas de la communication par paire de services

Dans le cas de la communication par paire (jusqu'à la version 2.0 du RateService), la définition d'un service de type *rate* passe par deux étapes.

Premièrement, il doit exister deux services tels que l'on a un service **x** et un service **x/comment**, le service **x** transportant la valeur et **x/comment** le commentaire.

Dans PVSS cette vérification peut se faire aisément en vérifiant l'existence du service **x/comment** pour tout service **x** susceptible de contenir une valeur (en effet on peut récupérer la liste complète des services auprès du DNS).

Deuxièmement il faut vérifier qu'un service **x** est susceptible de contenir une valeur de taux. Pour ce faire, on vérifie que le type de donnée du service **x** est bien réel.

Pour s'assurer que **x** et **x/comment** représentent bien des **rates**, il a été ajouté une convention consistant à ajouter un identifiant au début du commentaire (c'est-à-dire à la donnée du service **x/comment**). Cet identifiant est la chaîne de caractères **[rate]**.

Concrètement on pouvait trouver comme services produits par un RateService v1r0 les éléments suivants :

- **rate1** : un service transportant une valeur réelle.
- **rate1/comment** : un service transportant par exemple **[rate]event rate**.

Cas de la communication par service structuré

Dans le cas de la communication structurée, qui est le dernier mode de communication choisi à ce jour, les deux données sont diffusées à travers un seul service. Il n'est donc plus nécessaire comme dans la communication par paire de vérifier la cohérence et l'existence d'un service "**x**" et d'un service "**x/comment**".

De plus, avec le passage à ce mode de diffusion, il a été décidé d'inclure l'identifiant (permettant de certifier la nature "*rate*" du service) directement dans le nom du service lui-même.

L'avantage de cette façon de procéder est que l'on peut directement, au moment où l'on effectue la recherche des services auprès du DNS, spécifier un format correspondant à cet identifiant pour ne récupérer que les services de type taux. Ceci évite, dans PVSS, d'avoir à effectuer un tri des services un par un.

Enfin, l'inclusion de l'identifiant dans le commentaire dans le cas de la communication par paire obligeait la fonction de recherche des services de PVSS à s'abonner à chacun des services *commentaire* pour vérifier son contenu. Cette opération était longue et coûteuse en ressources pour le système PVSS ainsi que pour le DNS.

Conclusion

Le RateService présente donc les taux via DIM, sous forme de services structurés, à tout client susceptible de traiter ce genre d'informations. Il s'agit ici de PVSS, ou plus précisément d'un ensemble de panneaux et de scripts développés spécifiquement afin de proposer une visualisation et une gestion de l'archivage de ces données.

Chapitre 4

Composant PVSS

PVSS est un SCADA, et en tant que tel il permet l'acquisition de données représentant des mesures à surveiller et leur visualisation. PVSS fournit également un support de l'archivage de ces données. Ce sont ces trois propriétés : acquisition, visualisation, archivage, qui sont utilisées dans les différents éléments du composant développé pour le traitement des taux en provenance du RateService dans PVSS et nommé *RateCtrl*.

4.1 Système d'acquisition des données

Les données surveillées dans PVSS proviennent d'une part de composants communiquant par le biais de protocoles de type industriel (profibus, etc...) et d'autre part de logiciels diffusant des informations à l'aide de protocoles tels que DIM. Les données reçues dans les modules PVSS développés proviennent du logiciel de traduction et sont donc transmises par DIM. Pour acquérir ces données il faut procéder à un certain nombre d'opérations dans PVSS que nous allons développer.

4.1.1 Structure des données dans PVSS

Les données en provenance du logiciel de traduction sont composées en elles-mêmes de deux parties : une valeur et un commentaire. L'utilisation des *taux* fait cependant apparaître une autre information qu'il est utile de mémoriser, à savoir la partition à laquelle appartient un taux. Une partition représente une source d'informations (calorimètre, alimentation, refroidissement, etc...).

Ces trois informations sont pour l'instant les seules associées à un taux. Il est possible que des informations complémentaires soient à ajouter à la structure de données des rates dans PVSS pour éventuellement s'adapter à l'évolution du concept de taux au sein du LHCb.

Dans la configuration actuelle du projet, on extrait la valeur et le commentaire du service lui-même, et la partition du nom du service. Cette configuration est elle aussi susceptible d'être modifiée.

Par exemple le service `calo/Moore/MonitorScv/monRate/Hlt1AlleyOr` provient de la partition `calo` (pour calorimètre).

Dans PVSS on retrouve la structure de données par l'existence d'un DPT (voir figure 4.1) constitué de 3 DPEs : `value` la valeur de type *double* (réel), `comment` le commentaire de type *string* (chaîne de caractères) et `partition` de type *string*.

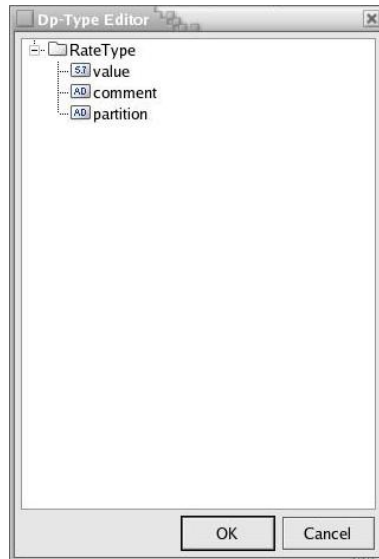


FIG. 4.1 – Structure du DPT pour le stockage des rates dans PVSS.

La création de ce DPT est automatisée par la fonction `createRateDPTType()` de la bibliothèque.

4.1.2 Parcours et sélection des services

La bibliothèque développée pour l'acquisition des données permet la découverte de tous les services définis comme des services de taux pour un DNS donné.

La découverte des services se fait pas le biais de la fonction `discoverRateService()`. Il est possible d'automatiser la découverte des services (afin de tenir à jour dans PVSS une liste des services et de pouvoir les visualiser et les archiver) en faisant appel à cette fonction à intervalles réguliers en exécutant un *CTRL script* dans un *CTRL manager* (fonctionnalité présente dans la dernière version testée).

Afin d'établir la communication avec le logiciel de traduction, PVSS a besoin d'un client DIM. Ce client se présente sous la forme d'un manager DIM à ajouter au projet et à configurer avec en particulier le nom du DNS auquel s'abonner. La communication DIM à travers ce manager est assurée par la bibliothèque `fwDIM` du framework JCOP.

La découverte des services se passe en deux étapes : premièrement, récupérer une liste de services auprès du DNS dont les noms correspondent à un modèle (ce modèle est actuellement en cours de discussion et est défini dans la bibliothèque) ; deuxièmement pour chacun de ces services, vérifier si il est déjà enregistré dans PVSS ou non, et si non, procéder à son enregistrement.

L'enregistrement comprend trois étapes : l'abonnement au service, la création d'un DP pour stocker les données provenant du service et la définition des paramètres d'archivage. L'abonnement est fait de telle sorte que, à tout changement de valeur du service, le DP soit modifié en conséquence.

4.2 Visualisation des données

La visualisation des données permet la surveillance du bon fonctionnement du système. Elle requiert donc l'utilisation des composants les plus adaptés présents dans PVSS.

4.2.1 Objectif de la visualisation

Basiquement, dans PVSS, il est possible de visualiser les données à l'aide d'un objet graphique de PVSS appelé *trend* (voir figure 4.2). Dans la configuration actuelle, la visualisation des taux se fait en fonction du temps.

Les trends de PVSS enregistrent les données qu'ils affichent durant tout le temps qu'ils sont ouverts. Ceci signifie que par défaut, ils n'affichent rien au moment où on les ouvre et ils affichent le tracé des valeurs passées jusqu'à leur fermeture.

La visualisation n'a cependant pas pour seul but de vérifier en temps réel l'évolution des paramètres représentés par les trends, mais aussi de visualiser des données préalablement enregistrées par PVSS en base de données. Le composant décrit dans la partie 4.2.2 permet de faire le lien entre la base de données et les trends.

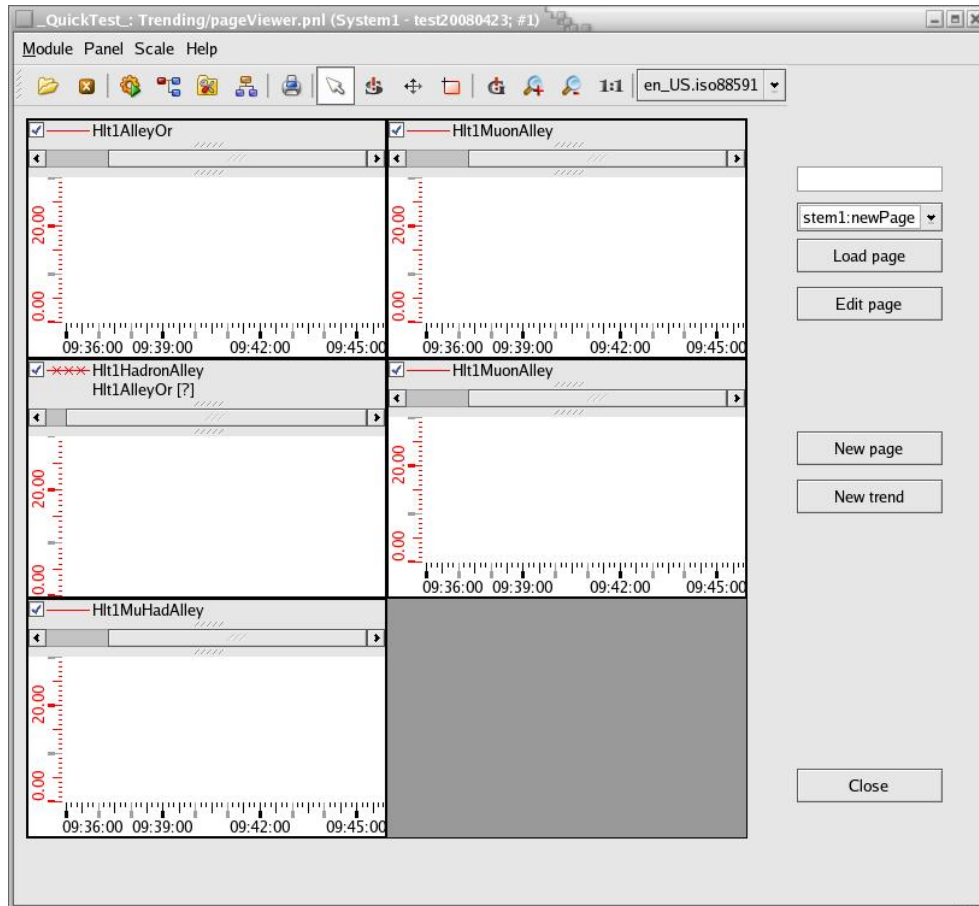


FIG. 4.2 – Aspect d'un panneau composé de trends PVSS de type XT et XY (deuxième ligne, première colonne) affichant des données de type rate. Ce panneau était un prototype permettant la visualisation de taux avant l'utilisation du composant fwTrending (voir partie 4.2.2).

4.2.2 Composant de visualisation de JCOP

Les trends de PVSS sont des objets graphiques simples dont on peut modifier la taille, l'aspect (couleur des courbes, noms des courbes, etc...). Seuls, ils ne sont cependant pas adaptés à l'usage courant au CERN. Pour faciliter leur manipulation, JCOP a créé un composant nommé *fwTrending* permettant entre autre de regrouper ces trends dans des pages et de classer ces pages dans une arborescence.

Dans *fwTrending*, un *trend* s'appelle *plot* et peut contenir plusieurs courbes à afficher

(comme les trends de PVSS). Il a été choisi de n'afficher qu'un seul taux par plot. L'aspect général d'une page et des plots est donné par la figure 4.3.

Le composant fwTrending permet également de paramétrer les plots afin de leur faire afficher des valeurs enregistrées en base de données. Lors du paramétrage d'un plot, il suffit d'indiquer que le paramètre (un DPE) doit être affiché avec les valeurs préalablement enregistrées (le trend se réfère alors à la *config* d'archivage du DPE pour récupérer et afficher les données présentes en base de données).

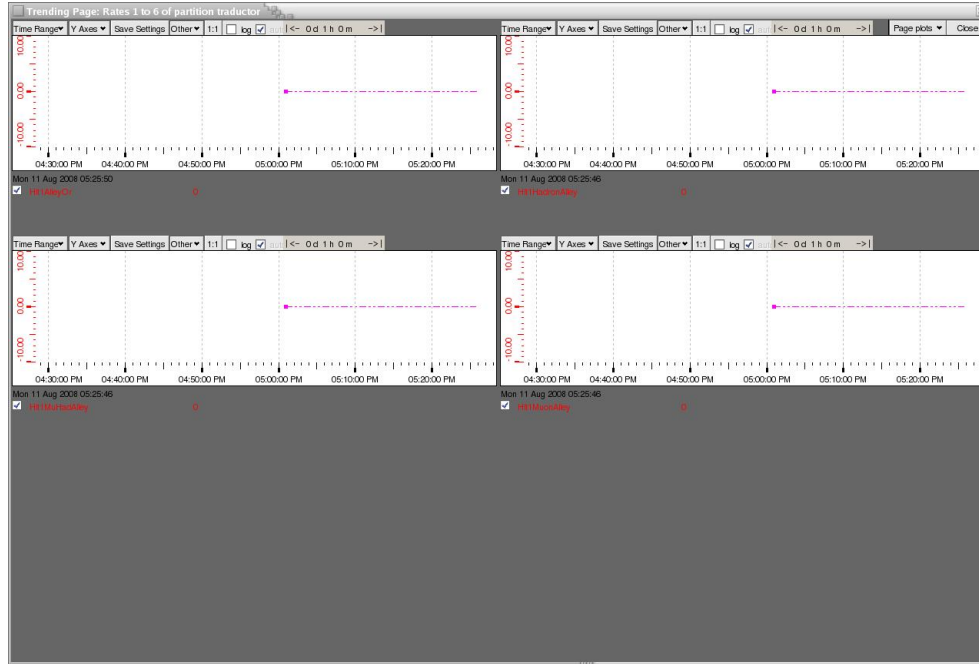


FIG. 4.3 – Aspect d'une page paramétrée pour afficher quatre plots ; chaque plot affiche un taux. Les taux représentés proviennent de services DIM déconnecté, d'où les tracés plats en pointillés violets.

Notons que contrairement aux trends basiques, fwTrending ne permet pas l'affichage en mode dit *XY* (une valeur en fonction d'une autre, voir 4.2 deuxième ligne, première colonne) mais uniquement d'une valeur en fonction du temps (mode dit *XT*). Cette fonctionnalité n'a pas encore clairement été désignée comme nécessaire, mais si elle devait être implémentée, il faudrait s'adresser à l'autorité compétente pour intervenir auprès de JCOP et demander son développement (*i.e.* Clara GASPARD pour le LHCb).

4.2.3 Automatisation et interfaçage avec fwTrending

Pour les besoins du projet, il a été nécessaire de développer une série de fonctions contenues dans une bibliothèque nommée *Trending*, pour la génération des pages (plots y compris) et de l'arborescence (voir figure 4.4 afin d'automatiser la création des interfaces de visualisation à partir des taux stockés dans PVSS).

La bibliothèque comprend deux parties : une identifiée par **Paging**, pour la création des plots et des pages à partir des taux enregistrés et une identifiée **TrendingTree** pour la génération de l'arborescence à partir du jeu de pages.

L'usage veut que les pages regroupent des trends visualisant des rates d'une même partition (même origine, donc de signification proche ou liée). Les pages peuvent, grâce à la bibliothèque, être créées et classées en conséquence.



FIG. 4.4 – Aspect d’une arborescence de (deux) pages regroupées par partition (testRateServiceByPVSS et traductor).

Génération des plots et des pages

Afin de créer les pages on a tout d’abord besoin de créer les plots correspondant aux différents taux à visualiser. La fonction `createRatingPlots()` permet de créer les plots (principalement par l’utilisation des méthodes `createPlot()` et `appendCurve()` de la bibliothèque `fwTrending`).

Les plots sont créés de manière à présenter les données de façon la plus pertinente possible. Par exemple, les plots permettent de donner une légende à une courbe. Rappelons qu’un plot ne contient qu’une courbe traçant l’historique d’un taux. La légende pour un taux est le commentaire associé à ce taux par le serveur C++ (commentaire équivalent à la description des compteurs dans le `MonRate`).

Différents paramètres, comme la couleur de la courbe, sont paramétrables mais n’ont pas donné lieu à de demandes précises de la part des futurs utilisateurs pour le moment.

La génération des pages est l’étape suivante. La bibliothèque `Trending` dispose de deux fonctions pour générer les pages à partir des plots : `makesPages()` et `makePagesByPartition()`. La première génère une liste de pages à partir d’une liste de plots et d’un nom de partition. Ce nom de partition permet de nommer les pages ; on peut omettre ce paramètre pour créer des pages sans notion de partition (la page ne sera rattachée à aucune partition dans l’arborescence par la suite). La seconde fonction permet de générer des listes de pages à partir d’une liste de plots en réalisant au préalable un tri des plots par partition d’origine à l’aide de la fonction mapping `_groupByPartition(dyn_string listOfPlots)` (la partition est déterminée à partir du DP du taux que le plot permet de visualiser).

Ces deux étapes permettent de créer un jeu de pages pour la visualisation de taux. Afin de faciliter leur accès, il faut les faire apparaître dans l’explorateur de `fwTrending` (voir figure 4.4).

Génération de l'arborescence

L'arborescence présente dans `fwTrending` pour présenter les pages utilise le composant `fwTree`. Les fonctions de type `TrendingTree`¹ de la bibliothèque `Trending` font usage de `fwTree` pour la création, le paramétrage et la suppression des nœuds de l'arborescence. L'arborescence associée à `fwTrending` est repérée par sa racine. Cette racine est un nœud nommé `fwTN_TrendTree`. Cette indication permet à `fwTrending` d'afficher uniquement les nœuds liés à l'affichage (pages, plots ou autres nœuds). À cette distinction il faut rajouter quelquepart une indication permettant de distinguer les nœuds liés aux taux. En effet, dans le projet dans lequel sera déployé le composant `RateCtrl`, il n'est pas exclu que d'autres composants utilisent `fwTrending`.

Un jeu d'identifiants a donc été défini pour distinguer les nœuds qui “nous” appartiennent. Notons que, parallèlement, un jeu de tels identifiants est également utilisé pour distinguer les plots et les pages affichant des taux des autres plots et pages présents dans le système².

La création et la suppression de l'arborescence sont les deux fonctions les plus importantes de la bibliothèque `Trending`³. Ces opérations sont réalisées par les fonctions `addPagesToTree()` et `clearTree()`.

La fonction `addPagesToTree()` permet de mettre à jour l'arborescence spécifique aux pages de taux en traitant une liste de pages à ajouter. D'une manière générale, la structure arborescente est obtenue en analysant les noms des pages. La solution adoptée consiste à créer un niveau à chaque fois que le caractère séparateur `/` est rencontré. Ainsi, le nom de page `autoRatePage/traductor/1to6` donnera l'arborescence suivante :

`autoRatePage` $\mapsto$ `autoRatePage/traductor` $\mapsto$ `autoRatePage/traductor/1to6`

Les éléments sont ici : (espace de nommage $\mapsto$ partition $\mapsto$ page)

La répétition du chemin d'accès pour nommer un nœud est indispensable pour nous permettre d'avoir, par exemple, plusieurs nœuds nommés `1to6`, comme on peut le voir sur la figure 4.4.

Il est à noter que la bibliothèque `fwTrending` ne propose pas de fonction pour créer des nœuds de type *plot* ou *page*. `FwTrending` permet uniquement d'appeler les panneaux de création de nœuds. Cette fonctionnalité n'est pas intéressante dans la mesure où l'arborescence doit être générée automatiquement et non créée manuellement par un opérateur. Afin de créer les nœuds *neutres*⁴ et les nœuds de pages, il a fallu observer, dans `PARA`, le contenu des nœuds existants pour savoir quels attributs des DPs utiliser. Il en est ressorti qu'un nœud contient, comme on pouvait s'y attendre, une référence vers son père (DPE `parent`) et une liste de référence de ses fils (DPE `children`). De plus, les DPs de type `_FwTreeNode` (le type des nœuds de la bibliothèque `FwTree`) disposent d'un attribut `device` qui doit contenir la référence vers la page à afficher, et un attribut `type` devant contenir `FwTrendingPage` (le DPT des pages de `FwTrending`) pour les nœuds de type page.

Une des conséquences de cette écriture locale de fonction touchant directement à la structure des DPs stockant les nœuds est que la bibliothèque développée doit être maintenue à

1. Commencant par l'indicatif `RateCtrl_TrendingTree`.

2. Les plots, pages et nœuds sont sauvegardés sous forme de DPs, donc visibles depuis le système dans son ensemble.

3. Pour la partie `TrendingTree` de la bibliothèque `Trending`.

4. Nœud ne référençant ni un plot ni une page.

jour de façon “périlleuse” en cas de modification des bibliothèques FwTrending ou FwTree. En effet, la façon dont FwTrending utilise la structure des DPs de type `_FwTreeNode`, mais également la structure même du DPT `_FwTreeNode` sont susceptibles de subir des modifications. De telles modifications obligent à adapter le code développé pour créer les nœuds dans la bibliothèque Trending.

Fort heureusement, dans la bibliothèque Trending, le code touchant à la structure des nœuds est lui-même très localisé, ce qui rend les modifications à apporter plus rapides et plus sûres à effectuer.

4.3 Archivage et récupération de données

L’archivage est une étape nécessaire dans le processus de surveillance. Les données reçues et stockées dans PVSS n’ont pas vocation à être finement analysées pour en extraire des conclusions scientifiques, mais doivent pourtant être sauvegardées afin de procéder *a posteriori* à l’analyse du fonctionnement des systèmes via l’étude des taux reçus du HLT.

4.3.1 But de l’archivage des données

Les données reçues sous la forme de taux donnent des indications sur le fonctionnement du HLT. Elles sont mises à la disposition des équipes de surveillance de l’expérience. Il est cependant nécessaire de procéder à des relectures de ces données pour, par exemple, comprendre l’évolution d’un dysfonctionnement.

Il est donc mis en place un système d’archivage des données (les taux stockés dans PVSS) dans une base de données Oracle, le système PVSS devant servir à la fois à l’enregistrement des données et à la relecture de celles-ci afin de les afficher dans les plots.

4.3.2 Système de gestion des archive de PVSS

Il existe pour l’expérience LHCb un composant nommé *lbArchive* permettant de faire le lien avec la base de données Oracle au travers d’un *RDB Archive Manager* (manager d’archivage en base de données relationnelles).

Rappelons qu’il existe dans PVSS un moyen de configurer manuellement les DPEs pour les archiver sur disque ou en base de données. Cette configuration manuelle passe par l’édition de la *config _archive* du DPE (via le module PARA ou à l’aide de fonctions dans un script). Il est probable que les fonctions permettant l’activation et la désactivation de l’archivage dans Oracle fassent des opérations de ce type et il serait tentant de le faire soi-même plutôt que d’utiliser le composant lbArchive. Cependant, dans un soucis de cohérence et de compatibilité du projet, c’est bien lbArchive qui est utilisé principalement à travers de la fonction de démarrage d’archivage.

On notera que l’archivage peut être paramétré, en particulier à l’aide de ce que l’on appelle le *smoothing*. Le *smoothing* (littéralement “lissage”) permet de réduire les quantités de données à archiver en les filtrant suivant différents paramètres : intervalle de temps minimum entre deux enregistrements successifs, palier de valeur minimum entre deux enregistrements successifs, etc... Les différents paramètres pouvant être combinés.

4.3.3 Bibliothèque de manipulation des archives

La bibliothèque du composant lbArchive est relativement simple d’utilisation. Son interface se compose des fonctions élémentaires pour se connecter et se déconnecter de la

base de données et pour activer et désactiver la sauvegarde d'un DPE. La bibliothèque *Archiving* a donc été développée pour fournir l'interface nécessaire l'activation et la désactivation de l'archivage des taux.

On sait, avec ce projet, quel usage doit être fait des données archivées : la valeur du taux sera tracée dans le plot approprié et le commentaire affiché comme légende pour cette courbe. Il est donc nécessaire, pour réaliser ceci, de sauvegarder la valeur du taux. La sauvegarde du commentaire et de la partition n'apporte pas d'intérêt pour deux raisons. La première est que ces deux informations n'ont pas vocation à varier dans le temps. La donnée la plus à jour dans le DP est donc de plus la plus pertinente et un archivage de leur évolution n'a pas vraiment de sens. La deuxième est qu'il n'est pas prévu d'afficher de telles évolution supposée. L'archivage n'a donc à être activé que pour le DPE de la valeur.

Cependant, les tests n'ayant pas encore été réalisés, l'activation de l'archivage pour le DP en lui même est également activé. On verra à l'usage si il est nécessaire ou non.

4.4 Interfaces de test

Afin de réaliser les tests de la façon la plus simple possible, il a été développé un ensemble d'interfaces permettant la manipulation des différentes fonctions de la bibliothèque du composant RateCtrl. De plus la bibliothèque comporte un certain nombre de paramètres qu'il conviendra de régler lors des tests (comme la fréquence d'échantillonnage des taux pour l'archivage ou les identifiants des jeux de pages). Les interfaces présentées et celles en cours de développement doivent permettre de définir ces paramètres lors des tests *online*.

4.4.1 Interface de découverte des services

Afin de tester le bon fonctionnement du système de découverte et d'enregistrement de services de la bibliothèque, une interface a été développée. Cette interface permet en outre de vérifier la bonne connection d'un service à son DP.

L'interface comprend un bouton déclenchant la recherche de services, en appelant la fonction `discoverRateServices()` de la bibliothèque RateCtrl. Tout service de type taux (diffusé par le RateService), non encore enregistré dans le système, doit donc être détecté et enregistré correctement. Pour vérifier que les services sont correctement enregistrés, on peut vérifier, à l'aide du module PARA, la présence des DPs correspondants et l'existence d'une entrée dans le DP gérant les connexions DIM liant le nom du service à son DP. La figure 4.5 montre également la possibilité de visualier les services découverts.

Le bouton indique le résultat de l'opération par un code de couleur :

- orange pour “recherche en cours”.
- jaune pour “recherche infructueuse”.
- vert pour “recherche terminée avec de nouveaux services trouvés et enregistrés”.

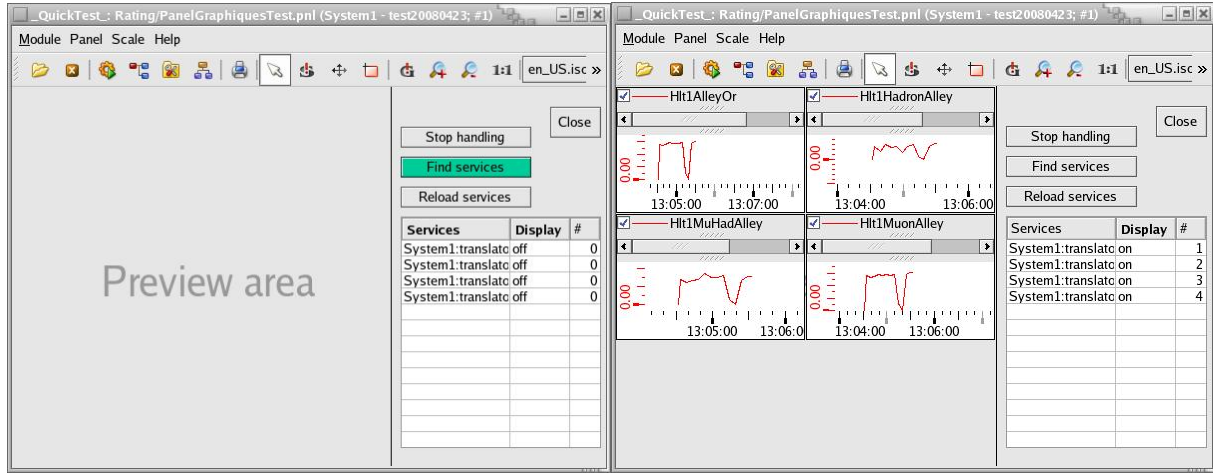


FIG. 4.5 – L’interface de recherche de services indique le résultat de la recherche par un code de couleur (gauche), donne la liste des services trouvés dans la liste et permet de visualiser les services (droite).

Ces couleurs sont prises de la palette de couleur de JCOP (respectivement **FwStateAttention2**, **FwAlarmWarmUnack** et **FwStateOKPhysics**). Les noms choisis ne correspondent pas tout à fait à la réalité représentée par les couleurs correspondantes, mais le sens s’en approche (*attention*: en cours, *alarm*: recherche non fructueuse, *OK*: recherche ayant donné des résultats).

4.4.2 Interface de création de pages

Il est fort probable qu’une fois en production, la génération des pages de visualisation soit fortement automatisée et liée à la phase de découverte des services. Cependant, pour pouvoir tester les fonctions de création des plots, des pages et de l’arborescence, une interface permettant d’effectuer individuellement ou successivement (de façon semi-automatisée, car via l’interface, l’intervention d’un opérateur est nécessaire) ces différentes étapes a été réalisée.

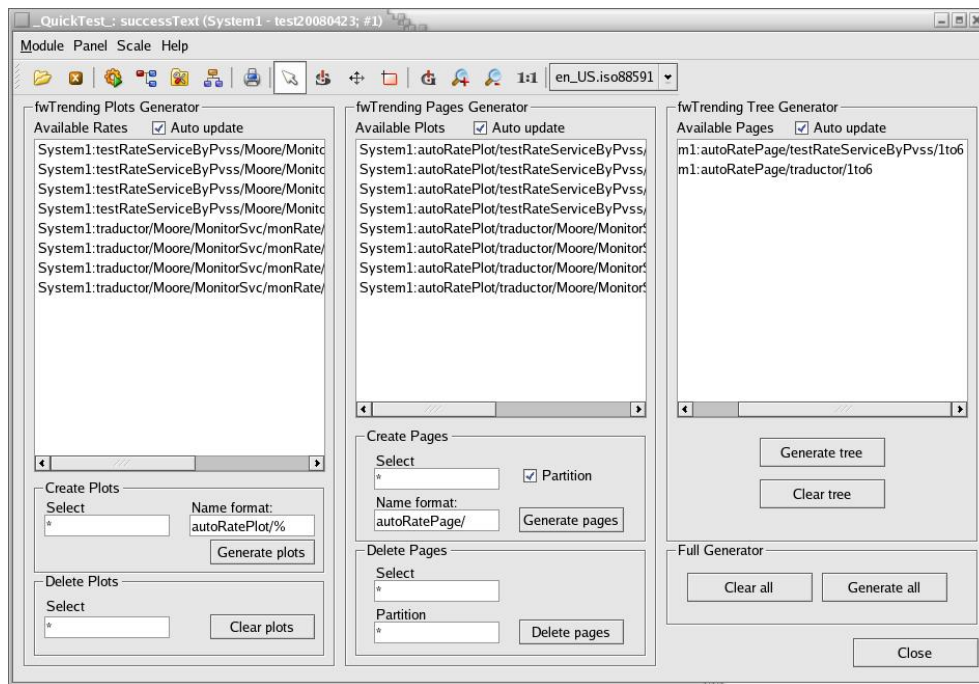


FIG. 4.6 – Interface permettant la création semi-automatisée des plots, des pages et de l’arborescence pour fwTrending.

Cette interface (voir figure 4.6) permet la création de plots à partir des taux stockés dans le système en spécifiant un format au nom de ces plots (nom du plot basé sur le nom du taux) ainsi que la création des pages à partir des plots existants, en regroupant ou non les plots par partition et enfin, la génération de l’arborescence à partir des pages existantes.

L’arborescence créée peut être visualisée dans le composant fwTrending (voir figure 4.4) et permet d’accéder aux différentes pages afin de les visualiser.

Conclusion

Le composant RateCtrl permet, à l’heure actuelle, de stocker les taux provenant du RateService sous forme de DP, et de visualiser ces taux à l’aide du composant fwTrending, en créant les plots pour afficher les taux, les pages pour regrouper ces plots et l’arborescence permettant de parcourir ces pages. La gestion correcte de l’archivage sous Oracle est encore à tester.

Chapitre 5

Résultats et discussion

5.1 Tests du RateService

Le RateService est un programme intermédiaire dans la chaîne de transmission des données et est à ce titre dépendant de très nombreux éléments, que ce soit le comportement du programme lui fournissant les MonRates à convertir, ou la compatibilité des composants Gaucho et autre qu'il utilise pour son fonctionnement.

5.1.1 Installation du composant

L'usage au CERN est d'utiliser le logiciel de gestion de versions CVS [11] pour sauvegarder les versions des différents logiciels et permettre par la suite leur installation sur les postes de production (ainsi que sur les postes de développement).

Une page web permet de visualiser les composants disponibles dont le RateService :

<http://isscvs.cern.ch/cgi-bin/cvsweb.cgi/Online/?cvsroot=lhcb>.

5.1.2 Tests fonctionnels

Les tests du composant se sont déroulés en deux phases. Une première phase tout au long du développement dans un contexte dit *offline*, c'est à dire sur mon poste de travail et sur le réseau général du CERN et une deuxième phase, *online*, sur les machines du système de contrôle du LHCb.

Comme on l'a vu précédemment, au chapitre 3, des programmes simulant le HLT ont été utilisés pour tester le RateService. Ces programmes sont les mêmes que ceux qui tournent aujourd'hui sur le HLT, et qui ont été utilisés pour les tests *online*.

Les tests réalisés, à la fois *offline* et *online* ont donc consisté à soumettre au RateService divers jeux de services MonRate et de vérifier que les taux étaient bien présents en sortie sous la forme voulue (format du service) et avec les valeurs désirées.

L'outil DID¹ de DIM a permis de vérifier la présence des services. Le programme *HltRate* de Bruno SOUZA DE PAULA a permis de son côté de vérifier le bon contenu des services, en particulier la bonne valeur des taux.

1. Distributed Information Display : outil de visualisation des services de DIM.

5.1.3 Performances

Le RateService se doit de traiter correctement tous les compteurs de tous les MonRates afin de permettre leur visualisation et leur archivage apr la suite dans PVSS. Des tests permettant de valider les performances temporelles du RateService sont en cours de réalisation. La limite du nombre de compteur est cependant connue et il y aura donc une certaine efficacité à atteindre pour pleinement valider le serveur.

5.1.4 Modifications à venir

L’aspect global du RateService dans son utilisation actuelle peut être résumé par la figure 5.1. D’après les critères actuels d’exploitation du HLT et de monitoring il semble que ça soit cette configuration qui sera utilisée pour les débuts de l’exploitation du LHCb. Comme dit précédemment, les modifications du framework ne sont pas censé avoir d’incidence sur le fonctionnement du RateService. Une modification des MonRates par exemple se traduirait par une recompilation du projet avec les nouvelles bibliothèques pour mettre à jour le serveur.

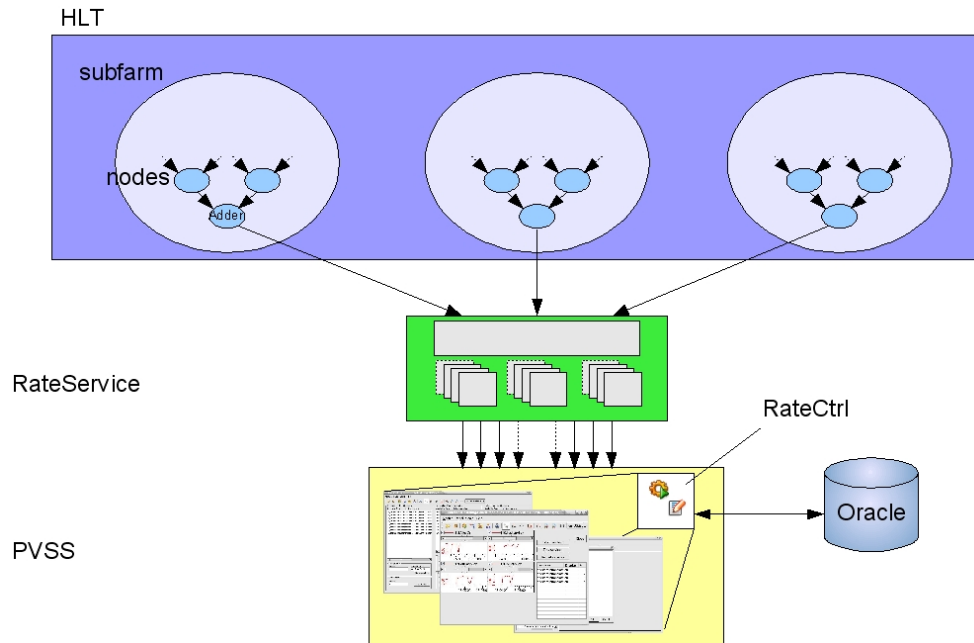


FIG. 5.1 – Les Adders du HLT diffuse leurs données vers le RateService qui les adapte pour PVSS qui gère l’archivage et la visualisation.

Les véritables modifications potentielles que pourrait subir le RateService serait un changement d’architecture interne pour passer du modèle *algorithm* au modèle *service*. Cette modification doit permettre d’intégrer au mieux le serveur dans le système de monitoring.

Des modifications sont également envisagées quant au système de recherche des services MonRate. Du code de Juan OTALORA, déjà utilisé dans les Adders et les Savers², devrait être utilisé dans le RateService pour lui permettre une plus grande flexibilité.

Il est également prévu d’enrichir les données diffusées par le RateService sous forme de taux en y ajoutant, pour chaque MonRate traité, le nombre de processus utilisant ce MonRate pour diffuser des compteurs, afin de contrôler la “santé” de la ferme.

2. Programme effectuant des sauvegardes d’histogrammes généré par le HLT.

5.2 Test du composant PVSS

Tester et valider l'interface et les bibliothèque du composant PVSS réalisé et une étape indispensable pour permettre de l'utiliser par la suite comme premier outils de visualisation et de sauvegarde, et également comme base pour de future modifications ou améliorations.

5.2.1 Déploiement du composant

La voie privilégiée pour installer un nouveau composant dans un projet PVSS au CERN est l'utilisation de l'outil *fwInstallation*. FwInstallation permet la sauvegarde d'un ensemble de panneaux, de scripts, et de managers nécessaire à l'utilisation d'un composant et, par la suite, l'installation dans un nouveau projet.

Une liste des composants PVSS pour LHCb, dont fait partie le RateCtrl, est disponible à la page suivante :

<http://lhcb-online.web.cern.ch/lhcb-online/ecs/lhcb-fw/default.htm>

Le composant tels qu'il est présenté pour installation est composé des dossiers et fichiers suivants :

-  config : contient des fichiers paramétrant l'installation.
 -  RateCtrl.config
 -  RateCtrl.init
 -  RateCtrl.postInstall
-  panels : contenant les panneaux.
 -  RateCtrl
 -  PanelGraphiquesTest.pnl
 -  automationForFwTrending.pnl
-  scripts : contenant les bibliothèques et les scripts CTRL.
 -  RateCtrl : contient les scripts CTRL (tâches de fond).
 -  autoUpdateDiscovery.ctl : script de recherche automatique de services.
 -  libs
 -  RateCtrl : contient les bibliothèques du RateCtrl.
 -  RateCtrl.ctl : fonctionnalités générales
 -  Trending.ctl : gestion de la visualisation
 -  Archiving.ctl : gestion de l'archivage
-  RateCtrl.xml : décrit le contenu du composants.
-  README.txt : un historique des modification apportées.

FIG. 5.2 – Contenu du répertoire permettant l'installation du composant RateCtrl.

5.2.2 Test des éléments

Les éléments produits sur la machine de développement, c'est à dire *offline* (par opposition à *online* signifiant dans l'environnement de production) on d'abord été tester pour vérifier qu'ils fonctionnaient correctement entre eux. Les éléments produits offline requièrent en effet d'autres composants (comme DIM, fwTrending, etc...), il est important

de vérifier que le travail réalisé est compatible avec les versions de ces composants utilisées en production.

5.2.3 Performances

Les performances du composant doivent être testées pour plusieurs facteurs : la capacité à tenir à jour une quantité importante de rates, la robustesse du système de génération du jeu de pages, la capacité à générer et lire des archives des données.

5.2.4 Validation par les utilisateurs

Le composant, tel qu’il est prévu, est censé fonctionner pour une grande partie en tâche de fond. La découverte des services, l’archivage des données et la création des pages de visualisation n’ont pas besoin d’intervention humaine une fois paramétrés.

Il faudra toutefois, au cours des derniers tests, déterminer la valeurs à donner à ces paramètres (smoothing, convention de nommage des plots et des pages, fréquence de la recherche automatisée des services, etc...).

5.3 Organisation

Tout au long de ce stage, j’ai été amené à découvrir de nouveaux outils, un “nouveau langage” (CTRL de PVSS), et le travail d’équipe à proprement parler.

J’ai en particulier suivi une semaine de formation sur PVSS (voir figure 5.3) après avoir commencé à l’apprendre par moi-même. Ayant déjà utilisé des environnements de développement équivalents dans d’autres langages (C++, Java) et étant habitué à la programmation orientée événements, la prise en main de cet outil s’est avérée relativement aisée.

<i>Tâches</i>	Avril	Mai	Juin	Juillet	Août
Prise en main et cours PVSS	x	x			
Développement RateService (serveur)					
• v1r0	x	x	x		
• v2r0			x	x	
• v2r1				x	x
Développement RateCtrl (client PVSS)					
• découverte et gestion des taux					
◦ pour RS1.0 à RS2.0 (paires de services)		x	x	x	
◦ pour RS2.1 (services structurés)				x	
• visualisation des taux					
◦ prototype basé sur les trends PVSS		x	x		
◦ système basé sur fwTrending				x	x
• archivage					
◦ documentation		x	x		
◦ prototype				x	
Tests offline	x	x	x	x	
Tests online					x
Documentation			x	x	x

FIG. 5.3 – *Tâches et planning*

La difficulté est plutôt venue de l'utilisation des composants, tant PVSS que C++, déjà présents et de leur compréhension. En effet, le contexte global du projet dans lequel j'ai réalisé mon travail demande plus que quelques mois pour être pleinement appréhendé, et je pense qu'une implication plus importante de ma part dans la compréhension de la finalité du projet m'aurait permis d'être plus efficace.

En revanche, je pense m'être assez bien impliqué dans le travail en lui-même en allant fréquemment consulter mes collègues pour des questions techniques. Ma participation aux différentes réunions de suivi de projet m'a permis, je pense, d'améliorer mes prestations orales, en particulier en Anglais.

Du point de vue de l'organisation du développement en lui-même, l'évolution par phases successives du RateService et du RateCtrl m'a permis de tester étape par étape le bon fonctionnement de mes logiciels, ce qui a probablement contribué au succès de la partie qui m'était attribuée.

Le mérite en revient je pense à Eric VAN HERWIJNEN qui m'a ainsi permis de découvrir au bon rythme la complexité de mon "environnement technique" de travail.

Conclusion

Les deux projets réalisés sont destinés à être utilisés, modifiés et améliorés au cours du temps. Le structure et leur fonctionnement sont également sujets à de possibles modifications.

Il faudra donc veiller à rendre ces projets les plus compatibles possibles avec les standards du LHCb lors des dernières semaines de travail. Ceci passera certainement par la modification du fonctionnement du RateService pour le transformer d'*algorithme* en *service* et de réaliser les premières release sur CVS. Il faudra également s'assurer, pour le composant PVSS, que sa distribution via le fwInstallation fonctionne pour permettre son installation sur les différents projet pouvant nécessiter son utilisation.

Conclusion

Ce projet avait pour but de permettre l’affichage et l’archivage des taux de sélection du HLT dans le système de supervision PVSS du LHCb. Au vu des derniers tests réalisés dans les semaines précédant la mise en fonctionnement de l’expérience, on constate que la récupération des taux dans PVSS et leur visualisation est opérationnelle. Il est en revanche malheureusement trop tôt pour être sûr que l’archivage des données sera possible à temps.

Les deux parties du projet sont installées dans la salle de contrôle et redistribuables selon les critères du groupe Online (CVS pour le RateService et fwInstallation pour le RateCtrl).

Différentes améliorations seront à apporter au RateService en terme d’intégration au système de monitoring. En particulier, la publication du nombre de processus utilisant un même MonRate devra être effectuée par le RateService, sous la forme d’un service de taux, afin que PVSS puisse l’afficher à l’aide du composant RateCtrl.

Le RateCtrl devra, lui-aussi, subir des modifications notamment sur le plan de l’archivage des taux ou encore concernant l’apparence des visualisations dans fwTrending.

De plus, une fois tous les tests réalisés et les composants validés, une application stricte des conventions de nommage pourra être appliquée afin de rendre le code le plus maintenable possible.

Sur un plan personnel, ce stage m’a permis d’avoir une vision précise des problèmes liés à la conduite de projet en participant à plusieurs réunions de mon groupe de travail, notamment pour présenter l’avancée de mon travail. J’ai également pu constater les difficultés et le challenge que représentent la mise sur pied de systèmes informatiques tels que le système de supervision du LHCb. Enfin le contexte très international du travail au CERN m’a permis de prendre conscience de mes capacités et mes lacunes dans le domaine du travail d’équipe.

Bibliographie

- [1] LHCb collaboration. *Technical Proposal, A Large Hadron Collider Beauty Experiment for Precision Measurements of CP Violation and Rare Decays*. CERN/LHCC 98-4.
- [2] D. Wiedner. *Early physics with the LHCb detector*. CERN. 2008.
http://lhcb-doc.web.cern.ch/lhcb-doc/presentations/ConferencePosters-/Postscript/2008/dirk_wiedner_perugia_poster.pdf.
- [3] H. Ruiz. *HLT homepage*. CERN. 2008.
<http://lhcb-trig.web.cern.ch/lhcb-trig/HLT/Default.htm>.
- [4] C. Gaspar. *PVSS Introduction for Newcomers*. CERN.
<http://itcobe.web.cern.ch/itcobe/Services/Pvss/Documents/PvssIntro.pdf>.
- [5] *PVSS Homepage*. ETM. 2008.
http://www.etm.at/index_e.asp?id=2&m0id=6.
- [6] W. Salter. *Detector controls for LHC experiments*. CERN. 2008.
<http://cerncourier.com/cws/article/cern/33360>.
- [7] C. Gaspar et coll. *Distributed Information Management System*. CERN. 2006.
<http://dim.web.cern.ch/dim/>.
- [8] LHCb Online. *The Gaucho Homepage*.
<http://lhcb-online.web.cern.ch/lhcb-online/ecs/Gaucho/default.htm>
- [9] *The Gaudi Project*.
<http://proj-gaudi.web.cern.ch/proj-gaudi/>
- [10] C. Arnault. *Configuration Management Tool*. 2006.
<http://www.cmtsite.org/>
- [11] IT Department. *CERN Central CVS Service*.
<http://cvs.web.cern.ch/cvs/>

ANNEXES

Annexe A

Documentation du serveur C++ RateService

Cette documentation a été réalisée avec Doxygen et peut être consultée dans le répertoire `/doc/doxygen/html` du composant *RateService*.

Le composant est, lui, disponible à l'adresse suivante :

<http://isscvs.cern.ch/cgi-bin/cvsweb.cgi/Online/?cvsroot=lhcb>.

DimInfoMonRate Class Reference

```
#include <DimInfoMonRate.h>
```

[List of all members.](#)

Public Member Functions

```
DimInfoMonRate (std::string monRateSvcName, int refreshTime, std::string source)
virtual ~DimInfoMonRate ()
void createMonRate ()
void loadMonRate ()
virtual void infoHandler ()
MonRate* getMonRate ()
bool isMonRateCreated ()
void setMsgService (IMessageSvc *ims)
std::string getServiceName ()
```

Private Member Functions

```
void extractData ()
int lookForNewRates ()
std::string makeServiceNameHeader ()
```

Private Attributes

```
bool m_constructorDone
bool m_hasData
std::string m_name
bool m_monRateCreated
std::string m_monRateServiceName
MonRate* m_monRate
int m_currentCycleNumber
IMessageSvc* m_msgSvc
int m_stringSize
std::string m_sourceName
ExtractorMap m_extractorMap
std::string serviceNameHeader
```

Detailed Description

A DimInfo based class subscribing to a MonRate service. Uses [RateExtractor](#) objects to publish services included in the MonRate.

Author:

Jean-Francois Menou

Constructor & Destructor Documentation

```
DimInfoMonRate::DimInfoMonRate ( std::string monRateSvcName,
int refreshTime,
```

```
std::string DimInfoMonRate::getServiceName() [inline]
```

Get the name of the DIM service handled by the instance.

Returns:
The service name.

```
void DimInfoMonRate::extractData () [private]
```

Method processing the MonRate object : call the [extractData\(\)](#) method for each rate contained in the MonRate.

```
int DimInfoMonRate::lookForNewRates () [private]
```

Looks for new counters (not beeing converted into rate by a [RateExtractor](#) object yet).

Returns:
The number of new counters found.

```
std::string DimInfoMonRate::makeServiceNameHeader ( ) [private]
```

Create a header string (stored in serviceNameHeader) based on algorithm name and MonRate service name.

Returns:
The header.

Member Data Documentation

```
bool DimInfoMonRate::m_constructorDone [private]
```

Dummy flag supposed to show if constructor is done, to prevent infoHandler to be executed between service subscription and constructor's end.

```
bool DimInfoMonRate::m_hasData [private]
```

Flag set to true if the MonRate has data.

Deprecated:

```
std::string DimInfoMonRate::m_name [private]
```

Name of the MonRate.

Deprecated:

```
bool DimInfoMonRate::m_monRateCreated [private]
```

Flag indicating if the MonRate object has correctly been initialized in the [createMonRate\(\)](#) method.

```
std::string source)
```

Constructor.

Parameters:

monRateSvcName Name of the DIM service to subscribe and handle.
refreshTime DimInfo refreshTime parameter.
source Name of the algorithm creating this instance.

```
DimInfoMonRate::~DimInfoMonRate ( ) [virtual]
```

Member Function Documentation

```
void DimInfoMonRate::createMonRate ( )
```

Initialize the MonRate from the subscribed DIM service.

```
void DimInfoMonRate::loadMonRate ( )
```

Updates the MonRate content from the subscribed DIM service.

```
void DimInfoMonRate::infoHandler ( ) [virtual]
```

Callback on MonRate service changes.

```
MonRate* DimInfoMonRate::getMonRate ( ) [inline]
```

Get a pointer on the MonRate object receiving data from the DIM service.

Returns:
The pointer on the MonRate.

```
bool DimInfoMonRate::isMonRateCreated ( ) [inline]
```

Checks if the MonRate ha been created.

Returns:
True if the MonRate has been created.

```
void DimInfoMonRate::setMsgService ( IMessageSvc * ims ) [inline]
```

Set the message service.

Parameters:
ims The message service interface to use.

```
std::string DimInfoMonRate::getServiceName() [inline]
```

Get the name of the DIM service handled by the instance.

Returns:
The service name.

```
void DimInfoMonRate::extractData () [private]
```

Method processing the MonRate object : call the [extractData\(\)](#) method for each rate contained in the MonRate.

```
int DimInfoMonRate::lookForNewRates () [private]
```

Looks for new counters (not beeing converted into rate by a [RateExtractor](#) object yet).

Returns:
The number of new counters found.

```
std::string DimInfoMonRate::makeServiceNameHeader ( ) [private]
```

Create a header string (stored in serviceNameHeader) based on algorithm name and MonRate service name.

Returns:
The header.

Member Data Documentation

```
bool DimInfoMonRate::m_constructorDone [private]
```

Dummy flag supposed to show if constructor is done, to prevent infoHandler to be executed between service subscription and constructor's end.

```
bool DimInfoMonRate::m_hasData [private]
```

Flag set to true if the MonRate has data.

Deprecated:

```
std::string DimInfoMonRate::m_name [private]
```

Name of the MonRate.

Deprecated:

```
bool DimInfoMonRate::m_monRateCreated [private]
```

Flag indicating if the MonRate object has correctly been initialized in the [createMonRate\(\)](#) method.

```
std::string DimInfoMonRate::m_monRateServiceName [private]
```

The name of the MonRate service that is processed by the instance

```
MonRate* DimInfoMonRate::m_monRate [private]
```

Pointer on the MonObject actually owned by the DimInfoMonObject object

```
int DimInfoMonRate::m_currentCycleNumber [private]
```

Indication on the last processed cycle. updated in extractDate() method.

```
IMessageSvc* DimInfoMonRate::m_msgSvc [private]
```

Pointer to the message service...

```
int DimInfoMonRate::m_stringSize [private]
```

Length of the last read DIM buffer for the subscribed service.

```
std::string DimInfoMonRate::m_sourceName [private]
```

Name of the algorithm owning this object...

```
ExtractorMap DimInfoMonRate::m_extractorMap [private]
```

List of object in charge of extracting and republishing rates from MonRate.

```
std::string DimInfoMonRate::serviceNameHeader [private]
```

This string is to be used by [RateExtractor](#) object to publish their services. in a coherent way.

The documentation for this class was generated from the following files:

- [/afs/cern.ch/user/fj/menou/cmtuser/Online_v4r10/Online/RateService/v2r1/src/ DimInfoMonRate.h](#)
- [/afs/cern.ch/user/fj/menou/cmtuser/Online_v4r10/Online/RateService/v2r1/src/ DimInfoMonRate.cpp](#)

Generated on Wed Jul 23 16:41:55 2008 for RateService_v2r1 by  1.5.1

RateExtractor Class Reference

```
#include <RateExtractor.h>
```

[List of all members.](#)

Public Member Functions

	RateExtractor (std::string <i>rateId</i> , MonRate * <i>pMonRate</i>)
	~RateExtractor ()
double &	getRateValue ()
char *	getRateComment ()
std::string	getRateId ()
bool	extractData (longlong <i>time</i>)
void	publishServices (std::string <i>nameHeader</i>)

Private Member Functions

std::string	makeServiceName (std::string <i>nameHeader</i>)
void	reshapeComment ()
int	sentDataSize ()

Private Attributes

DimService *	m_structuredService
std::string	m_rateId
MonRate *	m_pMonRate
RateExtractorData	m_data
int	m_commentSize
double	m_counterOldValue
double	m_counterNewValue
longlong	m_oldTime
longlong	m_newTime

Classes

struct	RateExtractorData
--------	-----------------------------------

Detailed Description

Class extracting rate from a given counter in a MonRate object. Publishes a pair of DIM services (rate value and rate comment).

Author:

Jean-Francois Menou

Constructor & Destructor Documentation

```
RateExtractor::RateExtractor (std::string rateId,  
                             MonRate * pMonRate  
                             )
```

nameHeader: Header of the complete service name (Header+counter name).

```
std::string RateExtractor::makeServiceName (std::string nameHeader ) [private]
```

Trivial method to create the name of the services.

Parameters:

nameHeader: Header to add before the counter name.

Returns:

The service name.

```
void RateExtractor::reshapeComment ( ) [private]
```

Method forming the comment.

Deprecated:

Comment does not contain label to recognize rate services.

```
int RateExtractor::sentDataSize ( ) [private]
```

Calculate the size of the data (value+comment) to send.

Returns:

Size of the data to send.

Member Data Documentation

```
DimService* RateExtractor::m_structuredService [private]
```

Pointer to the published structured.

```
std::string RateExtractor::m_rateId [private]
```

Identifier of the converted counter in the given MonRate object.

```
MonRate* RateExtractor::m_pMonRate [private]
```

Pointer to the MonRate object containing the counter to transform.

```
RateExtractorData RateExtractor::m_data [private]
```

Data extracted of the MonRate counter and stored to be sent by DIM.

```
int RateExtractor::m_commentSize [private]
```

Size of the comment including null character. Max size defined by s_maxRateCommentSize

Constructor

Parameters:

rateId Identifier of the processed counter in the MonRate.
pMonRate Pointer to the MonRate owning the processed counter.

```
RateExtractor::~RateExtractor ( )
```

Destructor

Member Function Documentation

```
double& RateExtractor::getRateValue ( ) [inline]
```

Get a reference on the value container.

Returns:

Reference on m_data.value.

```
char* RateExtractor::getRateComment ( ) [inline]
```

Get the current comment.

Returns:

A pointer on the current comment.

```
std::string RateExtractor::getRateId ( ) [inline]
```

Get the processed counter name.

Returns:

The processed counter name.

```
bool RateExtractor::extractData ( longlong time )
```

Extraction method, based on time of the new event.

Parameters:

time: Time of the last event in the processed cycle.

Returns:

True if no error.

```
void RateExtractor::publishServices (std::string nameHeader )
```

Method publishing the structured service.

Parameters:

```
double RateExtractor::m_counterOldValue [private]
```

Counter values got from MonRate, used to make counter's value comparisons

```
double RateExtractor::m_counterNewValue [private]
```

```
longlong RateExtractor::m_oldTime [private]
```

Time values got from MonRate, used to make time comparisons.

```
longlong RateExtractor::m_newTime [private]
```

The documentation for this class was generated from the following files:

- /afs/cern.ch/user/fjmenou/cmtuser/Online_v4r10/Online/RateService/v2r1/src/ [RateExtractor.h](#)
- /afs/cern.ch/user/fjmenou/cmtuser/Online_v4r10/Online/RateService/v2r1/src/ [RateExtractor.cpp](#)

Generated on Wed Jul 23 16:41:55 2008 for RateService_v2r1 by  1.5.1

RateService Class Reference

```
#include <RateService.h>
```

[List of all members.](#)

Public Member Functions

	RateService (const std::string &name, ISvcLocator *pSvcLocator) <i>Constructor of this form must be provided.</i>
StatusCode	initialize () <i>Initialize mandatory member functions of any algorithm.</i>
StatusCode	execute () <i>Execute mandatory member functions of any algorithm.</i>
StatusCode	finalize () <i>Finalize mandatory member functions of any algorithm.</i>

Private Member Functions

bool	isHandledService (std::string serviceName)
StatusCode	findServices ()

Private Attributes

std::string	m_UTGID
std::string	m_dimName
	???
std::string	m_monRateServiceName
bool	m_found
std::vector< DimInfoMonRate * >	m_dimInfoMonRate
time_t	time_old <i>not used anymore.</i>
time_t	time_new
int	sleepTime <i>Time to sleep between.</i>

Detailed Description

RateService is an Algorithm subscribing to MonRate services and publishing structured services to PVSS.

Author:

Jean-Francois Menou

Constructor & Destructor Documentation

```
RateService::RateService ( const std::string & name,
                          ISvcLocator * pSvcLocator
                          )
```

Constructor of this form must be provided.

Member Function Documentation

```
StatusCode RateService::initialize ( )
```

Initialize mandatory member functions of any algorithm.

```
StatusCode RateService::execute ( )
```

Execute mandatory member functions of any algorithm.

```
StatusCode RateService::finalize ( )
```

Finalize mandatory member functions of any algorithm.

```
bool RateService::isHandledService ( std::string serviceName ) [private]
```

Checks if a MonRate service is already being processed by a DimInFoMonRate.

Parameters:

serviceName The DIM service name to be checked.

Returns:

True if the service is already handled.

```
StatusCode RateService::findServices ( ) [private]
```

Discovers MonRate services

Returns:

A StatusCode.

Member Data Documentation

```
std::string RateService::m_UTGID [private]
```

utgid given to this algorithm.

```
std::string RateService::m_dimName [private]
```

???

```
std::string RateService::m_monRateServiceName [private]
```

name of the service publishing the MonRate object to be subscribed

```
bool RateService::m_found [private]
```

if the service has been found then don't ; if the service has been found then don't

```
std::vector< DimInfoMonRate * > RateService::m_dimInfoMonRate [private]
```

Pointers to the DimInfoMonObject handling the MonRate services.

```
time_t RateService::time_old [private]
```

not used anymore.

```
time_t RateService::time_new [private]
```

```
int RateService::sleepTime [private]
```

Time to sleep between.

The documentation for this class was generated from the following files:

- /afs/cern.ch/user/f/jmenou/cmtuser/Online_v4r10/Online/RateService/v2r1/src/ [RateService.h](#)
- /afs/cern.ch/user/f/jmenou/cmtuser/Online_v4r10/Online/RateService/v2r1/src/ [RateService.cpp](#)

Annexe B

Documentation du client PVSS RateCtrl

Cette documentation a été réalisée avec Doxygen et peut être consultée dans le répertoire `/help/en_US.iso88591/RateCtrl/scripts/libs/RateCtrl` du composant *RateCtrl*.

Le composant est, lui, disponible à l'adresse suivante :

<http://lhcb-online.web.cern.ch/lhcb-online/ecs/lhcb-fw/default.htm>.

RateCtrl.ctl File Reference

Detailed Description

This library contains utility functions for rate services acquisition.

Creation Date
01/08/2008

Modification History
01/08/2008: nothing

Constraints
None

Usage
Public

PVSS managers
VISION, CTRL

Author:
Jean-Francois Menou

Function Documentation

string _RateCtrl_extractPartition(string *serviceName*)

Determines the partition name from the given service name

Usage Internal

Parameters:
serviceName The name of the service to process.

Returns:
True in case of success.

bool _RateCtrl_setPartition(string *serviceName*)

Set the partition DPE of the given service name according to the partition extraction rule define in `extractPartition()`.

Usage Internal

Parameters:
serviceName The name of the service to be set.

Returns:
True in case of success.

bool RateCtrl_createParamDPTYPE()

Creates DPT for global rating parameter. Also fill a DP with default values.

Usage Public

Returns:
True in case of success.

bool RateCtrl_createRateDPTYPE()

This document is created with trial version of HTML2PDF Pilot 2.15.78.

Returns:
True in case of success.

bool RateCtrl_getPartitionDPE(string *serviceName*)

Get complete path to partition for a given service name.

Usage Public

Parameters:
serviceName The rate to get the partition path from.

Returns:
True in case of success.

dyn_string RateCtrl_getRateDPList(string *pattern* = "*")

Get the list of the DPs of type RATE_DPT. This list contains all DPS of type RATE_DPT, NOT only properly handled services.

Usage Public

Parameters:
pattern A pattern to select a part of the DPs.

Returns:
A list of DPs of type RATE_DPT matching the pattern.

**bool RateCtrl_getValue(string *serviceName*,
float & *value*
)**

Get value for a given service name.

Usage Public

Parameters:
serviceName The rate to get the value from.
value The value is returned here.

Returns:
True in case of success.

string RateCtrl_getValueDPE(string *serviceName*)

Get complete path to value for a given service name.

Usage Public

Parameters:
serviceName The rate to get the value path from.

Returns:
True in case of success.

bool RateCtrl_handleService(string *serviceName*)

Stores the rate service description in DP and subscribes to the service. (this could be optimized by using multiple subscription => handleServices).

Usage Public

Parameters:
serviceName The name of the service as found by DIM.

Returns:
True in case of success

Usage Public

Returns:
True in case of success.

int RateCtrl_discoverRateServices()

Discovers services according to service selection criteria.

Usage Public

Returns:
The number of discovered and correctly handled services.

bool RateCtrl_existRateServiceDP(string *dpName*)

Checks whether a RATE_DPT DP exist with this name.

Usage Public

Parameters:
dpName The name of the DP to check.

Returns:
True if the DP exists.

**bool RateCtrl_getComment(string *serviceName*,
string & *comment*
)**

Get value for a given service name.

Usage Public

Parameters:
serviceName The rate to get the comment from.
comment The comment is returned here.

Returns:
True in case of success.

string RateCtrl_getCommentDPE(string *serviceName*)

Get complete path to comment for a given service name.

Usage Public

Parameters:
serviceName the rate to get the comment path from.

Returns:
True in case of success.

**bool RateCtrl_getPartition(string *serviceName*,
string & *partition*
)**

Get partition for a given service name.

Usage Public

Parameters:
serviceName The rate to get the partition from.
partition The partition is returned here.

This document is created with trial version of HTML2PDF Pilot 2.15.78.

bool RateCtrl_isHandledService(string *serviceName*)

Determines whether a service is properly handled or not

Usage Public

Parameters:
serviceName The name of the service to check.

Returns:
True if the service is properly handled.

**bool RateCtrl_isRateService(string *serviceName*,
string & *format*
)**

Determines if a DIM service is a rate service.

Usage Public

Parameters:
serviceName The name of the service to process.
format The format of the DIM service to compare with RATE_SERVICE_DIM_FORMAT.

Returns:
True if the service is a rate service.

RateCtrl_setDiscoveryPattern (string *pattern*)

Set the service name pattern used to discover services.

Usage Public

bool RateCtrl_unHandleService(string *serviceName*)

Stops service subscription, archiving and removes DP

Usage Public

Parameters:
serviceName The service to be stopped to be handled

Returns:
True in case of success.

string removeSystemName(string *dp*)

Removes the system name for a given DP.

Usage Public

Parameters:
dp The DP to be processed.

Returns:
The DP name without the system name.

Trending.ctl File Reference

Detailed Description

This library contains utility functions for rate services acquisition.

Creation Date
01/08/2008

Modification History
01/08/2008: nothing

Constraints
None

Usage
Public

PVSS managers
VISION, CTRL

Author:
Jean-Francois Menou

Function Documentation

```
int _RateCtrl_Paging_column( int n )
```

Get the column number for the 'n'th plot in a page.

Usage Internal

Parameters:
n The index of the plot in the page.

Returns:
The column index ; -1 in case of error.

```
bool _RateCtrl_Paging_getFwTrending_PagePlots( string pageDp,
                                              dyn_string & plotList )
```

Get the list of the plots of a page.

Usage Internal

Parameters:
pageDp The dp of the page to look in.
plotList The list of plots is returned here.

Returns:
True for success.

```
bool _RateCtrl_Paging_getFwTrending_PageTitle( string pageDp,
                                              string & pageTitle )
```

Get the title of a given page

Usage Internal

Parameters:
pageDp The dp of the page.
pageTitle The page title is returned here.

Returns:
True for success.

```
string _RateCtrl_Paging_getRateDPFromPlotDP( string plotDP )
```

Get the DP of the Rate associated to plot.

Usage Internal

Parameters:
plotDP The DP of the plot to look in.

Returns:
The DP of the Rate or "" in case of error.

Returns:
True for success.

```
bool _RateCtrl_Paging_setFwTrending_PagePlots( string pageDp,
                                              dyn_string plotList )
```

Set the list of the plots of a page.

Usage Internal

Parameters:
pageDp The dp of the page to be modified.
plotList The list of plots to be set.

Returns:
True for success.

```
bool _RateCtrl_Paging_setFwTrending_PageTitle( string pageDp,
                                              string pageTitle )
```

Set the title of a given page

Usage Internal

Parameters:
pageDp The dp of the page to be modified.
pageTitle The page title to be set.

Returns:
True for success.

```
bool _RateCtrl_Trending_IsRatingPage( string page )
```

Checks if a page is displaying rating plots according to its name (which should begin with PAGE_IDENTIFIER).

Usage Public

Parameters:
plot Name of plot (should be a DP name).

Returns:
true (it is a rating page) or false

```
bool _RateCtrl_Trending_IsRatingPlot( string plot )
```

Checks if a plot is displaying a rate according to its name (which should begin with PLOT_IDENTIFIER).

Usage Public

Parameters:
plot Name of plot (should be a DP name).

Returns:
true (it is a rating plot) or false

```
bool _RateCtrl_TrendingTree_addPageToTree( string pageName )
```

Adds a page to the TrendingTree.

Usage Internal

Parameters:
pageName The page to be added to the tree.

Returns:
True in case of success.

```
bool _RateCtrl_TrendingTree_createNode( string nodeName,
                                       string parent,
                                       string & givenName )
```

Creates a node in the TrendingTree.

Usage Internal

Parameters:
nodeName The name wished for this node.
parent The name of the parent for this node.
givenName The name given to the node by the `fwTree_createNode()` function is returned here. Should equals *nodeName* if *nodeName* is well formed.

```
mapping _RateCtrl_Paging_groupByPartition( dyn_string listOfPlots )
```

Groups plots depending on the partition of the Rate.

Usage Internal

Parameters:
listOfPlots The list containing the plots to be grouped.

Returns:
A map associating a partition (the key, string) to a list of plots for this partition (the value, *dyn_string*).

```
string _RateCtrl_Paging_makePageName( int i,
                                       string partition,
                                       string pattern = "" )
```

Makes a page name using *i* partition and pattern and global string *pageNamePart1*. Pages contain plots. Plots are linked to 1 rate. Rates have partition information. Pages are built from a set of plots. Plots can be selected with their rate's partition information and/or a ptern. The index of the first plot in this set is used to determined the page name. Adds an identifier at the begin of the name to recognize it from other pages in the system.

Usage Internal

Parameters:
i The index of the first plot to be added to the page.
partition The name of the partition to which added plots belongs
pattern The pattern the plots have been selected with.

Returns:
The name of the page.

```
string _RateCtrl_Paging_makePageTitle( int i,
                                       string partition,
                                       string pattern )
```

Makes a page title using *i* partition and pattern and global string *pageNamePart1*. Pages contain plots. Plots are linked to 1 rate. Rates have partition information. Pages are built from a set of plots. Plots can be selected with their rate's partition information and/or a ptern. The index of the first plot in this set is used to determined the page title.

Usage Internal

Parameters:
i The index of the first plot to be added to the page.
partition The name of the partition to which added plots belongs
pattern The pattern the plots have been selected with.

Returns:
The title of the page.

```
string _RateCtrl_Paging_makePlotName( string rateName )
```

Makes a plot name using *rateName* and the two global strings *plotNamePart1* and *plotNamePart2*. Checks if *rateName* contains a system name, if it does the system name is removed. Adds an identifier at the begin of the name to recognize it from other plots in the system.

Usage Internal

Parameters:
rateName Name of a rate (should be a DP name).

Returns:
A string with this format: *plotNamePart1* + *rateName* + *plotNamePart2* (where *rateName* has lost its system name if it had one).

```
int _RateCtrl_Paging_row( int n )
```

Get the row number for the 'n'th plot in a page.

Usage Internal

Parameters:
n The index of the plot in the page.

Returns:
The column index ; -1 in case of error.

```
bool _RateCtrl_Paging_setCurveLegend( string plotName,
                                       string legend )
```

Set the legend for the curve of a plot (plots having only one curve in this context).

Usage Internal

Parameters:
plotName The DP of the plot to be modified.

Returns:
True for success.

```
bool _RateCtrl_Paging_setFwTrending_PagePlots( string pageDp,
                                              dyn_string plotList )
```

Set the list of the plots of a page.

Usage Internal

Parameters:
pageDp The dp of the page to be modified.
plotList The list of plots to be set.

Returns:
True for success.

```
bool _RateCtrl_Paging_setFwTrending_PageTitle( string pageDp,
                                              string pageTitle )
```

Set the title of a given page

Usage Internal

Parameters:
pageDp The dp of the page to be modified.
pageTitle The page title to be set.

Returns:
True for success.

```
bool _RateCtrl_Trending_IsRatingPage( string page )
```

Checks if a page is displaying rating plots according to its name (which should begin with PAGE_IDENTIFIER).

Usage Public

Parameters:
plot Name of plot (should be a DP name).

Returns:
true (it is a rating page) or false

```
bool _RateCtrl_Trending_IsRatingPlot( string plot )
```

Checks if a plot is displaying a rate according to its name (which should begin with PLOT_IDENTIFIER).

Usage Public

Parameters:
plot Name of plot (should be a DP name).

Returns:
true (it is a rating plot) or false

```
bool _RateCtrl_TrendingTree_addPageToTree( string pageName )
```

Adds a page to the TrendingTree.

Usage Internal

Parameters:
pageName The page to be added to the tree.

Returns:
True in case of success.

```
bool _RateCtrl_TrendingTree_createNode( string nodeName,
                                       string parent,
                                       string & givenName )
```

Creates a node in the TrendingTree.

Usage Internal

Parameters:
nodeName The name wished for this node.
parent The name of the parent for this node.
givenName The name given to the node by the `fwTree_createNode()` function is returned here. Should equals *nodeName* if *nodeName* is well formed.

Returns:
True in case of success.

```
bool _RateCtrl_TrendingTree_IsRatingNode( string node )
```

Checks if a node has rating stuff or not.

Usage Internal

Parameters:
node The node to test.

Returns:
true if has rating stuff.

```
bool _RateCtrl_TrendingTree_IsTrendingNode( string node,
                                           dyn_string & exInfo,
                                           dyn_string & path )
```

Determines if node belongs to the TrendingNode.

Usage Internal

Parameters:
node The node to test.
exInfo An exception container.
path A string array to track the path node to the root node of its tree. Not important. To be removed.

Returns:
True if node belongs to the TrendingTree.

```
dyn_string _RateCtrl_TrendingTree_makeNodeList( string pageName )
```

Creates a list of string form a page name. Page names are slash-separated. Each part of a page name represent a level in the tree. If *pageName* is "xxx/yyyyzzz", the return list will be {"xxx", "xxx/yyyy", "xxx/yyyyzzz"}.

Usage Internal

Parameters:
pageName The page name to be parsed to create the list

Returns:
The list of levels.

```
int _RateCtrl_TrendingTree_removeArtefactNodes( bool confirm = false )
```

Removes node beginning with 'X' and belonging to the Trending tree.

Usage Internal

Parameters:
confirm Pops up a confirmation dialog is set to true.

Returns:
The number of removed nodes.

```
bool _RateCtrl_TrendingTree_removeNode( string node )
```

Removes a given node.

Usage Internal

Parameters:
node The node to be removed.

Returns:
True in case of success.

```
dyn_string _RateCtrl_TrendingTree_selectRatingNodes( dyn_string nodes,
                                                    dyn_string exInfo )
```

Selects the nodes having rating stuff.

Usage Internal

Parameters:
nodes A set of nodes.
exInfo An exception container.

Returns:
The list of nodes having rating stuff.

```
dyn_string _RateCtrl_TrendingTree_selectTrendingNodes( dyn_string nodes,
dyn_string exinfo
)
```

Selects the nodes beeing attached to the TrendingTree.

Usage Internal

Parameters:
nodes A set of nodes.
exinfo An exception container.

Returns:
 The list of nodes belonging to nodes and beeing attached to the TrendingTree.

```
bool _RateCtrl_TrendingTree_setNode( string nodeName )
```

Sets DPEs for a pure node (not leaf).

Usage Internal

Parameters:
nodeName The node to be set.

Returns:
 True in case of success.

```
bool _RateCtrl_TrendingTree_setPageNode( string nodeName,
string pageName
)
```

Sets DPEs for a page node (a leaf in the tree)

Usage Internal

Parameters:
nodeName The node to be set.
pageName The parameter to be set as NodeDevice.

Returns:
 True in case of success.

```
clear( dyn_string & dyn )
```

Dummy function calling dynClear().

Usage Public

Parameters:
dyn The array to clear.

```
bool confirmDPDelete( string DPTYPE,
dyn_string DPps
)
```

Pops up a dialog box asking for confirmation. This function does not delete anything.

Usage Public

Parameters:
DPTYPE The type of the DPps to be deleted in case of confirmation.
DPps A list of DPps to be printed in the dialox box.

Returns:
 True if the user want to delete ; false else.

```
int RateCtrl_Paging_createRatingPlots( dyn_string ratesList )
```

Creates a set of FwTrendingPlot(s) from a list of Rates. One plot with one curve is created for a given rate.

Usage Public

Parameters:
ratesList The list of Rates (DPs).

Returns:
 The number of created plots.

```
int RateCtrl_Paging_deleteRatingPages( string pattern = "*",
string partition = "",
bool confirm = false
)
```

Usage Public

Parameters:
pattern The pattern the page name must match to be deleted.
partition The partition the page must be linked with to be deleted.
confirm Must be set to true to make a confirmation dialog pop up.

Returns:
 The number of deleted pages.

```
int RateCtrl_Paging_deleteRatingPlots( string pattern = "*",
bool confirm = false
)
```

Deletes plots matching the pattern.

Usage Public

Parameters:
pattern The pattern to select plots to delete.
confirm This function will pop up a dialog box asking for confirmation before deleting if true.

Returns:
 The number of deleted plots.

```
int RateCtrl_Paging_makePages( dyn_string listOfPlots,
string partition = ""
)
```

Creates a set of fwTrending pages from a given list of plots. Plots are groupe 6 by 6. Optional parameter "partition" is used to create the pages names. "partition" given when makePages is called by makePagesByPartition.

Usage Public

Parameters:
listOfPlots The list of plot to be placed in pages.
partition The name of the partition used to select this plots. "" means that this function has not been called to create pages by partition.

Returns:
 The number of created pages.

```
RateCtrl_Paging_makePagesByPartion ( dyn_string list )
```

Creates a set of fwTrending pages from a list of given plots. First build one list of plot by partition from the given list. Then call makePages with the lists and their partition.

Usage Public

Parameters:
list The list of plot to be displayed in pages.

```
bool RateCtrl_TrendingTree_addPagesToTree( dyn_string pages )
```

Adds pages to the TrendingTree. Due to node creation issues, this function removes any node belonging to TreningTree having their name beginning with 't'. Such nodes are supposed to be created

Usage Public

Parameters:
pages The list of pages to add to the tree.

Returns:
 True in case of success.

```
int RateCtrl_TrendingTree_clearTree( bool confirm = false )
```

Clears the TrendingTree.

Usage Public

Parameters:
confirm Ask for confirmation with a pop up dialog if set to true.

Returns:
 The number of deleted nodes.

Generated on Mon Aug 25 14:47:31 2008 by  1.4.1

[Main Page](#) | [Directories](#) | [File List](#) | [File Members](#)

[scripts](#) / [libs](#) / [RateCtrl](#)

Archiving.ctl File Reference

Detailed Description

This library contains utility functions for rate services archiving.

Creation Date
 01/08/2008

Modification History
 01/08/2008: nothing

Constraints
 None

Usage
 Public

PVSS managers
 VISION, CTRL

Author:
 Jean-Francois Menou

Function Documentation

```
int RateCtrl_Archiving_startRateRecord( string dp,
int smoothingType = -1,
int timeSC = 0,
float totVal = 0
)
```

Starts Oracle archiving for a given data point. Sets archiving for the DP and the value DPE.

Usage Public

Parameters:
dp Archiving is started for this dp.
smoothingType See lbArchive_setArchOn() 2nd parameter.
timeSC See lbArchive_setArchOn() 3rd parameter.
totVal See lbArchive_setArchOn() 4th parameter.

Returns:
 See lbArchive_setArchOn() return code.

```
int RateCtrl_Archiving_startRatesRecord( string pattern )
```

Starts Oracle archiving for a set of data points.

Usage Public

Parameters:
pattern Archiving is started for DPs matching this pattern.

Returns:
 The number of started DP archives.

```
int RateCtrl_Archiving_startRatesRecordList( dyn_string list )
```

Starts Oracle archiving for a given list of data points. For each DP in list, archiving is started with default parameter of RateCtrl_Archiving_startRateRecord()

Usage Public

Parameters:
list Archiving is started for All DPs in this list.

Returns:

```
int RateCtrl_Archiving_stopRateRecord( string dp )
```

Stops Oracle archiving for a given data point.

Usage Public

Parameters:
dp Archiving is stopped for this dp.

Returns:
 See lbArchive_setArchOff() return code.

```
int RateCtrl_Archiving_stopRatesRecord( string pattern )
```

Stops Oracle archiving for a set of data points.

Usage Public

Parameters:
pattern Archiving is stopped for DPs matching this pattern.

Returns:
 The number of stopped DP archives.

```
int RateCtrl_Archiving_stopRatesRecordList( dyn_string list )
```

Stops Oracle archiving for a given list of data points.

Usage Public

Parameters:
list Archiving is stopped for all DPs in this list.

Returns:
 The number of stopped DP archives.

Generated on Mon Aug 25 14:47:31 2008 by  1.4.1