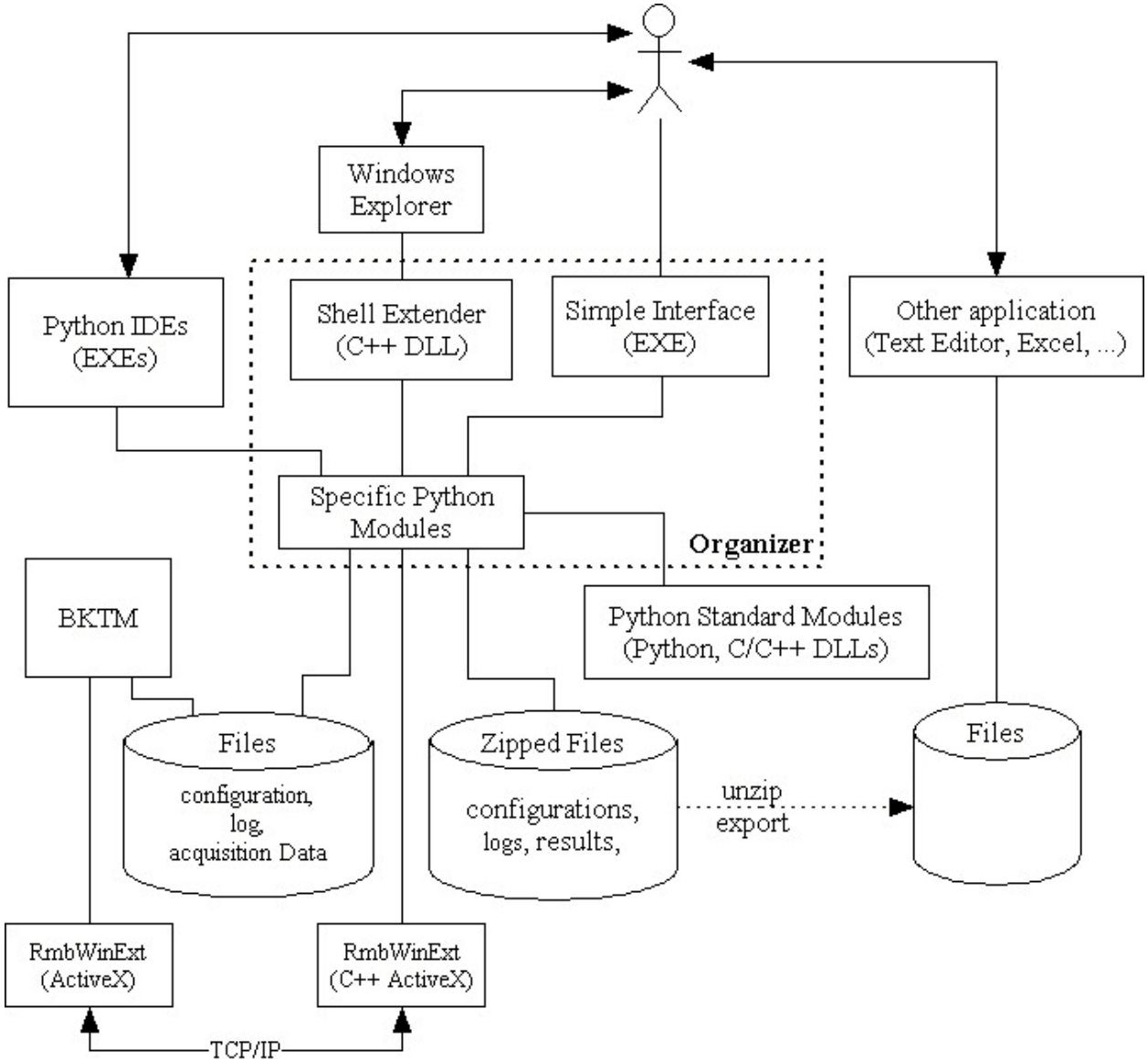
Architecture Organizer

Choix fondamentaux

- développer une application centrée autour de ses “ documents ” et non autour de ses “ programmes ”.
- séparer la partie “ Test & Mesure ” de la partie “ Organisation & Automatisation”
- utiliser les possibilités d’extension de l’explorateur Windows, plutôt que de développer une application complète indépendante. Cela permet d’avoir d’emblée un outil d’une grande qualité ergonomique, familier à de nombreux utilisateurs.
- choisir le langage Python comme langage principal de développement.
- développer un module “ Journal ” qui permet de garder une trace du fonctionnement de l’application, et notamment un maximum d’informations en cas d’anomalie.
- choisir les formats de fichiers principaux :
    - “ ZIP ” comme format de regroupement de fichiers liés fonctionnellement
    - “ INI ” comme format des fichiers de configuration
    - “ EXCEL ” comme format d’exportation
    - “ ASCII ” pour les fichiers de résultats
- séparer le plus possible les dialogues utilisateurs des composants d’exécution. Cela permet de :
    - faire évoluer les fonctions offertes dans les meilleures conditions de coût et de sécurité.
    - réaliser un ensemble de tests unitaires exhaustif se déroulant sans intervention de l’utilisateur.
    - utiliser l’environnement d’exécution adapté à une fonction particulière: Explorer, IDE Python, ..

Vue d ’ensemble



Alcos

Exécutables, DLLs, Composants, Modules

L'Organizer est composé :

- de modules spécifiques écrits en Python
- d'un DLL d'interface écrit en C++ permettant d'étendre les fonctions du Windows Explorer (Tooltips, ..)
- d'une application " alpyshell.exe " écrite en Visual Basic qui offre 3 interfaces :
    - dialogue de gestion des instruments et mesures
    - fenêtre d'exécution de scripts simples
    - fenêtre d'exécution de scripts parcourant une arborescence

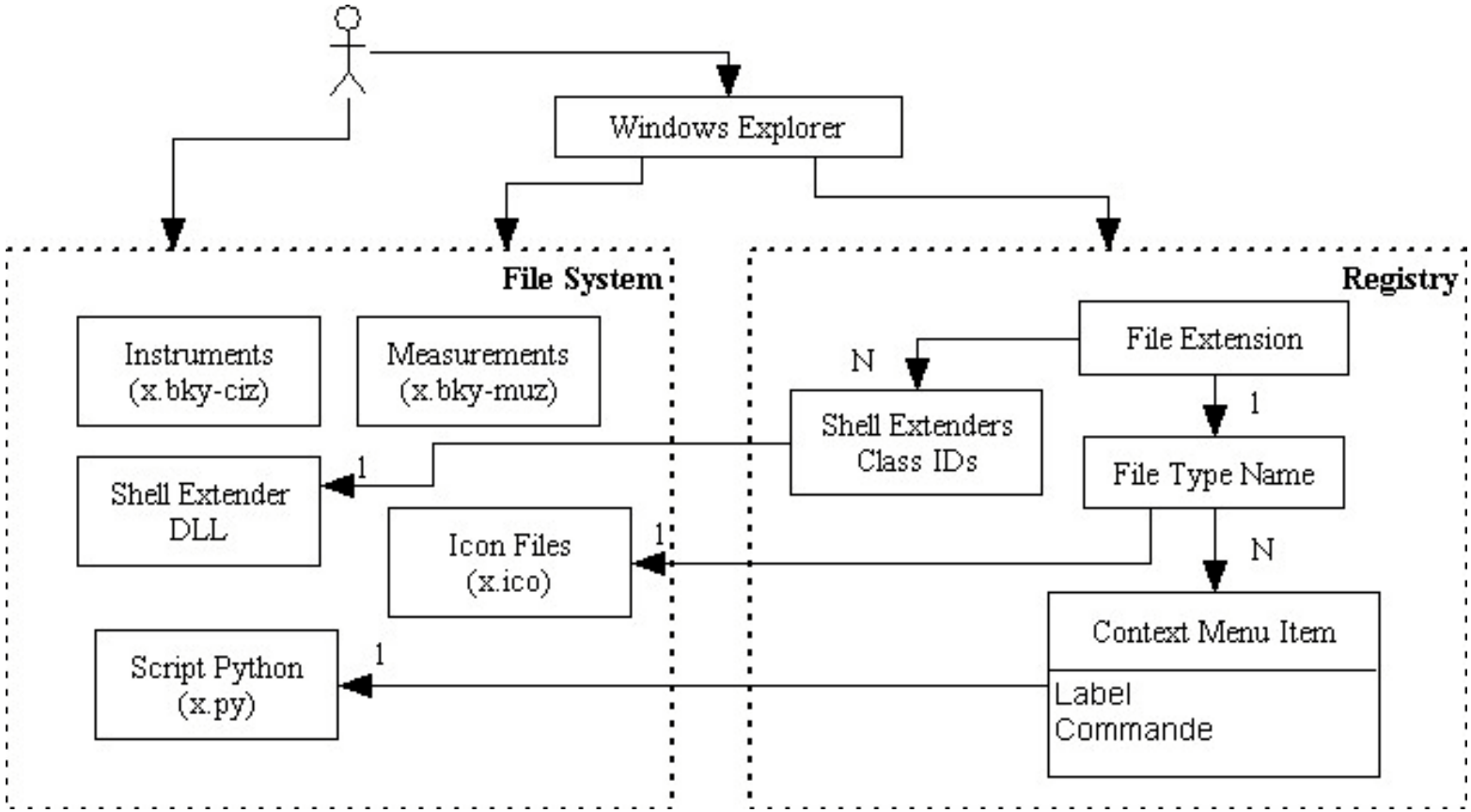
Les modules spécifiques sont développés et mis au point avec les " Integrated Developement Environnement " standards Python : PythonWin.exe, IDLE, Python.exe.

L'Organizer utilise :

- les modules génériques de la bibliothèque standard Python
- un composant ActiveX développé par Alcos et distribué par Bourbaky : RmbWinExtender

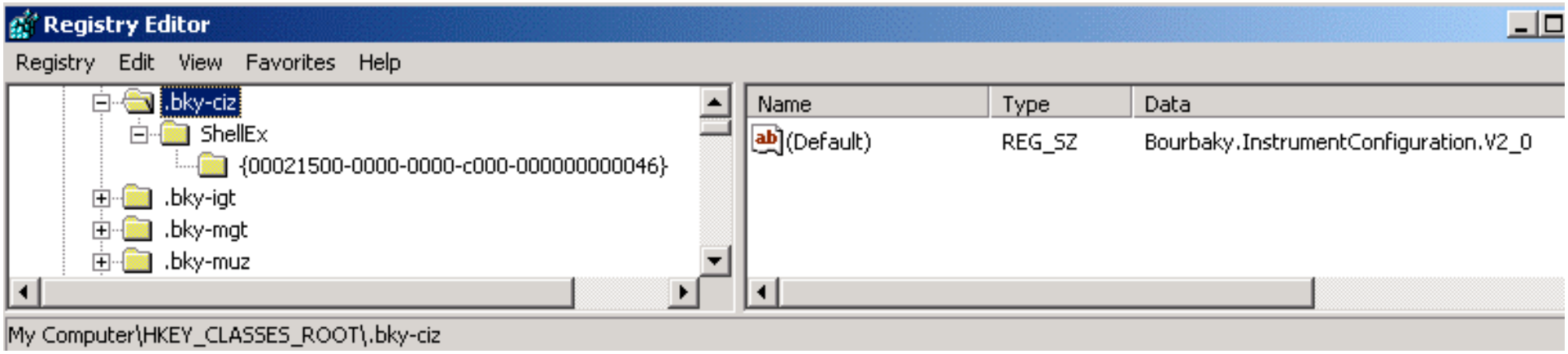
Alcos

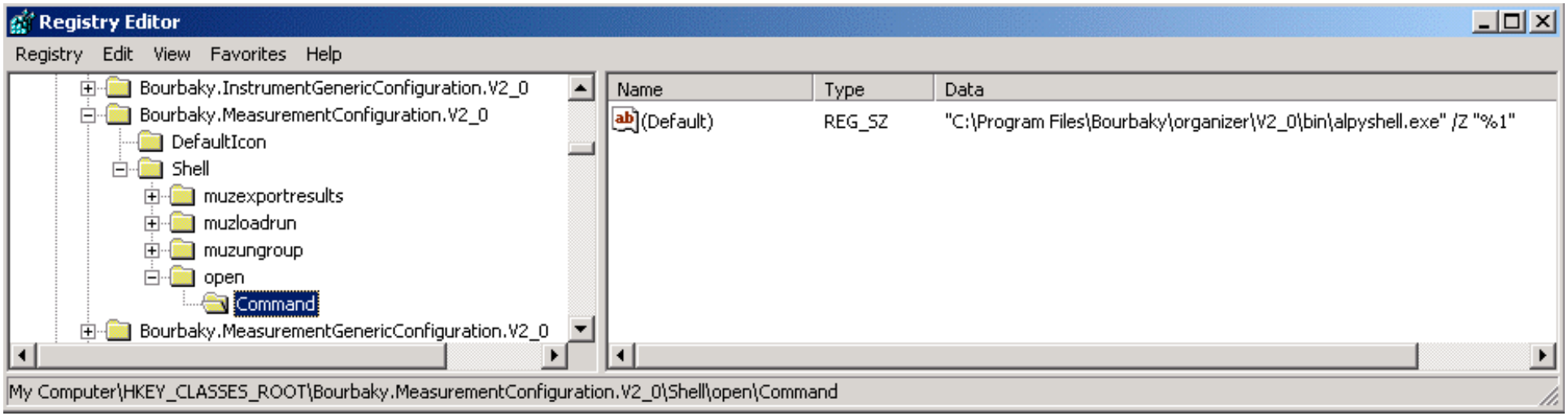
Intégration avec “ Windows Explorer ”



Alcos

Registry samples





Alcos

Exemple Python : exportation des Résultats

```
#--------------------------------------------------------------------
class MuzTreeExporter(RunnerBase):
    def   init  (self,directoryPath):
    def runSpecifics(self):
    def getExpectedDuration(self):
    def durationVisit(self,dirname,names):
    def getRelativePath(self, targetPath):
    def runVisit(self,dirname,names):
        # sort in order to create same file even if system order changed
        names[:] = [ name.lower() for name in names]
        names.sort()
        for nam in names:
            absPath = path.normpath(path.join(dirname,nam))
            relPath = self.getRelativePath(absPath)
            if not path.isfile(absPath):
                continue
            if path.splitext(absPath)[1] != bktm.common.FileExt.Muz:
                continue
            if self.flagTimeToStop:
                raise bktm.common.UserAbortException, 'Application interrupt i
            self.meas = bktm.measure.MeasurementZip(absPath)
            if self.meas.hasData():
                self.currentRunState = relPath
                self.logInfo("Exporting Results of %s" % relPath)
                try:
                    self.meas.exportResultsToExcel()
                except bktm.common.ResultsException, e:
                    self.logError("%s, %s" % (str(e),relPath))
            else:
                self.logInfo("No Results to export in %s" % relPath)
            if not self.meas is None: self.meas.__del__()
            self.meas = None
```

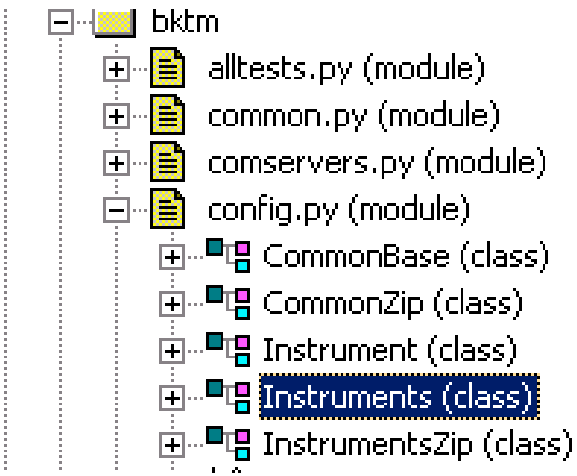

lcos

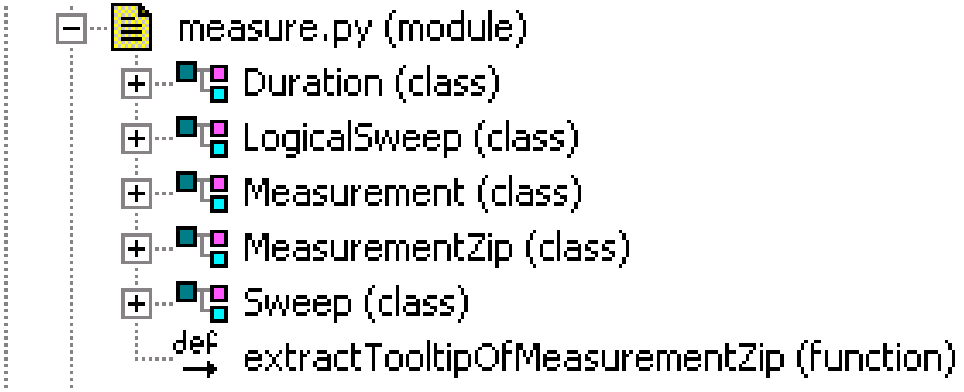
Quelques Modules Python

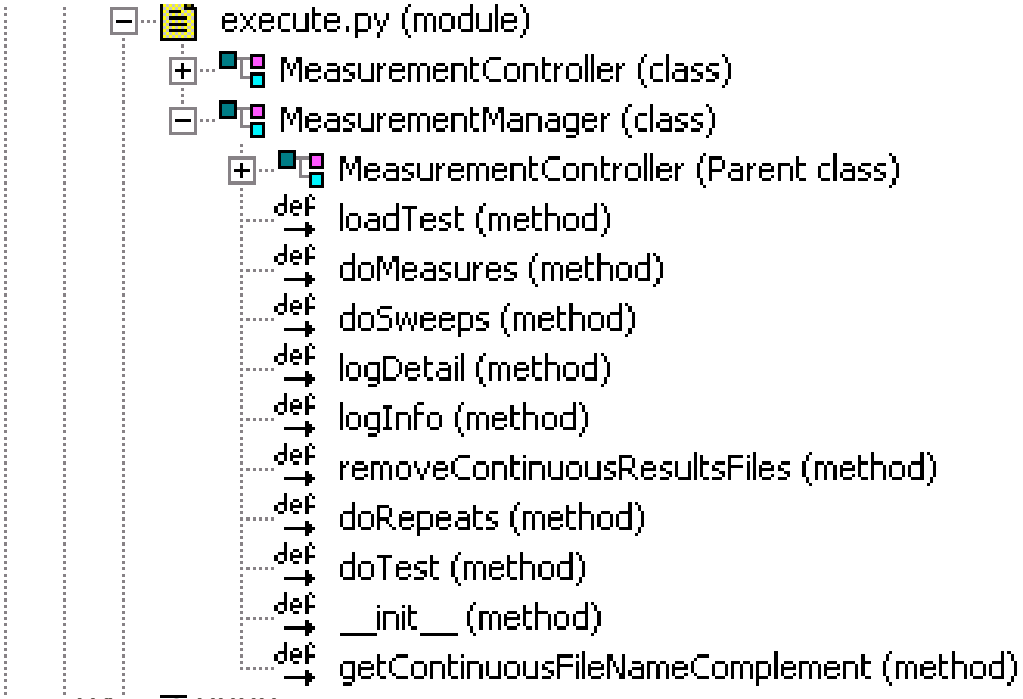
- Standard Python Library
- AddonLibraries
- BkyOrg
  - alcos
    - alltests.py (module)
    - common.py (module)
    - commondialog.py (module)
    - comservers.py (module)
    - config.py (module)
    - dateformat.py (module)
    - enumeration.py (module)
    - excel.py (module)
    - filecomparer.py (module)
    - fileselector.py (module)
    - hsm.py (module)
    - journal.py (module)
    - registry.py (module)
    - rmbwinext.py (module)
    - scriptrunner.py (module)
    - security.py (module)
    - sysstd.py (module)
    - utils.py (module)
    - win32.py (module)
    - zip.py (module)
    - __init__.py (module)
  - alcostest
  - bktm
  - bktmtest

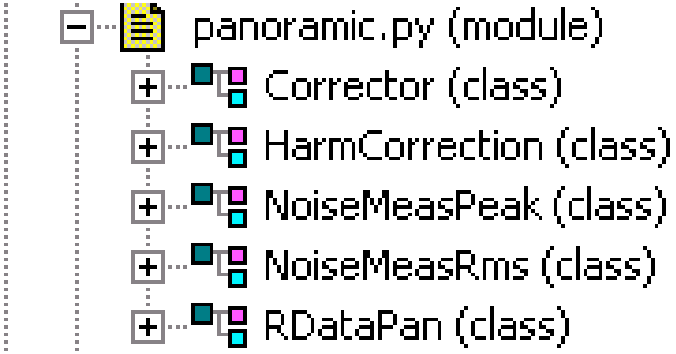
- excel.py (module)
  - Area (class)
  - Launcher (class)
  - Wrapper (class)
    - def saveAllSheetsAsText (met
    - def getRange (method)
    - def fixStringsAndDates (metho
    - def show (method)
    - def __del__ (method)
    - def existsSheet (method)
    - def getCellValue (method)
    - def exportToSheet (method)
    - def selectSheet (method)
    - def getNamedCell (method)
    - def close (method)
    - def deleteEmptySheets (metho
    - def setNamedRange (method)
    - def getNamedRange (method)
    - def __init__ (method)
    - def setRange (method)
    - def saveSheetAsText (method
    - def setCellValue (method)
    - def hide (method)
    - def createSheet (method)
    - def setCellName (method)
    - def save (method)
    - def setAreaName (method)

Quelques Modules Python, suite



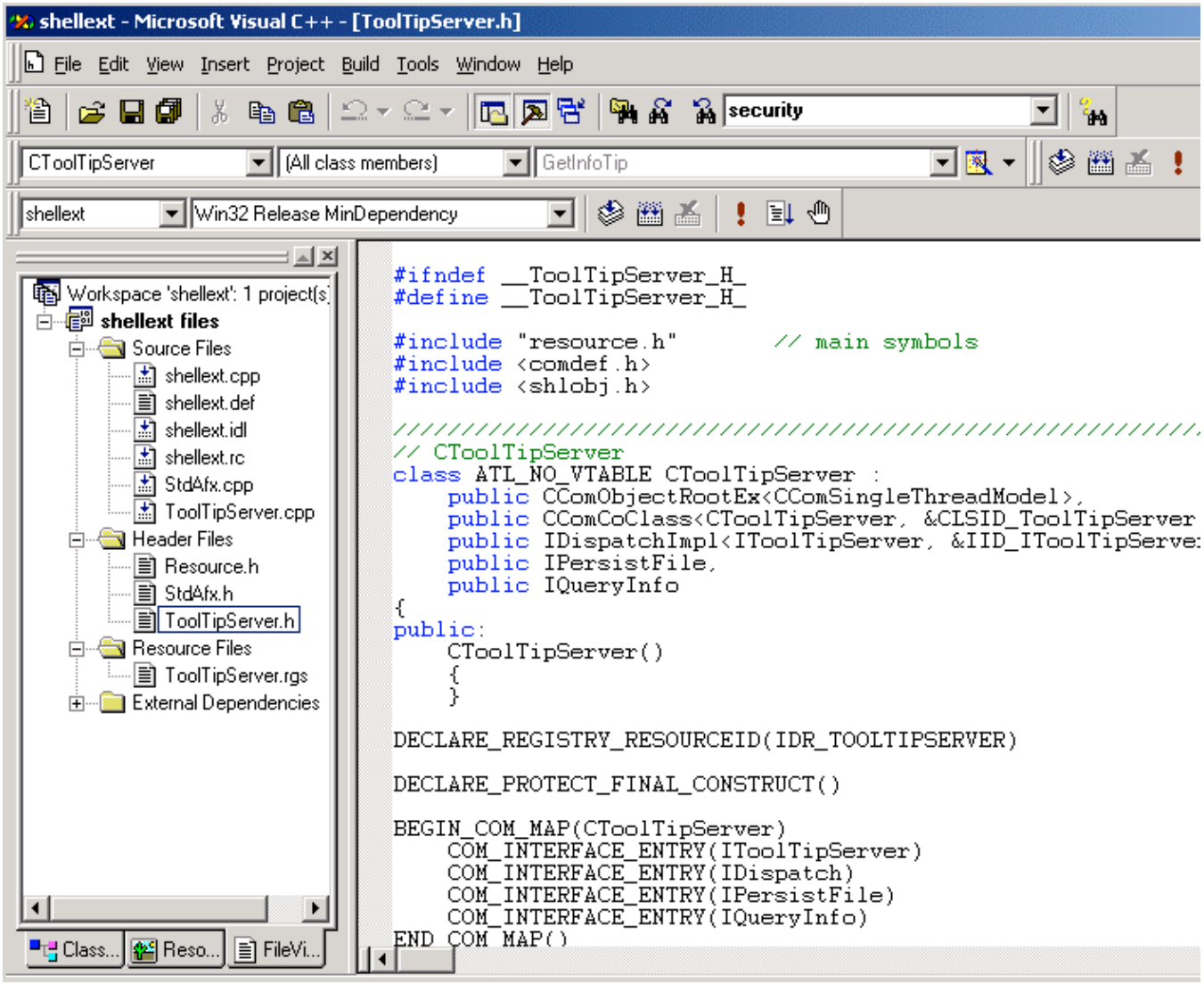






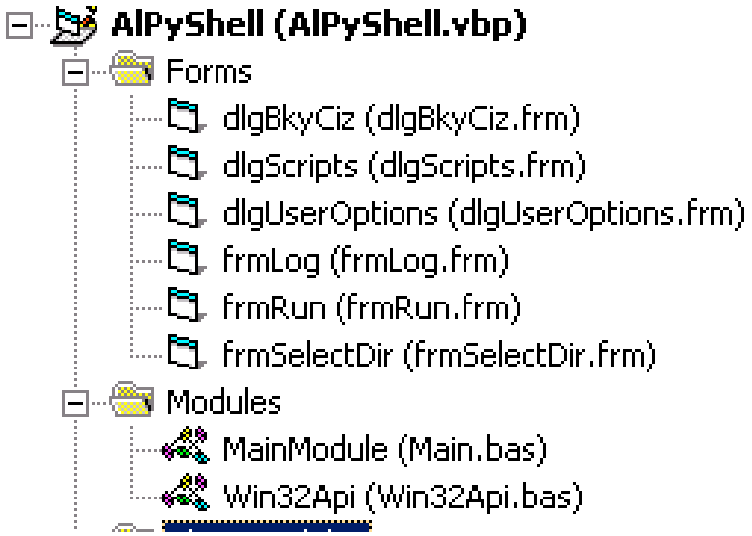
Alcos

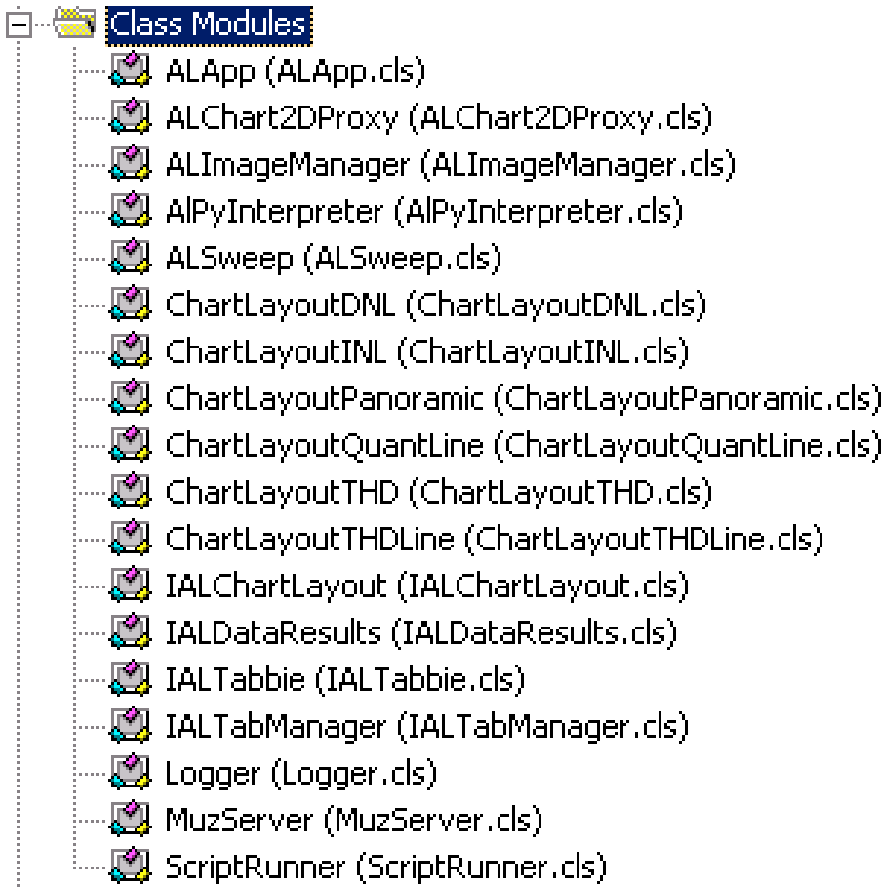
shellext.dll se conforme à l ’interface attendue par le Windows Explorer



Alcos

Alpyshell.vbp, un projet Visual Basic Orienté Objet (autant que possible)

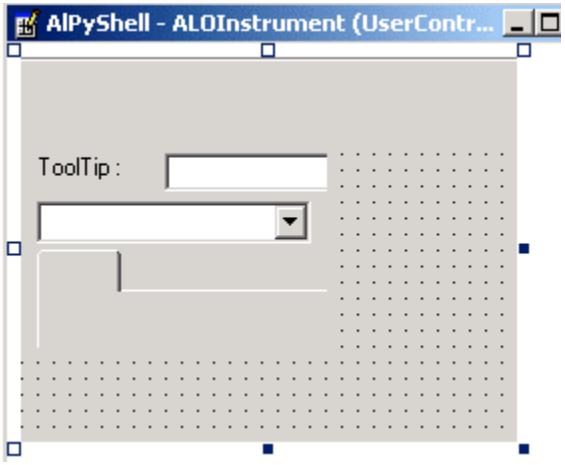


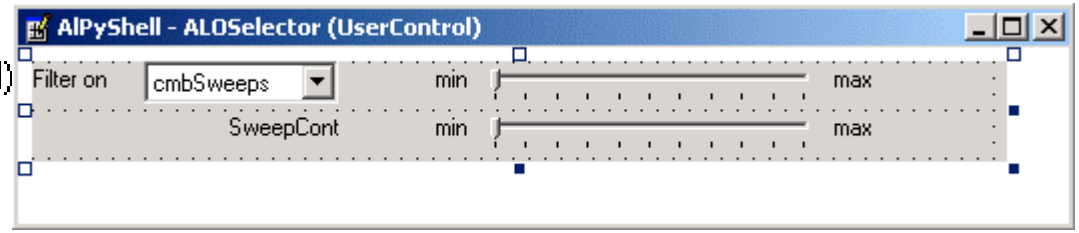


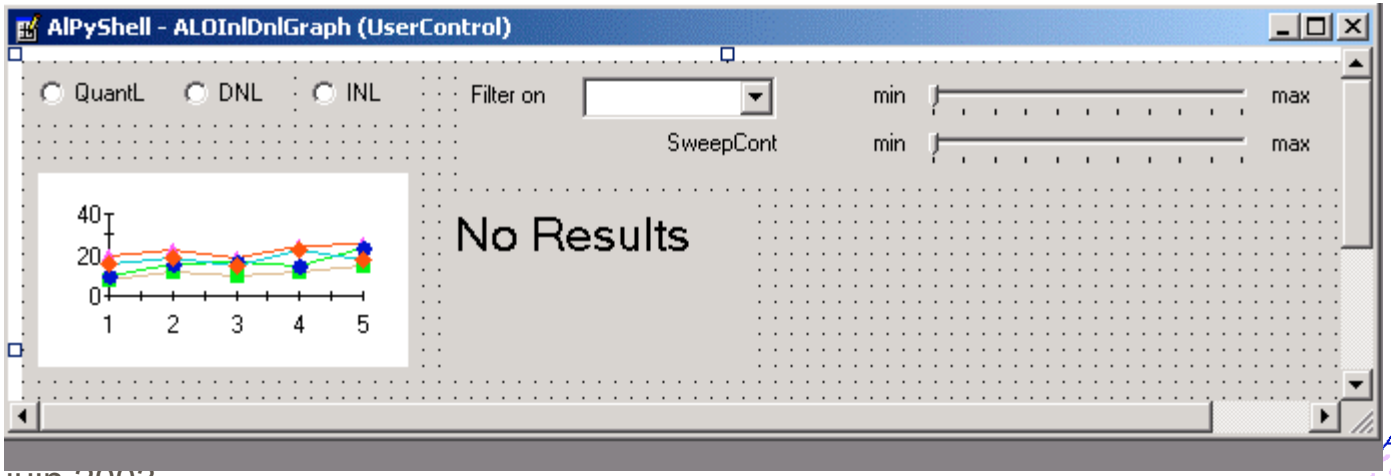
Alcos

Alpyshell.vbp, les contrôles spécifiques

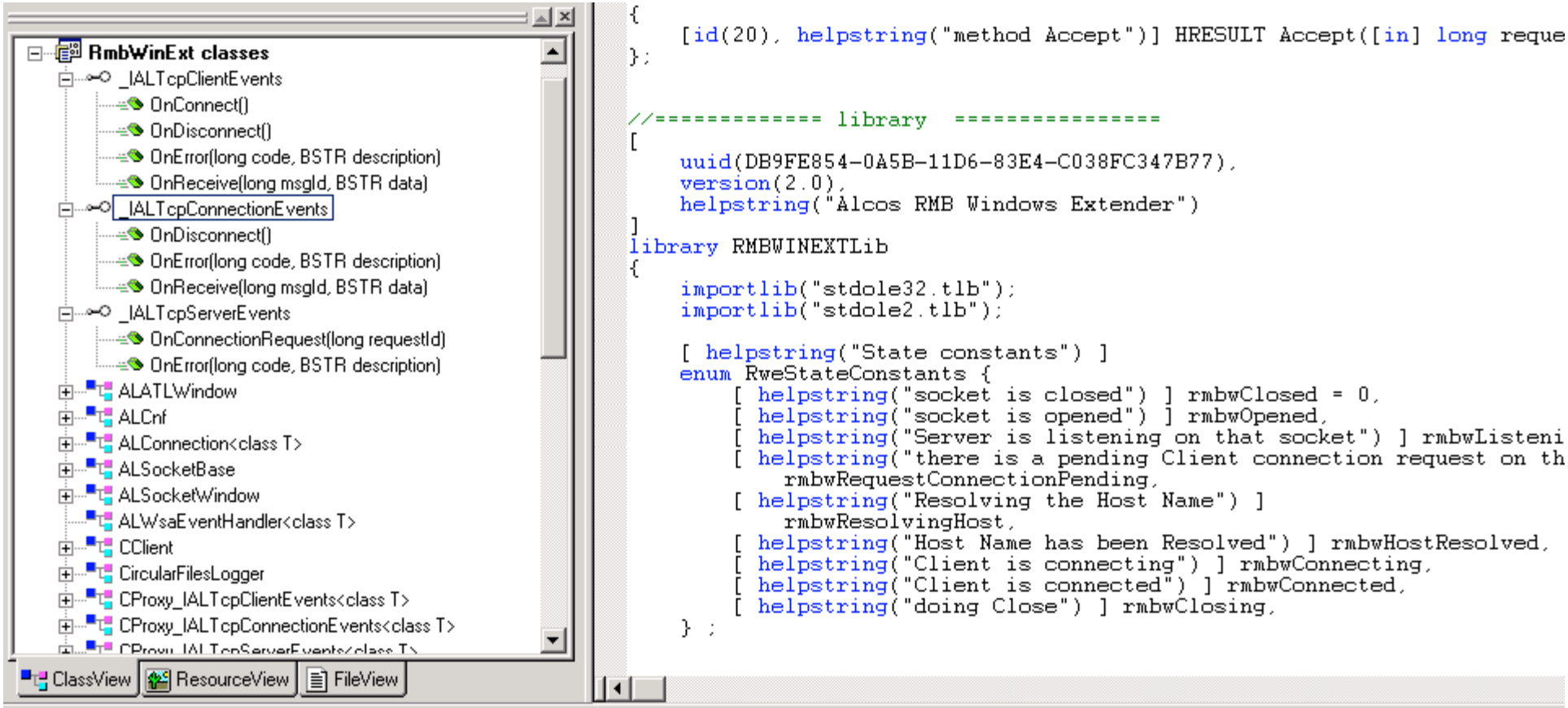
- User Controls
  - ALOCommonIM (ALOCommonIM.ctl)
  - ALODataGeneric (ALODataGeneric.ctl)
  - ALODataScope (ALODataScope.ctl)
  - ALOExecute (ALOExecute.ctl)
  - ALOInlDnlGraph (ALOInlDnlGraph.ctl)
  - ALOInstrument (ALOInstrument.ctl)
  - ALOJournal (ALOJournal.ctl)
  - ALOLogViewer (ALOLogViewer.ctl)
  - ALOMeasure (ALOMeasure.ctl)
  - ALOSectionData (ALOSectionData.ctl)
  - ALOSelector (ALOSelector.ctl)
  - ALOThdGraph (ALOThdGraph.ctl)







RmbWinExt.dll : un composant ATL



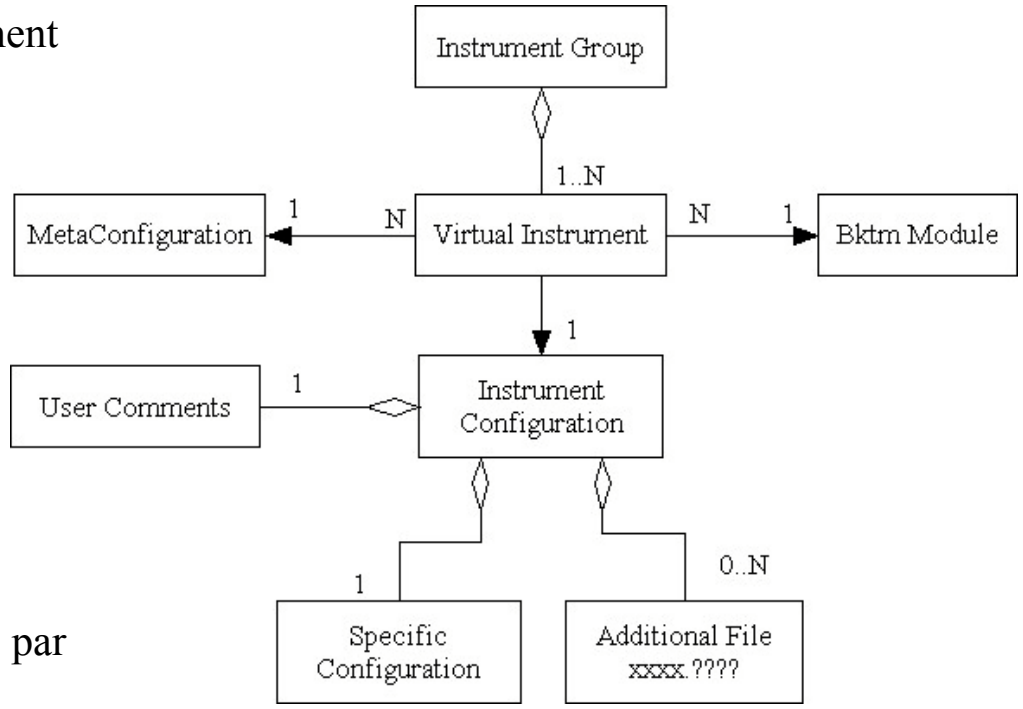
Alcos

Instrument Group

Un Instrument Group comporte de 1 à N « Virtual Instrument »

Chaque “ Virtual Instrument ” associe :

- la MetaConfiguration qui spécifie comment l’Organizer doit présenter et gérer l’Instrument Réel
- le module programme de BKTM qui contrôlera l’instrument réel
- une configuration courante



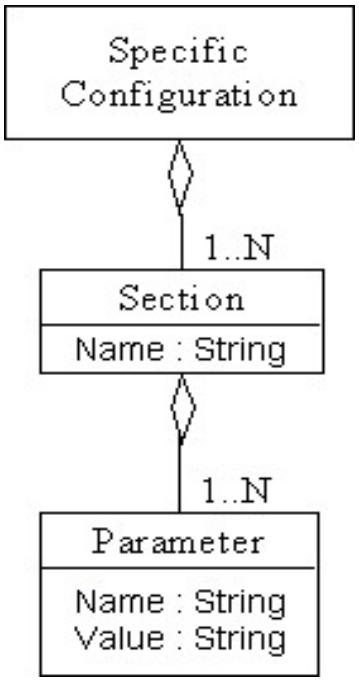
Une configuration courante est constituée de :

- commentaires de l’utilisateur
- fichiers qui seront Uploadés/ Downloadés par l’intermédiaire de BKTM :
  - une configuration spécifique stockée dans un fichier dont l’Organizer comprend le format
  - de 0 à N fichiers additionnels que l’Organizer se contente de copier tel quel

Alcos

Configuration Spécifique

La “ Specific Configuration ” est extraite d’un fichier de syntaxe proche de celle des fichiers .INI de Windows. Elle comporte N sections, chaque section comporte N paramètres. Chaque paramètre est identifié par son nom et a une valeur courante.

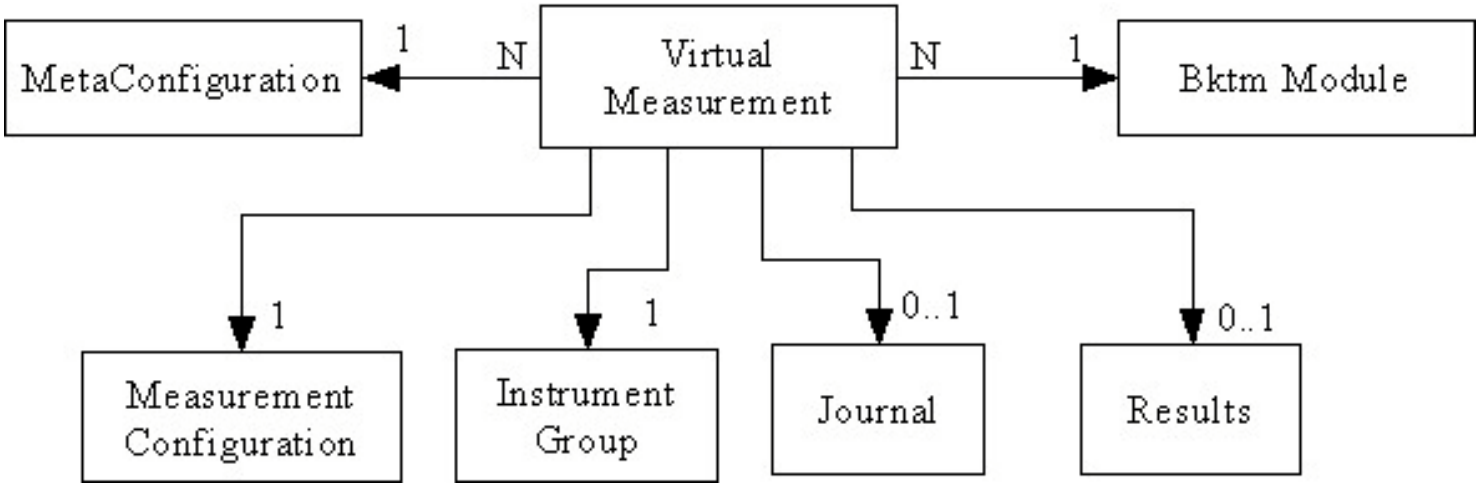


Alcos

Virtual Measurement

Chaque “ Virtual Measurement ” associe :

- la MetaConfiguration qui spécifie comment l’Organizer doit présenter et gérer la Mesure Réelle
- le module programme de BKTM qui effectuera la mesure
- une configuration courante
- un groupe d’instruments utilisé pour effectuer la Mesure
- un journal qui enregistre les évènements importants de la mesure, notamment les erreurs éventuelles
- un ensemble de résultats appelé « Results »

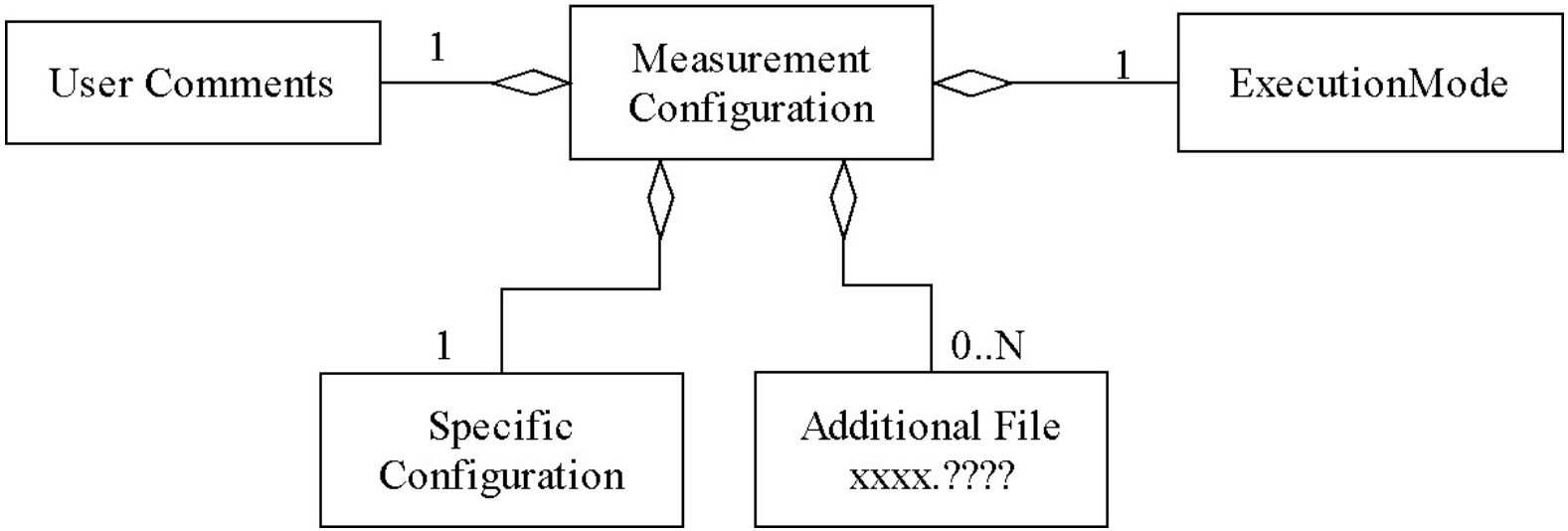


Alcos

Measurement Configuration

La configuration de Mesure courante est constituée de :

- commentaires de l'utilisateur
- le mode d'exécution et ses paramètres : SigleShot, Continuous, Sweep, ..
- les durées précédentes de chargement et d'exécution de la Mesure qui seront utilisées pour évaluer la durée probable de la prochaine exécution
- fichiers qui seront Uploadés/ Downloadés par l'intermédiaire de BKTM :
    - une configuration spécifique stockée dans un fichier dont l'Organizer comprend le format
    - de 0 à N fichiers additionnels que l'Organizer se contente de copier tel quel



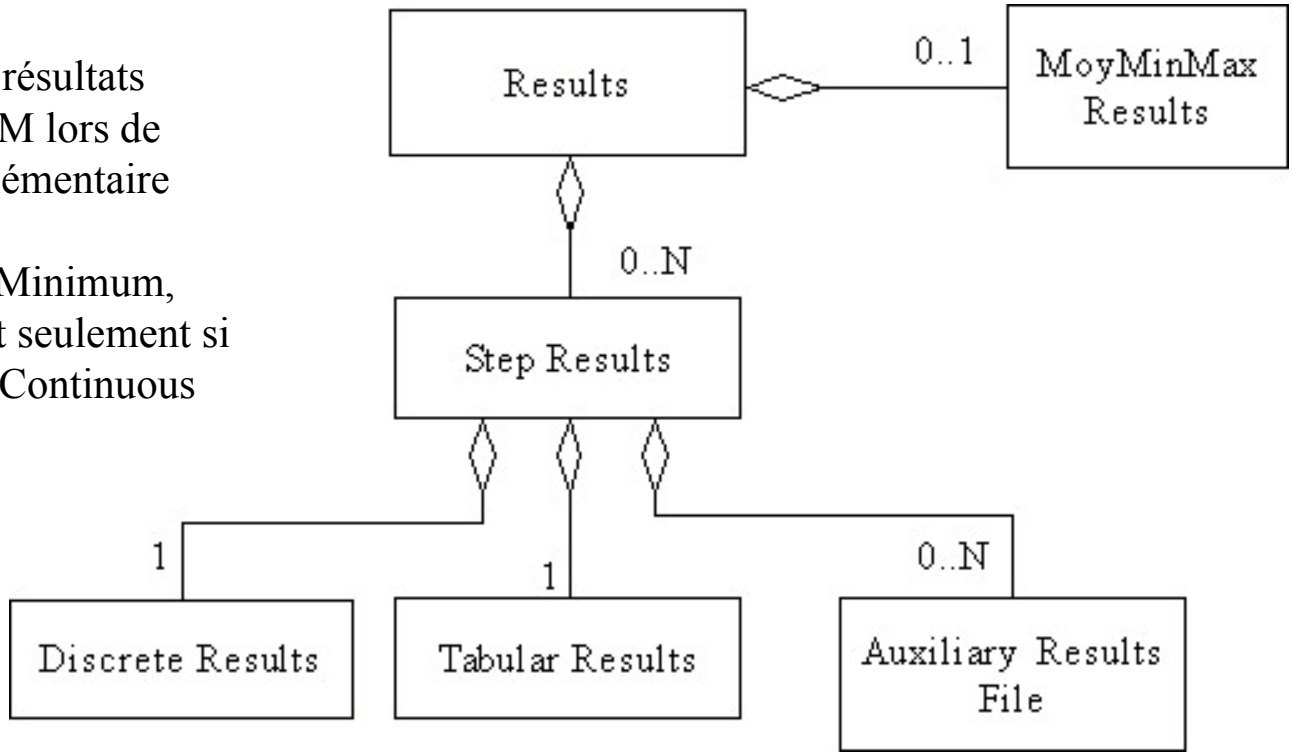
Alcos

Results

Chaque « Results » associe :

- de 0 à N “ Step Results ”, résultats générés par le module BKTM lors de l’exécution d’une mesure élémentaire

- 1 ensemble de résultats “ Minimum, Maximum, Moyenne ”, si et seulement si l’exécution se fait en mode Continuous



Chaque “ Step Results ” regroupe :

- 1 ensemble de valeurs discrètes
- 1 ensemble éventuellement vide de valeurs tabulées
- de 0 à N fichiers additionnels que l’Organizer se contente de copier tel que

Alcos

Fichiers Résultats

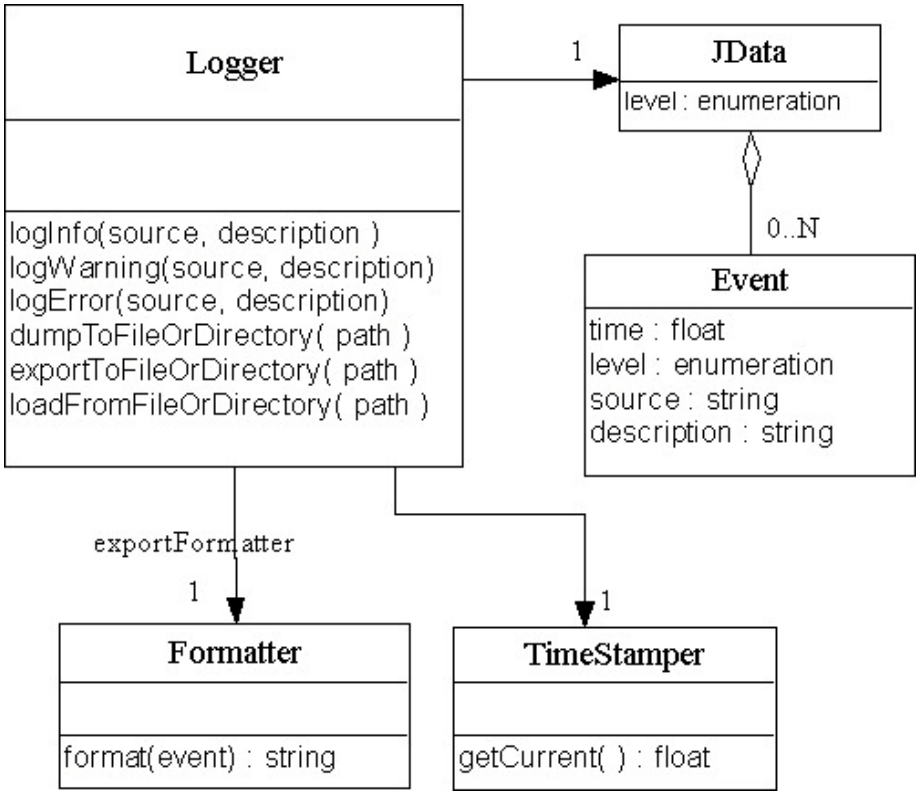
Un fichier xx.bky-muz contient éventuellement un catalogue “ Results ” qui contient :
- 0 ou 1 fichier binaire contenant les minimum, maximum et moyennes de résultats obtenues en mode continuous
- de 0 à N séries de résultats élémentaires. Chaque série élémentaire contient :
- un fichier xxx.bky-rif contenant les valeurs discrètes
- un fichier xxx.bky-rad contenant les tables de résultats. Pour certaines mesures, ce fichier peut être vide.
- de 0 à N fichiers auxiliaires.

Les noms des fichiers de résultats sont de la forme MMMPPP.EXT
- **MMM** => radical qui dépend du type de mesure
- **EXT** => extension : **bky-rif**, **bky-rad**, quelconque pour les fichiers auxiliaires
- **PPP** => selon le mode d’exécution
- **rien** en mode SingleShot
- **_NNNNN** en mode Continuous, NNNNN est le numéro d’ordre décimal de la mesure élémentaire de 00001 à 99999
- **_NNN** en mode SingleSweep, NNN est le numéro d’ordre décimal du pas du Sweep, de 001 à 999

_PPP_NNN en mode DoubleSweep, PPP et NNN sont les numéros d’ordre décimal des pas du Sweep, PPP pour le premier paramètre et NNN pour le deuxième

Journal

Le **Journal** est un des services de base de l'Organizer.
Un **Logger** permet aux modules clients de demander l'enregistrement d'**Event**s en fournissant :
level : Information, Avertissement, Erreur
source : une chaîne de caractères
description : une chaîne de caractères



Au moment où l'**Event** est créé, le **Logger** ajoute un nombre flottant **time** attribué par le TimeStamper. Par défaut il s'agit de la valeur rendue par time.time( ), date et heure courante.

Le **Logger** stocke les **Event**s dans un objet **JData**. L'attribut **level** de celui-ci contient la valeur maximum du **level** de tous les Events.

Le **Logger** peut sauvegarder ou recharger le **JData** dans un fichier binaire.
Le **Logger** peut exporter le **JData** dans un fichier texte. Le format utilisé dépend du **Formatter**

Alcos