Guide pour le développement d'un module SALOME 2 en Python

Auteur: Caremoli C.

Contents

1 Présentation

Ce document a pour objectif de décrire les différentes étapes du développement d'un module SALOME 2 en Python. Il commence par un bref rappel des principes de SALOME 2. Puis, il décrit les étapes successives qui seront suivies pour arriver à une configuration de module complète.

2 Rappels sur les modules et composants SALOME 2

Dans son principe, la plate-forme SALOME 2 est décomposée en une plate-forme de base nommée module KERNEL et une collection de modules divers.

Un module est un produit compilable et installable qui se concrétise par une arborescence source qui doit respecter les règles générales SALOME2 et qui comprend une procédure de construction, installation qui respecte également les règles SALOME2. Chaque module est géré en configuration dans un module CVS indépendant. Chaque module peut livrer des versions à son rythme dans le respect des règles de cohérence de SALOME2. A chaque module est associée une base de tests de non régression.

Un module a des dépendances avec les autres modules (utilise, est utilisé). Le module KERNEL constitue la base de la plate-forme SALOME2. Tous les autres modules dépendent du module KERNEL.

Sous projets	**Modules**	**Utilise**	**Est utilisé par**
SP1 - Environnement de production			
SP2 - Architecture	KERNEL		MED, GEOM, SMESH, VISU,SUPERV
SP2 - Architecture	MED	KERNEL	SMESH, VISU
SP3 - Géométrie	GEOM	KERNEL	SMESH
SP5 - Maillage	SMESH	KERNEL, MED	
SP6 - Superviseur	SUPERV	KERNEL	
SP7 - Visualisation	VISU	KERNEL, MED	

Un module contient un ou plusieurs composants SALOME. Un composant SALOME est un objet CORBA qui respecte les règles SALOME et qui est déclaré à SALOME au moyen d'un catalogue. Un composant SALOME peut être doté d'une interface utilisateur graphique (GUI) qui doit elle-même respecter les règles SALOME.

3 Les étapes de construction du module exemple

Le module exemple choisi pour illustrer le processus de construction d'un module est extrèmement simple. Il contiendra un seul composant et ce composant aura un seul service nommé getBanner qui acceptera une chaine de caractères comme unique argument et qui retournera une chaine de caractères obtenue par concaténation de "Hello " et de la chaine d'entrée. Ce composant sera complété par un GUI graphique écrit en PyQt.

Les différentes étapes du développement seront les suivantes :

- créer une arborescence de module
- créer un composant SALOME 2 chargeable par un container Python
- configurer le module pour que le composant soit connu de SALOME
- ajouter un GUI graphique
- rendre le composant utilisable dans le superviseur

4 Création de l'arborescence du module

Dans un premier temps, on se contentera de mettre dans le module exemple un composant SALOME écrit en Python qui sera chargeable par un container Python. Il suffit donc d'une interface idl et d'une implantation Python du composant. Pour mettre en oeuvre ceci dans un module SALOME 2, il faut l'arborescence de fichier suivante:

```
+ PYHELLO1_SRC
  + build_configure
  + configure.in.base
  + Makefile.in
  + adm_local
    + unix
      + make_commence.in
      + make_omniorb.in
      + config_files
  + bin
    + VERSION
    + runAppli.in
    + runSalome.py
  + idl
    + Makefile.in
    + PYHELLO_Gen.idl
  + src
    + Makefile.in
    + PYHELLO
      + Makefile.in
      + PYHELLO.py
  + doc
```

Le module a pour nom PYHELLO1_SRC et le composant PYHELLO. Tous les fichiers sont indispensables à l'exception de VERSION, runAppli et runSalome.py. VERSION sert pour documenter le module, il doit donner sa version et ses compatibilités ou incompatibilités avec les autres modules. Il est donc fortement recommandé mais pas indispensable au fonctionnement. Les fichiers runAppli et runSalome.py ne sont pas indispensables mais facilitent la mise en oeuvre de l'exemple.

Mise en garde : il ne faut pas copier les fichiers de la plate-forme de base (KERNEL) pour initialiser une arborescence de module. Il est, en général, préférable de copier les fichiers d'un autre module comme GEOM ou MED.

4.1 Mise en oeuvre d'autoconf, configure

SALOME utilise autoconf pour construire le script configure qui sert à l'installation pour tester la configuration du système et pour préconfigurer les fichiers Makefile de contruction du module. Le fichier build_configure contient une procédure qui à partir de configure.in.base et au moyen d'autoconf construit le script configure. Tous les fichiers dont l'extension est in sont des squelettes qui seront transformés par le processus de configure.

Presque tous les fichiers utilisés pour ce processus sont localisés dans la plate-forme de base qui est référencée par la variable d'environnement KERNEL_ROOT_DIR. Cependant quelques fichiers doivent être modifiés en fonction du module cible. C'est le cas bien sur de build_configure et de configure.in.base qui doivent en général adaptés.

Les fichiers de base pour le configure du module KERNEL et des autres modules sont rassemblés dans le répertoire salome_adm du module KERNEL. Il faut cependant, pour pouvoir utiliser les objets CORBA du module KERNEL surcharger les fichiers make_commence.in et make_omniorb.in

du répertoire salome_adm. Cette surcharge est réalisée en mettant les fichiers make_commence.in et make_omniorb.in modifiés dans le répertoire adm_local du module exemple.

config_files est un répertoire dans lequel on peut mettre les fichiers m4 qui servent à tester la configuration du système dans le processus de configure. Si les fichiers de salome_adm ne sont pas suffisants, on peut en ajouter dans adm_local.

4.2 Répertoire idl

Dans le répertoire idl, il faut un Makefile qui doit mettre en oeuvre la compilation du fichier idl PYHELLO_Gen.idl et installer tous ces fichiers dans les bons répertoires de l'installation du module. Il suffit de modifier la cible IDL_FILES pour obtenir un fichier adapté.

Concernant le fichier idl lui-même, il doit définir un module CORBA dont le nom doit être différent du nom du module pour éviter les conflits de nom et définir une interface CORBA qui dérive à minima de l'interface Component du module Engines. Le nom du module CORBA sera PYHELLO_ORB et le nom de l'interface PYHELLO_Gen.

4.3 Répertoire src

Le répertoire src contiendra tous les composants et GUI graphiques du module. Chacune de ces entités doit avoir son propre répertoire.

Le module ne contiendra pour le moment qu'un seul répertoire pour le moteur du composant PYHELLO et son nom sera PYHELLO.

Le Makefile se contente de déclencher le parcours des sous répertoires qui sont décrits par la cible SUBDIRS.

4.3.1 Répertoire PYHELLO

Le répertoire contient le module Python qui représente le composant et donc contient la classe PYHELLO et un fichier Makefile dont le rôle est simplement d'exporter le module PYHELLO.py dans le répertoire d'installation du module SALOME.

Le module PYHELLO.py contient la classe PYHELLO qui dérive de l'interface PYHELLO_Gen du module CORBA PYHELLO_ORB__POA et de la classe SALOME_ComponentPy_i du module SALOME_ComponentPy.

4.4 Répertoire doc

Il ne contient rien pour le moment. Il pourrait contenir ce document.

4.5 Répertoire bin

VERSION sert pour documenter le module, il doit donner sa version et ses compatibilités ou incompatibilités avec les autres modules. Il est donc fortement recommandé mais pas indispensable au fonctionnement.

Le fichier runAppli.in est l'équivalent du runSalome du module KERNEL configuré pour mettre en oeuvre le module KERNEL et ce module PYHELLO.

Le fichier runSalome.py est le fichier du module KERNEL avec une correction de bug pour tourner seulement avec un container Python, une modification de la fonction test qui crée le composant PYHELLO au lieu d'un composant MED et un développement pour disposer de la complétion automatique en Python.

5 Création d'un composant chargeable par un container

Les fichiers présentés ci-dessus suffisent pour construire et installer le module PYHELLO1_SRC, lancer la plate-forme SALOME constituée des modules KERNEL et PYHELLO1 et demander au container Python le chargement d'un composant PYHELLO.

Toutes les étapes suivantes supposent que les logiciels prérequis de SALOME sont accessibles dans l'environnement du développeur de modules.

5.1 Construction, installation

Dans PYHELLO1_SRC, faire:

```
export KERNEL_ROOT_DIR=<chemin d'installation du module KERNEL>
./build_configure
```

Aller dans ../PYHELLO1_BUILD et faire:

```
../PYHELLO1_SRC/configure --
prefix=<chemin d'installation du module PYHELLO1>
make
make install
```

5.2 Lancement de la plate-forme

Aller dans <chemin d'installation du module PYHELLO1> et faire:

```
./bin/salome/runAppli
```

Cette commande lance SALOME configurée pour KERNEL et le module PYHELLO1. A la fin de ce lancement l'utilisateur est devant un interpréteur Python configuré pour SALOME et qui donne accès aux objets CORBA de SALOME.

runAppli est un shell qui exécute un script Python en lui passant des arguments en ligne de commande:

```
python -i $PYHELLO_ROOT_DIR/bin/salome/runSalome.py --modules=PYHELLO --xterm -
-containers=cpp,python --killall
```

Ces arguments indiquent que l'on prendra le script runSalome.py situé dans le module PYHELLO, que l'on activera le composant PYHELLO, les impressions seront redirigées dans une fenêtre xterm, on lancera un container Python et tous les processus SALOME existant avant le lancement seront tués.

Pour que cette commande fonctionne, il faut préalablement avoir positionné les variables d'environnement suivantes:

```
export KERNEL_ROOT_DIR=<chemin d'installation du module KERNEL>
export PYHELLO_ROOT_DIR=<chemin d'installation du module PYHELLO>
```

Cette méthode d'activation des modules et composants de SALOME 2 tend à confondre module et composant. Dans ce cas (1 composant par module), il n'y a pas de difficulté à paramétrer le lancement. Il suffit d'indiquer derrière l'option --modules la liste des composants demandés (KERNEL est inutile) et de fournir autant de variables d'environnement qu'il y a de composants. Le nom de ces variables doit être <Composant>_ROOT_DIR et doit donner le chemin du module contenant le composant. Dans le cas où on a plusieurs composants par module, c'est un peu plus compliqué. Ce sera présenté ultérieurement.

Mise en garde: il est possible que le lancement de SALOME 2 n'aille pas jusqu'au bout. En effet dans certaines circonstances, le temps de lancement des serveurs CORBA peut être long et dépasser le timeout fixé à 21 secondes. Si la raison en est le temps de chargement important des bibliothèques dynamiques, il est possible qu'un deuxième lancement dans la foulée aille jusqu'au bout.

5.3 Chargement du composant exemple

Pour avoir accès aux méthodes du composant, il faut importer le module PYHELLO_ORB avant de demander le chargement du composant au container Python. Ce container Python a été rendu accessible dans runSalome.py au moyen de la variable container:

```
import PYHELLO_ORB
c=container.load_impl("PYHELLO","PYHELLO")
c.makeBanner("Christian")
```

La dernière instruction doit retourner 'Hello Christian'. Pour voir les objets CORBA créés par ces actions, faire:

```
clt.showNS()
```

On peut voir que le composant a été créé et enregistré dans un contexte de nommage qui peut être incorrect en raison d'un bug identifié dans la version 1.2.1 du module KERNEL.

6 Composant SALOME déclaré

Pour le moment, le composant PYHELLO a été chargé en faisant une requête directe au container Python. Ce n'est pas la méthode standard pour charger un composant. La voie normale passe par le service LifeCycle qui utilise les services du catalogue pour identifier le composant et ses propriétés puis appelle le container demandé pour charger le composant.

Pour pouvoir utiliser cette méthode, il faut déclarer le composant dans un catalogue au format XML dont le nom doit être <Composant>Catalog.xml. Dans notre cas ce sera PYHELLOCatalog.xml. Ce catalogue sera rangé dans le répertoire resources. Arborescence actualisée:

```
+ PYHELLO1_SRC
  + build_configure
  + configure.in.base
  + Makefile.in
  + adm_local
  + bin
  + idl
  + src
  + doc
  + resources
    + PYHELLOCatalog.xml
```

En dehors de l'ajout du répertoire resources et du fichier PYHELLOCatalog.xml, le reste des fichiers est identique. Il faut cependant modifier le Makefile.in de tête pour que le catalogue soit bien installé dans le répertoire d'installation. Il suffit de le spécifier dans la cible RESOURCES_FILES.

6.1 Construction, installation

Il n'est pas nécessaire de refaire un configure pour prendre en compte cette modification. Il suffit d'aller dans PYHELLO1_BUILD et de faire:

```
./config.status
make
make install
```

6.2 Lancement de la plate-forme

Le lancement de la plate-forme se passe de la même manière que précédemment. Aller dans PYHELLO1_INSTALL et faire:

```
./bin/salome/runAppli
```

6.3 Chargement du composant exemple

La méthode de chargement du composant n'est pas très différente de la fois précédente. On utiilise maintenant les services du module LifeCycle au lieu d'appeler directement le container. La séquence d'appel est contenue dans la fonction test de runSalome.Py.

```
c=test(clt)
c.makeBanner("Christian")
```

La fonction test crée le LifeCycle. Puis elle demande le chargement du composant PYHELLO dans le container FactoryServerPy:

```
def test(clt):
    """
     Test function that creates an instance of PYHELLO component
     usage : pyhello=test(clt)
    """
    import LifeCycleCORBA
    lcc = LifeCycleCORBA.LifeCycleCORBA(clt.orb)
    import PYHELLO_ORB
    pyhello = lcc.FindOrLoadComponent("FactoryServerPy", "PYHELLO")
    return pyhello
```

6.4 Chargement depuis l'interface applicative (IAPP)

Pour pouvoir charger dynamiquement un composant en utilisant la barre à composants de l'IAPP, il faut déclarer l'icone représentative du composant dans le catalogue. Pour la déclarer il suffit d'ajouter une ligne pour l'icone au catalogue du composant:

```
<component-icone>PYHELLO.png</component-icone>
```

et de mettre le fichier correspondant dans le répertoire resources du module.

Pour tester la bonne configuration de la barre à composants, lancer SALOME 2 comme précédemment puis à partir de l'interpréteur Python lancer l'IAPP par:

```
startGUI()
```

et charger le composant en cliquant sur l'icone de PYHELLO après avoir ouvert une étude. L'IAPP doit signaler que le GUI du composant n'est pas correctement configuré mais le composant sera quand même créé après un temps d'attente. On peut le constater en tapant:

```
clt.showNS()
```

7 Ajout d'un GUI graphique

L'étape suivante pour compléter le module consiste à ajouter au composant PYHELLO une interface graphique qui sera écrite en Python en utilisant la bibliothèque de widgets Qt. Cette interface graphique doit s'intégrer dans l'interface applicative de SALOME (IAPP) et doit donc respecter certaines contraintes que nous allons voir.

Tout d'abord, précisons le contour du GUI d'un composant. Le comportement du GUI est donné par un module Python dont le nom est normalisé <Composant>GUI.py. Il doit proposer des points d'entrée conventionnels qui seront utilisés par l'IAPP pour activer ce GUI ou l'informer de certains évènements. L'activation des commandes du GUI est réalisée au moyen d'une barre de menu et d'une barre à boutons qui s'intègrent dans la barre à menus et dans la barre à boutons de l'IAPP.

7.1 Module Python implantant le comportement du GUI

Le comportement du GUI du composant PYHELLO est implanté dans le module Python PYHELLOGUI.py du sous-répertoire PYHELLOGUI. Le Makefile.in localisé dans le répertoire src doit être actualisé pour parcourir le sous-répertoire PYHELLOGUI. Un Makefile.in doit être ajouté dans le sous-répertoire PYHELLOGUI. Les cibles importantes sont PO_FILES et EXPORT_PYSCRIPTS.

La cible EXPORT_PYSCRIPTS doit être mise à jour avec le nom des modules Python à rendre visible dans Salome, c'est à dire principalement pour qu'ils soient importables (commande import de Python).

La cible PO_FILES doit être mise à jour avec les noms des fichiers qui sont utilisés pour le multi-linguisme. Pour le moment le fichier PYHELLO_msg_en.po (traduction pour langue anglaise) est vide car le multi-linguisme n'est pas mis en oeuvre dans cet exemple.

7.2 Barre à menus et barre à boutons

Les barres à menus et à boutons du composant PYHELLO sont décrites dans un fichier au format XML pour permettre leur chargement dynamique dans l'IAPP. Ce fichier est localisé dans le répertoire resources du module et a un nom standardisé <Composant>_en.xml pour la langue anglaise. Pour la langue française, il faut également un fichier de nom <Composant>_fr.xml. Pour le composant PYHELLO, le fichier PYHELLO_en.xml contient un menu avec un item et un bouton. L'icone du bouton est fournie par le fichier ExecPYHELLO.png localisé dans le répertoire resources du module.

8 Etapes ultérieures restant à décrire

- lien avec l'étude
- lien avec la sélection
- ajout d'un popup contextuel
- comment rendre le composant supervisable
- ajout de la persistance
- avoir plusieurs composants dans le même module

9 Rapides rappels sur l'architecture SALOME 2

9.1 Les containers

Dans SALOME, les composants sont dynamiquement chargeables. Cette propriété est obtenu en utilisant un mécanisme de container.

Dans ses grandes lignes, un container est un serveur CORBA dont l'interface dispose des méthodes nécessaires pour effectuer le chargement déchargement de l'implantation d'un composant SALOME. Pour effectuer le chargement d'un composant, on appellera la méthode load_impl du container.

La mécanique de base du chargement d'un composant est dépendante du langage d'implantation choisi.

En C++, la plate-forme utilise le chargement dynamique de bibliothèque (dlopen) et un mécanisme de fonction factory dont le nom doit être <Module>Engine_factory (par exemple GEOMEngine_factory, pour GEOM). Cette fonction doit retourner l'objet CORBA effectif qui est le composant SALOME.

En Python, la plate-forme utilise le mécanisme d'import de Python (import <Module>) et instancie le composant SALOME Python en utilisant une classe (ou une factory) de même nom (<Module>) pour ce faire.

9.2 Le multi-linguisme

A compléter

9.3 Les logiciels prérequis

A compléter