Manipulation de champs dans SALOME: maquette 2010

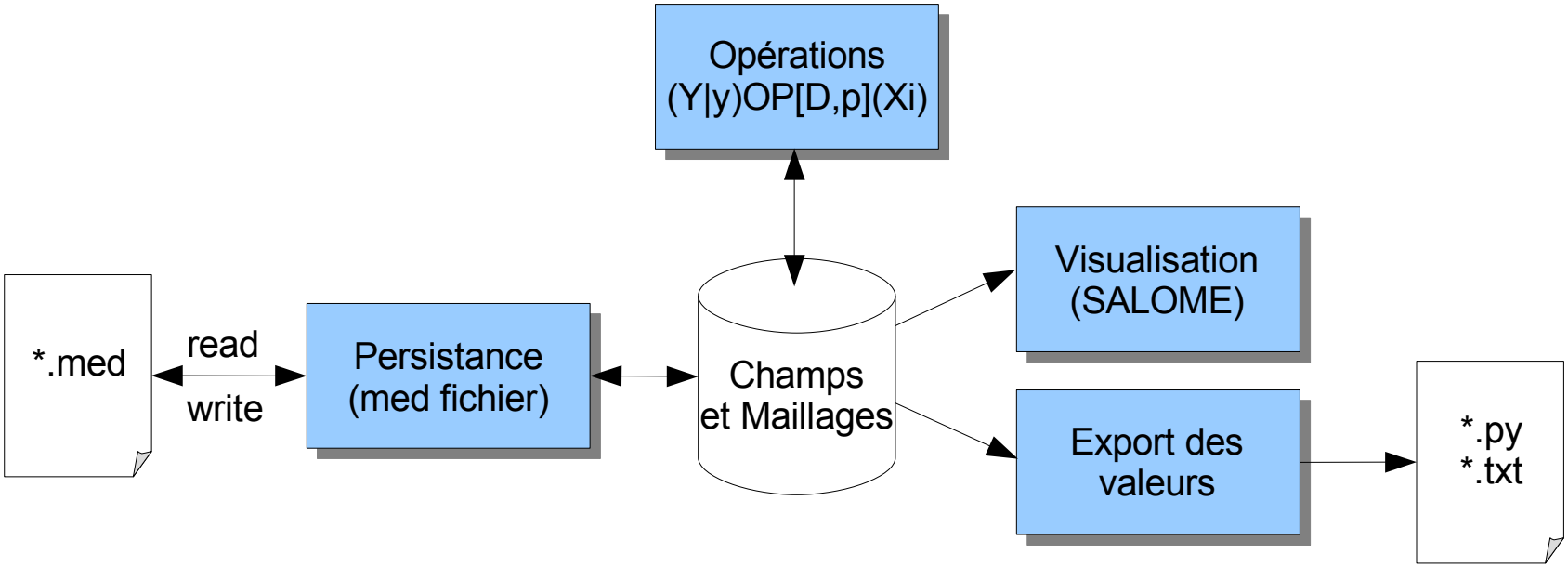
Magasin de travail EDF-CEA du 10-11 mars 2011

CHANGER L'ÉNERGIE ENSEMBLE

Rappel du contexte

- Objectifs et motivation
  - Pouvoir faire des opérations dont les opérandes sont des champs **U**(**r**,t)
  - Types d'opération (arithmétique, interpolation, génération, sémantique, diagnostic)
  - Portée d'opération (spatialle, temporelle, composante)
  - « On veut taper les opérations comme on les écrit sur le papier »
- Cas d'utilisation type
  - UC01: Produire les conditions de chargement d'une structure
  - UC02: Post-traiter un calcul paramétrique
  - UC03: Calcul d'un flux à travers une surface
- Choix techniques
  - Modèle MED pour la description des maillages et des champs
  - Composant logiciel MEDMEM, d'abord (2010), puis MEDCoupling, ensuite (2011, ...)

Découpage fonctionnel



Hypothèses de travail pour la maquette 2010

Hypothèses d'usage:

- La manipulation des champs se fait dans l'interface graphique de SALOME (c'est-à-dire dans le processus `SALOME_SessionServer`)
- On pré-sélectionne les champs à manipuler en indiquant les restrictions d'application (spatiale, temporelle, composante)
- On effectue les manipulations dans la console Python (c'est-à-dire au moyen de variables python), en formulant les opérations « comme on les écrirait » sur le papier

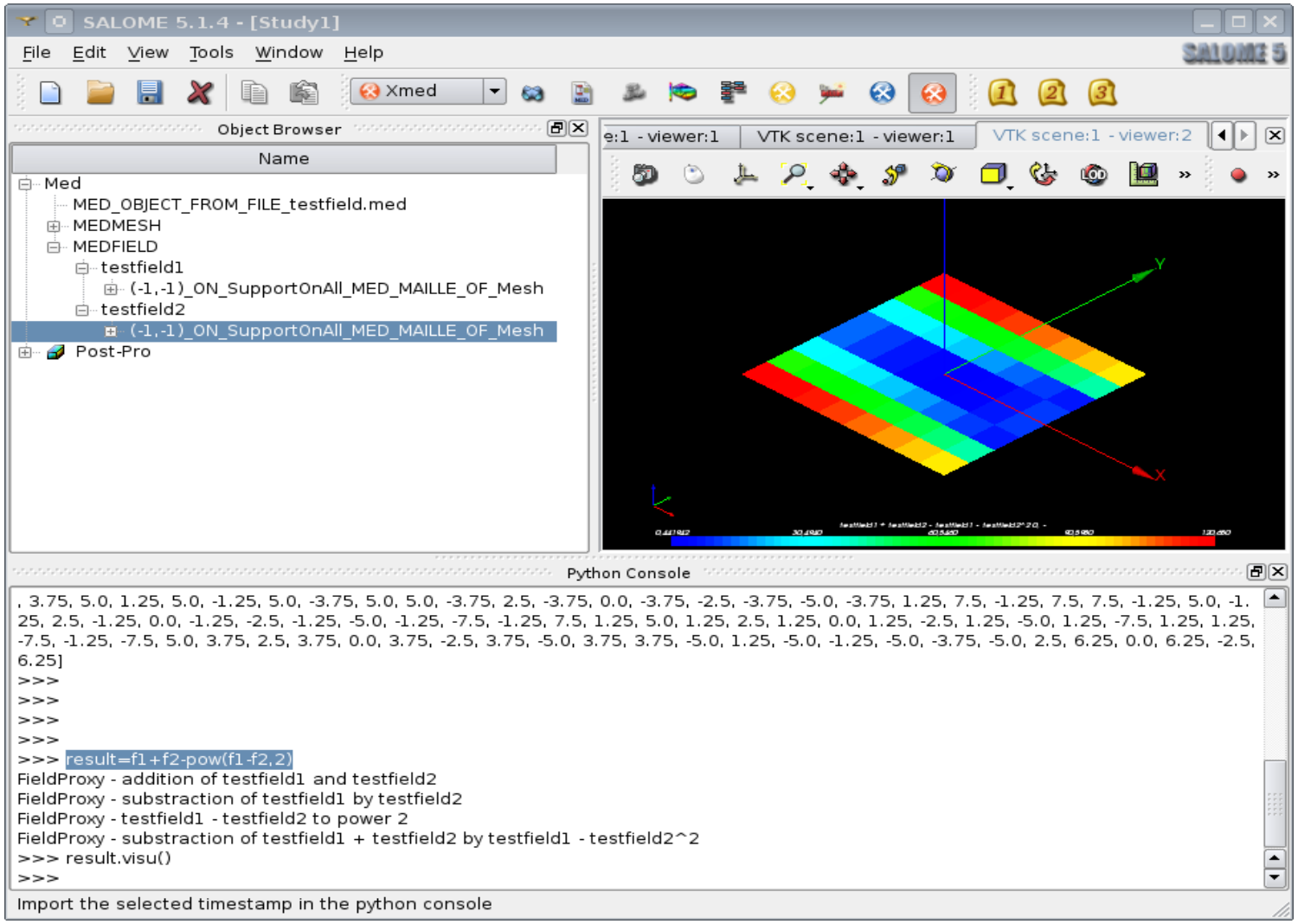
Hypothèses techniques:

- Les données sont physiquement chargées dans un composant SALOME MED (c'est-à-dire dans le processus `SALOME_Container`) et référencées dans l'étude SALOME
- On s'interdit la circulation de données: les opérations sont physiquement faites là où sont les données:
  - Performances en temps et en mémoire,
  - Utilisation directe de l'API MEDMEM,
  - Partage des données résultats entre modules via le servant CORBA

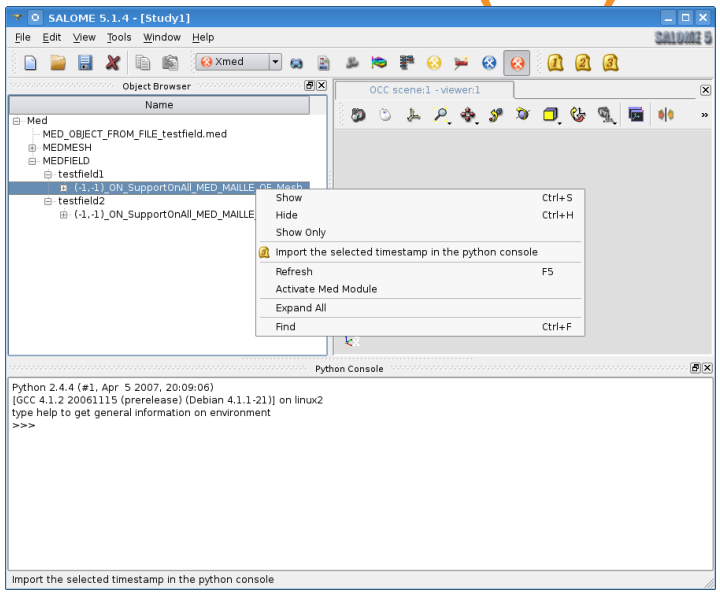
Cas d'utilisation de référence:

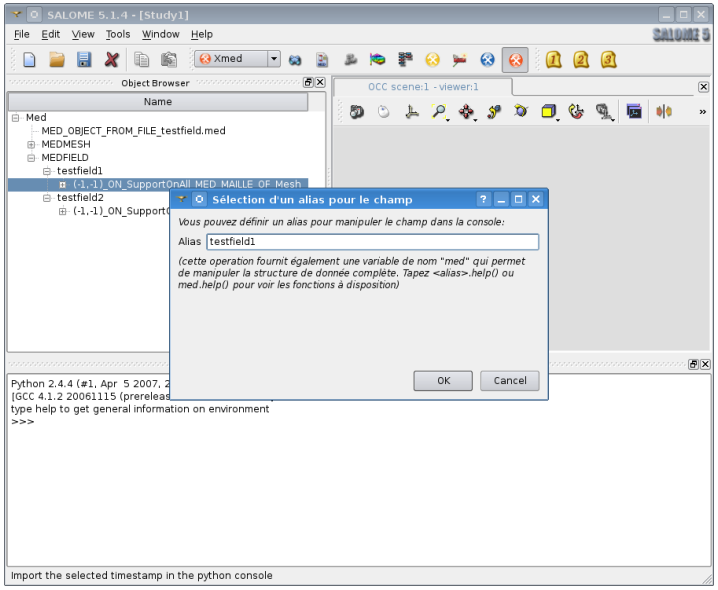
1. Chargement d'une structure MED (ou publication par un code de calcul)
2. Sélection graphique des champs à prendre en considération
3. Exécution d'opérations algébriques entre champs (+,-,*,/)
4. Contrôle visuel et export des résultats

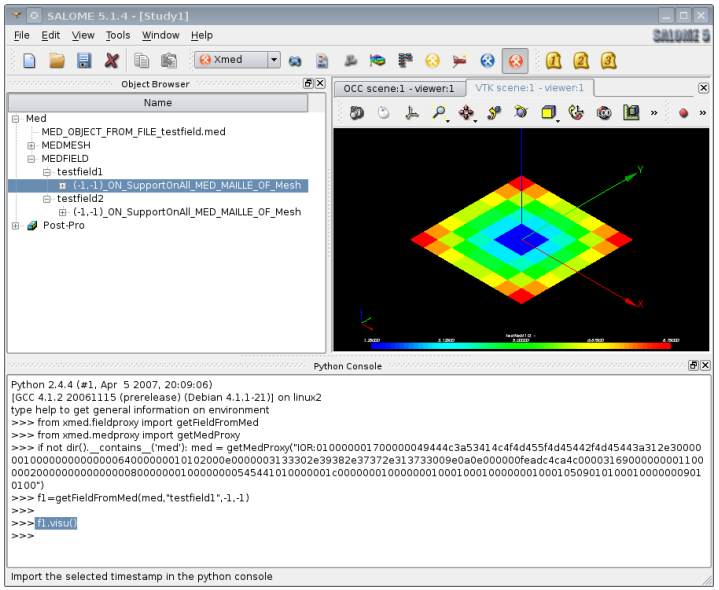
Démonstration (1/2)

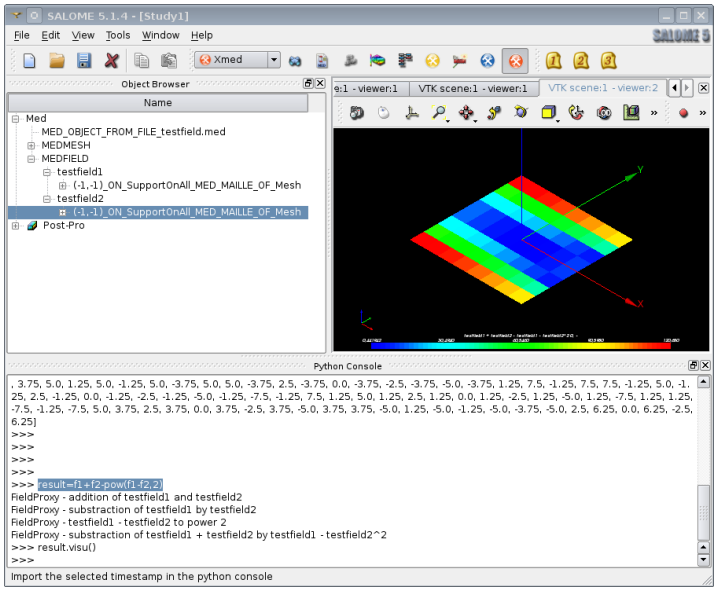


Démonstration (2/2)

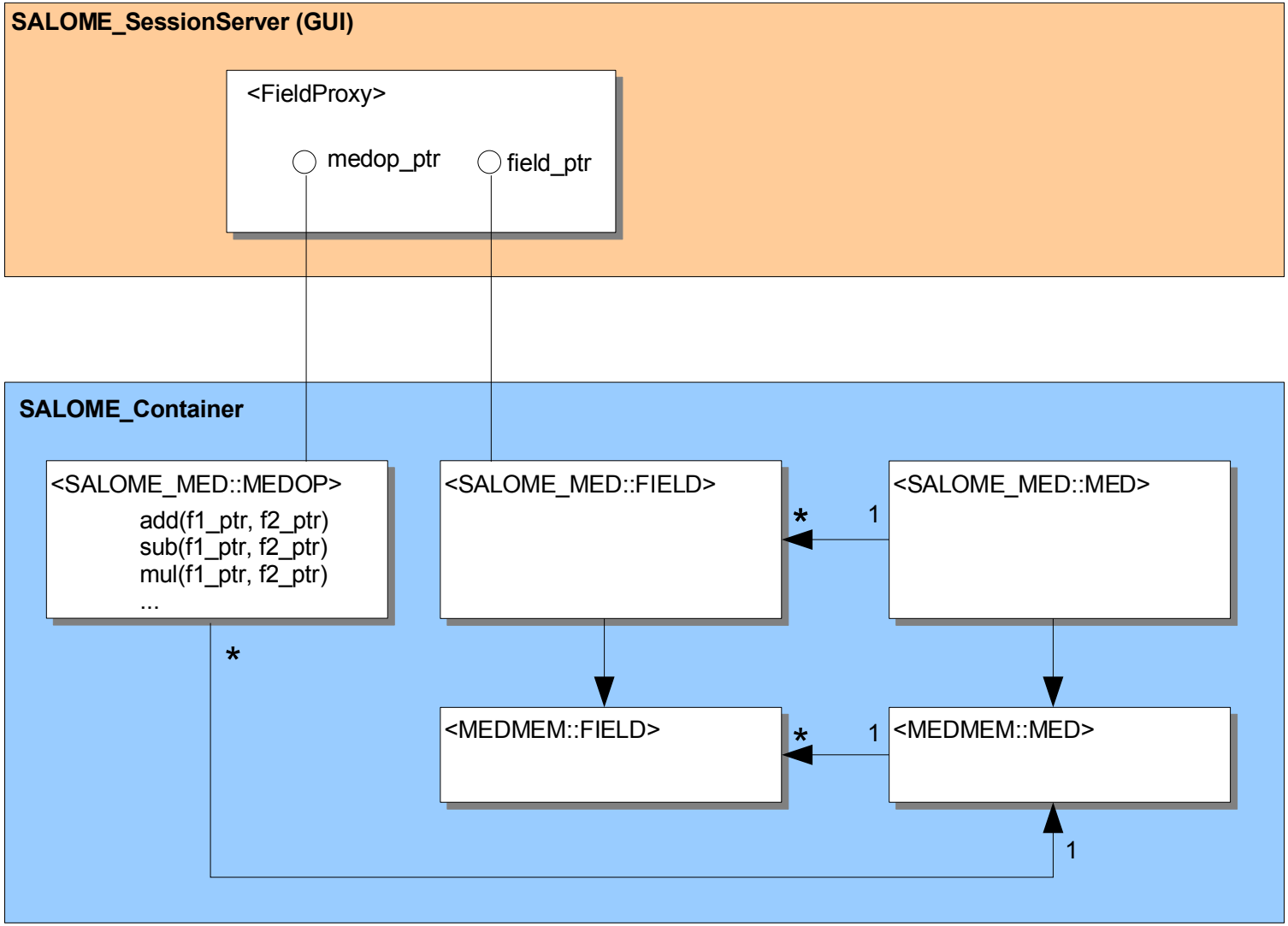








Éléments de conception



Implémentation du servant MEDOP

MEDOP.idl:

```
module SALOME_MED {
  interface FIELD;
  interface MEDOP : SALOME::GenericObj {
    /*! Addition of the fields f1 and f2 ( f1+f2) */
    FIELD add(in FIELD f1, in FIELD f2) raises (SALOME::SALOME_Exception);
    /*! Substraction of the fields f1 and f2 (f1-f2) */
    FIELD sub(in FIELD f1, in FIELD f2) raises (SALOME::SALOME_Exception);
    ...
    /*! Linear transformation of the field f (factor*f+offset) */
    FIELD lin(in FIELD f, in double factor, in double offset) raises (SALOME::SALOME_Exception);
    /*! Dublication of the field f */
    FIELD dup(in FIELD f) raises (SALOME::SALOME_Exception);
  };
};
```

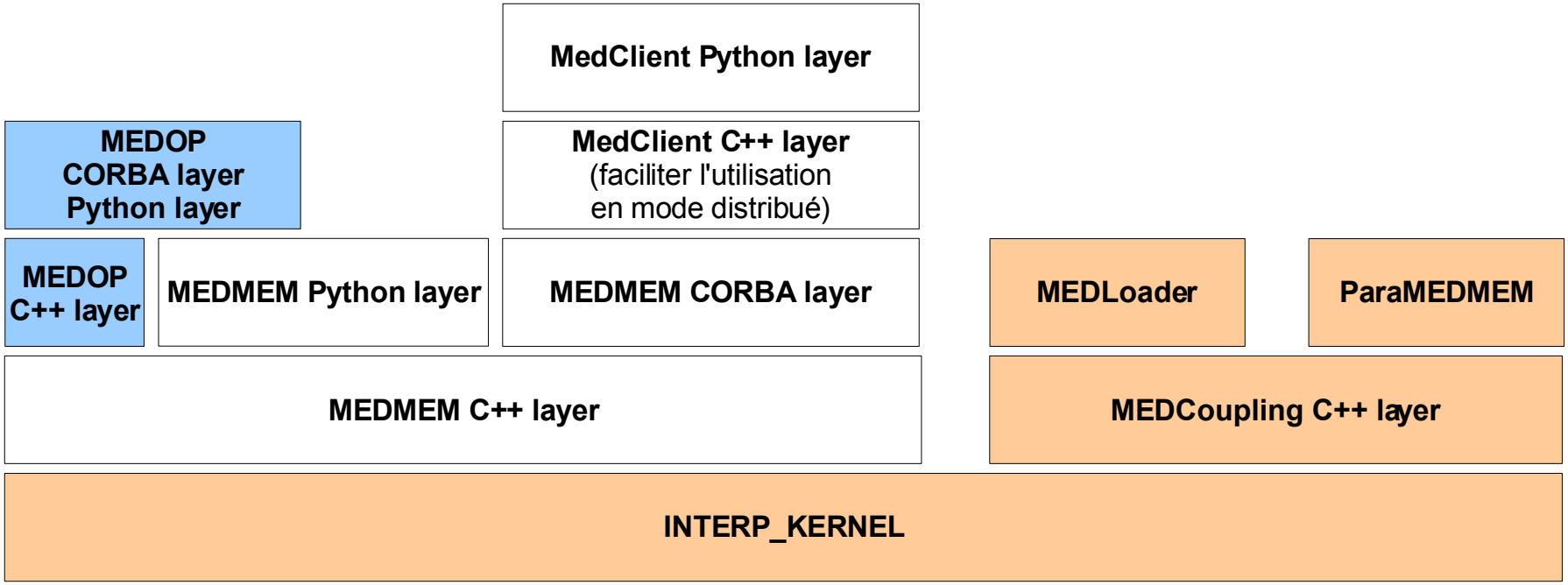
MEDMEM MedOp_i.cxx:

```
SALOME_MED::FIELD_ptr MEDOP_i::add(SALOME_MED::FIELD_ptr f1_ptr, SALOME_MED::FIELD_ptr f2_ptr) {
  // Get the MEDMEM fields embeded in the CORBA servants
  MEDMEM::FIELD<double> * f1 = _getFieldDouble(f1_ptr);
  MEDMEM::FIELD<double> * f2 = _getFieldDouble(f2_ptr);
  // Create the new field using the standard MEDMEM API
  FIELD<double> * field_result = FIELD<double>::add(*f1, *f2);
  // Register the newly created field in the MED data structure
  _med->addField(field_result);
  // Create a CORBA servant to encapsulate this newly create field
  FIELDTEMPLATE_I<double> *field_result_i = new FIELDTEMPLATE_I<double>(field_result);
  // And return a pointer to this CORBA servant
  return field_result_i->_this();
}
```

Implémentation des classes proxy

```
class FieldProxy:
....
    def __init__( self, med_ptr, field_ptr ):
        '''The proxy needs a med servant and the field servant to be initialized'''
        self.setMed(med_ptr)
        self.setField(field_ptr)

    def setMed(self, med_ptr):
        '''The med servant is used to create a MEDOP servant on the server side'''
        self.__medOp_ptr = self.__med_ptr.createMedOperator()

    def __getattr__( self, name ):
        '''Implementation of the pattern proxy'''
        return getattr( self.__field_ptr, name )

    def __add__(self, operande):
        '''Implementation of the operator +'''
        otherField_ptr = operande.__field_ptr
        rfield_ptr = self.__medOp_ptr.add(self.__field_ptr, otherField_ptr)
        return FieldProxy(self.__med_ptr, rfield_ptr)

    def __sub__(self, operande):
        '''Implementation of the operator -'''
        otherField_ptr = operande.__field_ptr
        rfield_ptr = self.__medOp_ptr.sub(self.__field_ptr, otherField_ptr)
        return FieldProxy(self.__med_ptr, rfield_ptr)

    def __mul__(self, operande):
        ...

    def __pow__(self, power):
        rfield_ptr = self.__medOp_ptr.pow(self.__field_ptr, power)
        return FieldProxy(self.__med_ptr, rfield_ptr)

    ...
```

Implantation technique



Limitations actuelles

- La maquette présente les limitations suivantes:
  - Seules les opérations entre champs qui partagent le même support med sont possibles. Ceci est une contrainte imposé par la conception actuelle de MEDMEM.
  - Les opérandes sont des champs MED (composé d'un pas de temps unique) et non pas des séries temporelles
  - Le résultat est calculé sur toutes les composantes et tout le domaine de définition du champs
  - Le résultat d'une opérations est calculé sur toutes les composantes et tout le domaine de définition des champs en opérande
  - Le nom du champ résultat est attribué par une convention (ceci n'est pas vraiment une limitation mais une caractéristique à connaître)
- On note également l'hypothèse suivante:
  - Les données MEDMEM sont supposées être chargées par le composant MED puis référencées dans l'étude SALOME (comme c'est fait aujourd'hui par le module MED).

Plan de travail 2012

- Implantation sur base MEDCoupling
  - Développement du servant CORBA minimal pour les besoins des variables proxy
- Intégration dans le module MED
  - révision de l'interface graphique
  - révision de la structure de donnée de l'étude
- Évolutions fonctionnelles
  - Ajout des opérations d'interpolations (Analyse préliminaire: N. Geimer)
  - Ajout du concept de domaine d'application (portée ou restriction des opérations sur un domaine spatial, temporel, ou un jeu de composantes)
- Évolutions techniques
  - Pouvoir manipuler des champs issus de deux structures MED différentes