SYSTÈME D'ÉVALUATION ET DE GESTION DES RISQUES D'INONDATION EN MILIEU FLUVIAL

PROJET SEGRI

RAPPORTS DE RECHERCHE 2003

Rapport de recherche No R-720-a *Janvier 2004*

Système d'Évaluation et de Gestion des Risques d'Inondation en milieu fluvial

Projet SEGRI

Rapports de recherche 2003

Présenté au

Fonds des Priorités Gouvernementales en Science et en Technologie – volet Environnement (FPGST-E)

15 janvier 2004

Équipe de réalisation

Institut National de la Recherche Scientifique – Eau, Terre et Environnement

Yves Secretan	Professeur, PhD
Eric Larouche	Ingénieur en informatique

Collaborateurs

Renaud Le Boulleur de Courlon	Stagiaire
Samuel Ouellet	Stagiaire
Maxime Derenne	Stagiaire
Kevin Solinski	Stagiaire
Olivier Kaczor	Stagiaire
Francis Larrivée	Stagiaire

Pour les fins de citation : **Secretan Y., Larouche E. & coll. (2003)**.

Système d'Évaluation et de Gestion des Risques d'Inondation en milieu fluvial (SEGRI) : Rapports de recherche 2003. Québec, INRS-Eau, Terre & Environnement. Pagination multiple. (INRS-Eau, Terre & Environnement, rapport de recherche 720 a)

Pour: Fonds des Priorités Gouvernementales en Science et en Technologie – volet Environnement (FPGST-E).

ISBN: 2-89146-516-4

Liste des rapports

RAPPORT #1 : Évaluation d'outils pour compilations automatisées

RAPPORT #2 : Module de visualisation

RAPPORT #3 : Mise en place de la base de données relationnelle

RAPPORT #4 : Mise en place du langage interprété

RAPPORT #5 : Champs-Séries

RAPPORT #6 : Étude des risques d'inondation de la rivière Chaudière

Rapport de développement logiciel

Évaluation d'outils pour compilations automatisées

Présenté par :

Francis Larrivée

30-05-2003

1. Introduction

1.1 Objectifs

Le but de la démarche qui justifie ce rapport était de trouver un outil pour effectuer l'automatisation des compilations dans le cadre du projet Modeleur 2.0.

Les outils qui ont été évalués sont les suivants:

- TMake
- CMake
- JAM
- Boost.Jam

Ces outils seront comparés et une recommandation sera faite, suivant les critères qui sont explicités dans la section suivante.

1.2 Contexte

Il est requis d'automatiser les processus de compilation sous différents compilateurs. Les "makefile" sont des outils très pratiques pour ce genre de travail mais ils sont aussi complexes à mettre en place car leur syntaxe est lourde et leur maintenances est ardue. Ceci explique pourquoi un autre outil est requis.

Afin de bien illustrer l'ensemble de la problématique, voici les critères de sélection pour le choix de l'outil:

- **Facilité d'utilisation:**
  Les fichiers de configuration doivent être faciles à construire et simples de structure.

- **Facilité de maintenance:**
  Les modifications dans le fichier de configuration doivent être faciles à mettre en place. Un développeur sans connaissances préalables doit s'y retrouver facilement.

- **Gestion des dépendances inter modules:**
  Les dépendances et inter-dépendances entre les modules doivent être trouvées et gérées de façon automatique.

- **Intégration avec CVS:**
  L'outil doit pouvoir interagir avec CVS, l'outil de gestion de code source du projet, pour extraire les fichiers qui composent les modules.

- **Gestion des versions:**
  Il doit être possible de gérer la notion de versions des modules via des "tags" (ou étiquettes) qui sont dans CVS.

- **Gestion du mode "debug/release":**
  L'outil doit pouvoir compiler les modes "debug/release".

- **Support des langages :**
  C++
  Fortran
  VB

- **Gestion des compilateurs:**
  Borland 5.0.1
  Borland 5.5.1
  Gnu G++ 3.2
  Gnu G77 3.2
  Visual C++ 7.0
  Intel C++ 7.0
  Intel Fortran 7.0
  Watcom C++ 11.0c
  Watcom Fortran 11.0c

- **Facilité de supporter d'autres "target":**
  La production de fichiers de sortie doit être flexible. Par exemple il doit être possible d'utiliser l'outil avec des "target" de type Web++ ou Doxygen.

- **Possibilité d'extension du langage**
  L'outil choisi doit permettre des modifications et des configurations sur mesure pour répondre aux besoins particuliers des compilateurs.

1.3 Structure

Ce rapport discute des avantages et des désavantages de chacun des outils. Pour chacun d'eux des exemples d'utilisation et de configurations sont présentés. Finalement, une recommandation est faite.

2 Évaluation

2.1 TMake v1.8

Cet outil à été développé par Trolltech. Voici un lien vers le site de TMake: http://www.trolltech.com/download/tmake.html .

TMake est un outil facile à utiliser pour créer et maintenir des fichiers makefile. C'est une tâche ardue de contrôler des fichiers makefile manuellement, particulièrement si le développement s'effectue sur plus d'une plate-forme ou si plus d'un compilateur est employé. TMake automatise et améliore ce processus d'écriture des fichiers makefile.

2.1.1 Avantages

- Utilise un fichier de configuration "*.pro" pour créer les "makefile". Les fichiers sources composant le module doivent être seulement spécifiés dans ce fichier.
- Offre une syntaxe de configuration facile à appliquer dans les fichiers *.pro.
- Supporte la gestion de plusieurs compilateurs sur plusieurs plates-formes. Le choix du compilateur peut être fait en modifiant la variable d'environnement TMakePath.
- Permet d'adapter la configuration de compilateur avec le fichier tmake.conf dans TMake\lib\OS-Compilateur .

2.1.2 Désavantages

- N'offre pas la gestion automatisée des dépendances inter modules.
- Ne supporte que le C++; l'ajout d'un autre langage pourait se faire en modifiant le code source.
- Ne supporte pas tous les compilateurs. L'ajout d'autres compilateurs nécessite la création d'un nouveau répertoire "OS-Compilateur" à l'intérieur duquel nous devrons créer les fichiers de configuration pour supporter le nouveau compilateur.

- N'offre aucune possibilité d'extension du langage.
- Nécessite l'exécution manuelle de TMake dans chaque répertoire ou sous répertoire.

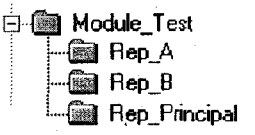
2.1.3 Exemples

- **Fichier de configuration.**

  Dans cet exemple, trois répertoires sont utilisés, le Rep_Principal, le Rep_A et le Rep_B. Il y a création d'un exécutable dans le répertoire Rep_Principal de type win32 console, appelé Principal, à partir du fichier source principal.cpp. Il dépendra de la librairie libA située dans le répertoire Rep_A, qui elle dépendra de la libB situé dans le répertoire Rep_B. Les libA et libB sont de type dll et sont crées avec les fichiers sources (*.cpp et *.h) de leurs répertoires.

  Principal.pro dans le répertoire Rep_Principal

```
SOURCES   = principal.cpp
TARGET    = Principal
CONFIG    = Console
LIBS      = ..\Rep_A\libA.lib
```



  A.pro dans le répertoire Rep_A

```
HEADERS   = a.h
SOURCES   = a.cpp
TARGET    = libA
CONFIG    = dll
LIBS      = ..\Rep_B\libB.lib
```

  B.pro dans le répertoire Rep_B

```
HEADERS   = b.h
SOURCES   = b.cpp
TARGET    = libB
CONFIG    = dll
```

- **Commande Line**

  Une fois les fichiers *.pro créés dans chaque répertoire, il faut créer un makefile dans les répertoires, Rep_Principal, Rep_A et Rep_B. Pour ce faire, exécuter TMake dans chaque répertoire, de la manière suivante:

  tmake –o makefile *.pro

Ensuite exécuter le "make" dans chacun de ces répertoires, dans l'ordre inverse des dépendances. Dans le cas cité en exemple, l'ordre serait: make dans le Rep_B, make dans le Rep_A et make dans le Rep_Principal.

- **TMakePath**

  Pour utiliser Borland 5.5.1, mettre la variable d'environnement :
  TMAKEPATH = "C:\program files\tmake\lib\win32-borland".

2.1.4 Conclusion

TMake est un très bon outil pour des modules indépendants car il est simple d'utilisation et facile de maintenance.

Pour le projet modeleur 2.0, les modules sont complexes et contiennent des dépendances. Ceci élimine à toute fin pratique TMake de la liste de possibilités puisqu'il ne gère pas les dépendances automatiquement.

2.2 CMake v1.6.6

Cet outil à été développé par Kitware. Voici un lien vers le site de CMake: http://www.cmake.org/HTML/Index.html .

CMake produit des fichiers makefile qui peuvent être employés avec certains compilateurs. Il permet de supporter des environnements complexes qui tiennent compte des dépendances inter modules.

2.2.1 Avantages

- Utilise un fichier de configuration CMakeLists.txt pour créer des "makefile". Les fichiers sources composant le module doivent être spécifiés ainsi que les liens de dépendances. Ceci simplifie la gestion des "makefile".
- Offre une syntaxe de configuration simple à appliquer dans les fichiers CMakeLists.txt.
- Offre la gestion automatisée des dépendances inter modules.
- Permet de supporter d'autres "target" via la commande ADD_CUSTOM_COMMAND.
- Permet l'utilisation du mode debug/release via la commande CMAKE_BUILD_TYPE.
- Offre un bon support technique sur le groupe de discussion "mailling list".

2.2.2 Désavantages

- Ne supporte que le C++, l'ajout d'un autre langage pourrait se faire en modifiant le code source.
- Offre un nombre de compilateurs limité (VS6, VS7, Borland, Gnu). L'ajout d'un nouveau compilateur est complexe. Il faudrait créer un nouveau fichier OS-compiler.cmake.
- N'offre aucune possibilité d'extension du langage.

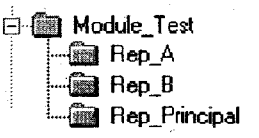
- Offre une documentation faible et des exemples difficiles à trouver.

2.2.3 Exemples

- **Fichier de configuration.**

  Dans cet exemple, trois répertoires sont utilisés le Rep_Principal, le Rep_A et le Rep_B. Il y a création d'un exécutable dans le répertoire Rep_Principal, appelé Principal, à partir du fichier source principal.cpp. Il dépendra de la librairie libA située dans le Rep_A, qui elle dépendra de la libB située dans le répertoire Rep_B. La libA est construite avec la commande ADD_LIBRARY(libA SHARED a) qui produit une libA de type dll(SHARED) à partir du fichier a.cpp.

  CMakeLists.txt dans le répertoire Rep_Principal
  ```
  ADD_EXECUTABLE(Principal principal)
  TARGET_LINK_LIBRARIES (Principal libA )
  SUBDIRS(../Rep_A )
  ```



  CMakeLists.txt dans le répertoire Rep_A
  ```
  ADD_LIBRARY(libA SHARED a )
  TARGET_LINK_LIBRARIES (libA libB)
  SUBDIRS(../Rep_B )
  ```

  CMakeLists.txt dans le répertoire Rep_B
  ```
  ADD_LIBRARY(libB SHARED b )
  ```

  N.B. Les extensions (.cpp) ne doivent pas être spécifiées.

- **Commande Line**

  Une fois les fichiers CMakeLists.txt créés dans chaque répertoire, exécuter les opérations suivantes à partir du répertoire Rep_Principal (exemple pour Borland)

  ```
  cmake –G"Borland Makefiles"
  make
  ```

- **Path**

  Puisqu'il existe une version win32 et cygwin de CMake, il est important de modifier le path en fonction du CMake que l'on veut utiliser.

  Ex : Pour utiliser Borland (bcc32)
  ```
  Mettre path  = "C:\borland\bcc55\bin;%path%"
  ```

Ex: Pour utiliser Gnu
Mettre path = "C:\cygwin\bin;%path%"

2.2.4 Conclusion

CMake est un outil de compilation automatisée simple et efficace. Toutefois, il n'est pas facile à configurer ou à modifier. Ceci en fait un outil de qualité pour de petits projets, qui ne nécessitent pas de configurations qui vont au delà de celles proposées par défaut dans CMake.

2.3 JAM v2.5

Cet outil à été développé par Perforce. Voici un lien vers le site de JAM: http://www.perforce.com/jam/jam.html .

Jam est un outil de compilation automatisée en remplacement d'un "make" standard. Il rend les compilations simples faciles et les compilations complexes maniables. Il compile les fichiers récursivement à partir des fichiers sources en utilisant les informations de dépendances et les expressions contenues dans le fichier "Jambase", qui est écrit dans le langage que Jam interprète. Jam utilise aussi le fichier "Jamfile", écrit par l'usager dans chacun des modules, pour connaître les fichiers sources et les "target" à construire.

2.3.1 Avantages

- Utilise un fichier de configuration "Jamfile" où les fichiers sources composant le module doivent être indiqués ainsi que les liens de dépendancé. Construit la "target" sans passer par les "makefile".
- Offre la gestion automatisée des dépendances inter modules.
- Supporte le C++ et le Fortran, pourrait supporter VB.
- Peut utiliser un autre fichier "Jamrules", pour l'ensemble du projet, avec des règles spéciales communes qui remplacent ou modifient celles de la "Jambase".
- Offre une bonne documentation.
- Fonctionne en mode debug (voir la documentation "jam.html").
- Peut supporter différents types de fichiers de sortie ("target").
- Extension du langage possible et bien faite.

2.3.2 Désavantages

- Ne supporte pas tous les compilateurs. Le support de nouveaux compilateurs nécessite l'ajout d'une configuration dans le fichier "Jamrules".

- Peut nécessiter la création d'un fichier "Jamrules" complexe pour effectuer certaines des tâches.

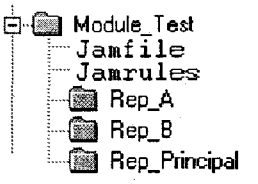
2.3.3 Exemples

- **Fichier de configuration de chacun des modules**

  Dans cet exemple, trois répertoires sont utilisés, le Rep_Principal, le Rep_A et le Rep_B. Il est nécessaire de définir un répertoire racine appelé "TOP" pour configurer l'ensemble du projet via la commande "SubDir TOP ;". Le fichier "Jamrules" explique les règles crées par l'utilisateur (ex: la règle Dll) .Il y a création d'un exécutable dans le répertoire Rep_Principal, appelé Principal, à l'aide de la commande "Main", à partir du fichier source principal.cpp. Il dépendra de la librairie libA située dans le Rep_A, qui elle dépendra de la libB située dans le répertoire Rep_B. La libA est construite avec la commande Dll qui produit une libA de type dll à partir du fichier a.cpp.

  -Jamfile dans le répertoire Module_Test:

  ```
  SubDir TOP ;
  SubInclude TOP Rep_Principal;
  ```

  -Jamrules dans le répertoire Module_Test:

  ```
  rule Dll{Voir annexe pour le code}
  ```

  Jamfile dans le répertoire Rep_Principal

  ```
  SubDir TOP Rep_Principal ;
  Main Principal : principal.cpp ;
  LinkLibraries Principal : libA ;
  SubInclude TOP Rep_A ;
  ```

  Jamfile dans le répertoire Rep_A

  ```
  SubDir TOP Rep_A ;
  Dll libA : a.cpp ;
  LinkLibraries libA : libB ;
  SubInclude TOP Rep_B ;
  ```

  Jamfile dans le répertoire Rep_B

  ```
  SubDir TOP Rep_B ;
  Dll libB : b.cpp ;
  ```

- **Commande Line**

Une fois les "Jamfile" et le "Jamrules" crées dans leur répertoire, lancer seulement Jam à partir du répertoire Module_Test(TOP) de la manière suivante (exemple pour Borland:).

```
set NT=40
Set BCCROOT=C:/Borland/bcc55/bin
Jam
```

2.3.4 Conclusion

Jam est un outil très intéressant. Il permet d' atteindre le niveau d'automatisation de compilation recherché. Toutefois, pour y arriver, il doit être configuré car il n'offre pas par défaut toutes les fonctionnalités requises.

2.4 Boost.Jam v3.1.4

Cet outil à été développé par Boost. Voici un lien vers le site de Boost.Jam:
http://www.boost.org/tools/build/jam_src/index.html - introduction

Boost.Jam est une amélioration de Jam et est aussi un outil de compilation automatisée en remplacement d'un "make" standard. Il rend les compilations simples faciles et les compilations complexes maniables. La version 3.1.4 de Boost.Jam est basée sur la version 2.4 de Jam. Elle contient des améliorations significatives pour faciliter l'utilisation de Jam.

2.4.1 Avantages

- Utilise un fichier de configuration "Jamfile" où les fichiers sources composant le module doivent être indiqués ainsi que les liens de dépendance. Construit la "target" sans passer par les "makefile".
- Offre la gestion automatisée des dépendances inter modules.
- Supporte le C++ et le Fortran, pourrait supporter VB.
- Peut utiliser un autre fichier "Jamrules", pour l'ensemble du projet, avec des règles spéciales communes qui remplacent ou modifient celles de la "Jambase".
- Offre une bonne documentation.
- Fonctionne en mode debug (voir la documentation "jam.html").
- Peut supporter différents types de fichiers de sorti ("target").
- Offre une extension du langage de Jam.
- Offre plusieurs configurations prédéfinies avantageuses.
- Offre un bon support pour la gestion des compilateurs.
- N'offre pas de support pour la gestion des versions dans les configurations standards. Mais peut-être étendu.

2.4.2 Désavantages

- N'offre pas de support avec CVS dans les configurations standards.

2.4.3 Exemples

- **Fichier de configuration de chacun des modules**

  Dans cet exemple, trois répertoires sont utilisés, le Principal, le libA et le libB. Le fichier "Jamrules" explique les règles crées par l'utilisateur (ex: la règle Module). Il y a création d'un exécutable dans le répertoire Principal, appelé Principal, à l'aide de la règle "Module". Il dépendra de la librairie libA située dans le répertoire libA, qui elle dépendra de la libB située dans le répertoire libB.

  -Jamfile dans le répertoire Module_Test:

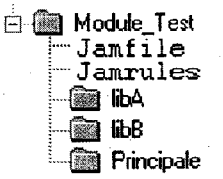
```
project-root ;
local l_lst_dep =
        Principale
        libA
        libB
;
Modules $(dd) : $(l_lst_dep) ;
```

  -Jamrules dans le répertoire Module_Test:

```
rule Modules ( module * : dependencies * ) { Voir annexe pour le code }
rule ModuleSrcLst{ Voir annexe pour le code }
```

  Jamfile dans le répertoire Principal

```
local l_module = Principale ;
Module <exe>$( l_module)   : # sources
    [ ModuleSrcLst $(l_module)  ]
  : # dependances
    <dll>libA
  : # lib
  : # requirements
    <include>../libA
  : # local-build
  ;
```



  Jamfile dans le répertoire libA

```
local l_module = libA ;
Module <dll>$(l_module)   : # sources
    [ ModuleSrcLst $(l_module) ]
  : # dependances
```

```
        <dll >libB
    : # lib
    : # requirements
      <include>../libB
    : # local-build
    ;
Jamfile dans le répertoire libB
    local l_module = libB ;
    Module <dll>$(l_module)
      : # sources
        [ ModuleSrcLst $(l_module) ]
      : # dependances
      : # lib
      : # requirements
      : # local-build
      ;
```

2.4.4 conclusion

Boost.Jam est un outil des plus intéressants. Il permet d'atteindre un niveau d'automatisation de compilation qui peut répondre à plusieurs besoins. Il est modifiable et configurable.

3 Conclusion

3.1 Résumé

TMake, CMake, Jam et Boost.Jam sont tous des outils qui offrent un lot de fonctionnalités intéressantes. Voici donc leur résumé.

TMake ne fait pas tout le travail exigé pour être notre choix. C'est principalement parce qu'il ne permet pas de gérer automatiquement les interdépendances entre les modules. De plus, les possibilités d'extension du langage sont complexes car nous devons modifier le code source .

CMake est adéquat pour un travail simple. Il est l'outil idéal pour les travaux d'automatisation de petite envergure. Dans le cadre du projet modeleur 2.0, nous avons besoin d'un outil qui se prête plus facilement aux modifications pour répondre aux besoins particuliers de chaque compilation.

Jam possède les fonctionnalités de CMake, mais avec un potentiel de modification supérieur et plus adapté à nos besoins de compilation. Toutefois, il est relativement complexe de l'adapter à nos besoins.

Boost.Jam possède les fonctionnalités de Jam, mais avec un potentiel supérieur car plusieurs modifications ou améliorations y sont déjà faites.

3.2 Tableau comparatif

Voici un tableau de comparaison des outils:

	TMake	CMake	JAM	Boost.Jam
Facilité d'utilisation	++	+	+	+-
Facilité de maintenance	++	+	+-	+-
Gestion des dépendances inter modules	--	++	++	++
Intégration avec CVS	N/A	N/A	N/A	N/A
Gestion des versions	N/A	N/A	N/A	+
Support en mode "debug/release"	N/A	+	++	++
Support des langages	-	-	++	++
Gestion des compilateurs	+	-	+	+
Facilité de supporter d'autres "target":	N/A	+	+	++
Possibilité d'extension du langage	--	--	+	++

3.3 Perspectives futures

Pour le critère d'intégration avec CVS, aucun des outils n'a répondu, simplement et intuitivement, à nos besoins. Dans les cas de TMake et CMake, aucun test n'a été effectué à ce sujet. Jam et Boost.Jam sont configurables, ils devraient donc être en mesure de répondre à nos besoins concernant CVS.

Après une analyse détaillée, c'est donc Boost.Jam qui à été sélectionné pour être l'outil d'automatisation pour les compilations.

4 Annexes

4.1 Jam

Jamrules

```
SUFSHR = .dll ;

rule Dll
{
 local _dll = $(<)$(SUFSHR) ;
 local _lib = $(_dll:B).lib ;

 if $(BCCROOT)
 {
       local DLL_LINKFLAGS = -tWD ;
       LINKFLAGS on $(_dll) += $(DLL_LINKFLAGS) ;
 }

 MainFromObjects $(_dll) : $(>:S=$(SUFOBJ)) ;
 Objects $(>) ;

 if $(BCCROOT)
 {
       ImportLibrary $(_lib) : $(_dll) ;
 }

 MakeLocate $(_lib) : $(LOCATE_TARGET) ;
}

rule ImportLibrary
{
 local _lib = $(<) ;
 local _dll = $(>) ;

 Depends  exe : $(_lib) ;
 Depends $(_lib) : $(_dll) ;
 Implib  $(_dll) ;
}

actions Implib
{
 $(BCCROOT)/implib -c $(<:S=$(SUFLIB)) $(<) ;
}


rule LinkLibraries
{
 local _e = [ FAppendSuffix $(<) : $(SUFEXE) ] ;
 Depends $(_e) : $(>:S=$(SUFLIB)) ;
 NEEDLIBS on $(_e) += $(>:S=$(SUFLIB)) ;

 local _d = [ FAppendSuffix $(<) : $(SUFSHR) ] ;
 Depends $(_d) : $(>:S=$(SUFLIB)) ;
 NEEDLIBS on $(_d) += $(>:S=$(SUFLIB)) ;
}

rule SubIncludeOnce
{
```

```
        local i ;
        local include_marker = included ;

        # value of include_marker is the concatenated directory names in the
        #   path to the directory being included
        for i in $(<[2-])
        {
            include_marker = $(include_marker)_$(i) ;
        }

        # if the variable whose name is the value of include_marker does not
        #   exist, then we know we haven't included that directory yet.
        if ! $($(include_marker))
        {
            # Do not include more than once
            $(include_marker) = TRUE ;
            SubInclude $(<) ;
        }
    }
```

4.2 Boost.Jam

Jamrules

```
# Boost-Jam Variables
PRELIB = "" ;           # Pas de préfixe aux noms de librairies
BIN_DIRECTORY = tgt ;   # Sous-répertoire des "targets"

# GREEN-Jam Variables
JAMSUBDIR       ?= build ;
DEFAULT_VERSION ?= derniere ;
GREEN_SRC_LST_EXTENSIONS = *.c *.cpp *.for *.f *.rc ;

# Configuration
INTELC ?= C:/Progra~1/Intel/Compiler70/IA32 ;


include "$(JAMSUBDIR)/Jamrules_FTN.txt" ;
include "$(JAMSUBDIR)/Jamrules_CVS.txt" ;
include "$(JAMSUBDIR)/Jamrules_Modules.txt" ;
```

Jamrules_FTN.txt

```
#==============================================================================
#  Sommaire:   Fortran
#  Fichier:
#==============================================================================
#  20-04-2003  Yves Secretan     Version initiale
#==============================================================================


#==============================================================================
#  Sommaire:
#
#  Description:
#      La règle <code>UserObject</code> redéfinit la règle de base. Elle
est
#     appelée pour       .
#
#  Syntaxe:
#
#==============================================================================
rule UserObject
```

```
{
   switch $(>)
   {
      case *.for  : Fortran $(<) : $(>) ;
      case *      : ECHO "unknown suffix on" $(>) ;
   }
}

rule Fortran
{
    DEPENDS $(<) : $(>) ;
    FTN-platform-specifics $(<) : $(>) ;
    FTN-action $(<) : $(>) ;
}

rule FTN-platform-specifics
{
   # If the compiler's -o flag doesn't work, relocate the .o
    if $(RELOCATE)
    {
        CcMv $(<) : $(>) ;
    }

#    local _h = $(SEARCH_SOURCE) $(HDRS) ;

#    if $(VMS) && $(_h)
#    {
#        SLASHINC on $(<) = "/inc=(" $(_h[1]) ,$(_h[2-]) ")" ;
#    }
#    else if $(MAC) && $(_h)
#    {
#        local _i _j ;
#        _j = $(_h[1]) ;
#        for _i in $(_h[2-])
#        {
#            _j = $(_j),$(_i) ;
#        }
#        MACINC on $(<) = \"$(_j)\" ;
#    }
}
```

Jamrules_CVS.txt

```
#=============================================================================
#  Sommaire:   CVS
#  Fichier:
#=============================================================================
#  20-04-2003  Yves Secretan     Version initiale
#=============================================================================

   GREEN_CVSROOT_LIB_INTERNES                                              =
":ext:secretyv@caxipi:/cygdrive/c/cvs/lib_internes" ;
   GREEN_CVSROOT_LIB_EXTERNES                                              =
":ext:secretyv@caxipi:/cygdrive/c/cvs/lib_externes" ;

#=============================================================================
#  Sommaire:   CVS
#
#  Description:
#
#  Syntaxe:
#     CVS liste_des_noms_modules_cvs
```

```
#
#=============================================================================
rule CVS
{
   Depends first : $(<) ;

   for m in $(>)
   {
      CVS_Module $(<) : $(m) ;
   }

   local l_d on $(<) = [ FSubDirPath TOP $(<) ] ;
   local l_c on $(<) = [ CygwinPath $(l_d) ] ;
   NOCARE $(l_c) ;
   echo ----> $(l_d) ;
   CVS_CheckOut $(<) : $(l_c) ;
}

#=============================================================================
#  Sommaire:   CVS
#
#  Description:
#
#  Syntaxe:
#     CVS liste_des_noms_modules_cvs
#
#=============================================================================
rule CVS_Module
{
   Depends $(<) : $(>) ;
   local l_d on $(<) = [ FSubDirPath TOP $(<) ] ;
   CVS_CheckOut $(<) : [ CygwinPath $(l_d) ] ;
}

#=============================================================================
#  Sommaire:   CVS
#
#  Description:
#
#  Syntaxe:
#     CVS liste_des_noms_modules_cvs
#
#=============================================================================
rule CygwinPath
{
   local l_0 = /::DummyBase:: $(<[1]) ;
   local l_p = $(l_0:J="/") ;
   local l_c = [ CygwinSubPath $(l_p:P) ] $(l_p:B) ;
   return $(l_c:J="/") ;
}

rule CygwinSubPath
{
   local l_f ;
   switch $(<:B)
   {
      case ::DummyBase:: : l_f = /cygdrive ;
      case [aA]:         : l_f = [ CygwinSubPath $(<:P) ] a ;
      case [bB]:         : l_f = [ CygwinSubPath $(<:P) ] b ;
      case [cC]:         : l_f = [ CygwinSubPath $(<:P) ] c ;
      case [dD]:         : l_f = [ CygwinSubPath $(<:P) ] d ;
      case [eE]:         : l_f = [ CygwinSubPath $(<:P) ] e ;
      case [fF]:         : l_f = [ CygwinSubPath $(<:P) ] f ;
```

```
            case [gg]:          : l_f = [ CygwinSubPath $(<:P) ] g ;
            case [hH]:          : l_f = [ CygwinSubPath $(<:P) ] h ;
            case [iI]:          : l_f = [ CygwinSubPath $(<:P) ] i ;
            case [jJ]:          : l_f = [ CygwinSubPath $(<:P) ] j ;
            case [kK]:          : l_f = [ CygwinSubPath $(<:P) ] k ;
            case [lL]:          : l_f = [ CygwinSubPath $(<:P) ] l ;
            case [mM]:          : l_f = [ CygwinSubPath $(<:P) ] m ;
            case [nN]:          : l_f = [ CygwinSubPath $(<:P) ] n ;
            case [oO]:          : l_f = [ CygwinSubPath $(<:P) ] o ;
            case [pP]:          : l_f = [ CygwinSubPath $(<:P) ] p ;
            case [qQ]:          : l_f = [ CygwinSubPath $(<:P) ] q ;
            case [rR]:          : l_f = [ CygwinSubPath $(<:P) ] r ;
            case [sS]:          : l_f = [ CygwinSubPath $(<:P) ] s ;
            case [tT]:          : l_f = [ CygwinSubPath $(<:P) ] t ;
            case [uU]:          : l_f = [ CygwinSubPath $(<:P) ] u ;
            case [vV]:          : l_f = [ CygwinSubPath $(<:P) ] v ;
            case [wW]:          : l_f = [ CygwinSubPath $(<:P) ] w ;
            case [xX]:          : l_f = [ CygwinSubPath $(<:P) ] x ;
            case [yY]:          : l_f = [ CygwinSubPath $(<:P) ] y ;
            case [zZ]:          : l_f = [ CygwinSubPath $(<:P) ] z ;
            case *              : l_f = [ CygwinSubPath $(<:P) ] $(<:B) ;
        }
        return $(l_f) ;
    }

    #=============================================================================
    #  Sommaire:   CVS
    #
    #  Description:
    #
    #  Syntaxe:
    #     CVS liste_des_noms_modules_cvs
    #
    #=============================================================================
    actions CVS_CheckOut
    {
        echo cvs -d $(GREEN_CVSROOT_LIB_INTERNES) checkout -d $(>) -N $(<)
    }

    actions CVS_Update
    {
        echo cvs -d $(GREEN_CVSROOT_LIB_INTERNES) update $(<) ;
    }
```

Jamrules_Modules.txt

```
    #=============================================================================
    #  Sommaire:   Modules
    #  Fichier:
    #=============================================================================
    #  20-04-2003  Yves Secretan     Version initiale
    #=============================================================================

    #=============================================================================
    #  Sommaire:   Modules
    #
    #  Description:
    #     La règle <code>Modules</code> définit l'arborescence pour le module
    #     et ses dépendances.
    #
    #  Syntaxe:
    #     Modules $(l_module) : $(l_dependencies) ;
```

```
#
#=============================================================================
rule Modules ( module * : dependencies * )
{
   if $(module)
   {
      subproject $(module) ;
   }

   for m in $(dependencies)
   {
      local target-root = [ xtr-target $(m) ] ;
      subinclude [ join-path $(target-root) $(JAMSUBDIR) ] ;
   }
}

#=============================================================================
#  Sommaire:   Module
#
#  Description:
#       La règle <code>Module</code> définit la règle générique utilisée
pour un
#     module. Elle encapsule les règles exe, dll et lib pour tenir compte
des
#     version. De manière générique un target est défini par:
#        <[exe|dll|lib]>target[<version>nom_version]
#
#  Syntaxe:
#
#=============================================================================
rule Module ( target : sources * : dependencies * : libs * : requirements
* : local-build * )
{
   local target-type    = [ ungrist $(target:G) ] ;
   local target-root    = [ xtr-target  $(target) ] ;
   local target-version = [ xtr-version $(target) ] ;

   local decorated-dependencies ;
   for local m in $(dependencies)
   {
      local g = $(m:G) ;
      local t = [ xtr-target  $(m) ] ;
      local v = [ xtr-version $(m) ] ;
      local p = [ join-path .. $(t) $(t) ] ;
      decorated-dependencies += $(p:G=$(g))$(v) ;
   }

   Modules $(target-root) : $(dependencies) ;

   $(target-type) $(target-root)$(target-version)
      : $(sources) $(decorated-dependencies)
      : $(libs) $(requirements)
      : $(local-build) ;
}

#=============================================================================
#  Sommaire:   Génère la liste des fichiers sources
#
#  Description:
#       La règle <code>ListeSource</code> génère la liste des fichiers
sources
#       d'un module. Le chemin d'accès intégré au nom est relatif par
rapport
```

```
    #       au répertoire du module. Les fichiers sources doivent se trouver
dans
    #      le sous-répertoire <code>source</code> du module.
    #      <p>
    #      La règle utilise la variable globale GREEN_SRC_LST_EXTENSIONS.
    #
    #  Syntaxe:
    #      local l_lst_src on $(l_module) = [ ListeSources $(l_module) ] ;
    #
    #=============================================================================
    rule ModuleSrcDir
    {
       local l_src_dir on $(<) = source ;
    #   if $(gGREEN_TARGET_VERSION($(module)))
    #   {
    #      l_src_dir += $(gGREEN_TARGET_VERSION($(module))) ;
    #   }
    #   echo ModuleSrcDir  $(l_src_dir:J=/) ;
       return $(l_src_dir:J=/) ;
    }

    rule ModuleSrcLst
    {
       local l_src_dir on $(<) = [ ModuleSrcDir ] ;
       local   l_src_lst   on   $(<)   =   [   GLOB   $(<)/$(l_src_dir)   :
$(GREEN_SRC_LST_EXTENSIONS) ] ;
    #   echo ModuleSrcLst  $(l_src_lst:D=source) ;
       return $(l_src_lst:D=source) ;
    }
```

Rapport de développement logiciel

Module de visualisation

Présenté par :

Maxime Derenne

07-20-2003

Rapport de développement logiciel

Module de visualisation

Présenté par :

Maxime Derenne

07-10-2003

1 Introduction

1.1 Contexte

Ce projet a pour but de remplacer l'ancien système graphique utilisé dans Modeleur 1.0. Il y a lieu d'implanter un nouveau système car l'ancien ne peut difficilement se voir ajouter des fonctionnalités supplémentaires.

Voici certaines des fonctionnalités recherchées justifiant le développement d'un nouveau module de visualisation: affichage 3D, gestion de plans et des couches d'information, affichage de bitmaps en arrière plan, « picking » en 2D / 3D, capacité de faire des pans et des zooms de manière conviviale.

1.2 Structure

Tout d'abord, le sujet de la recherche d'un outil graphique sera abordé. Plus précisément, il sera discuté de l'analyse des besoins, des critères de sélection ainsi que des raisons qui ont poussé le choix de VTK comme outil graphique.

En second lieu, VTK sera décrit en détails: son installation, l'introduction de ses fonctions, le contrôle de l'affichage, l'affichage d'images, le picking et la variation de la couleur en fonction d'un scalaire.

Par la suite, un prototype liant VB et VTK sera décrit à l'aide de l'énumération des contraintes et de l'approche retenue.

Finalement, le design du module de visualisation sera présenté, les rôles seront décrits, les lacunes et fonctionnalités manquantes seront énumérées.

1.3 Architecture logicielle

1.3.1 Contrainte sur le design

Le design du module de visualisation avait pour contrainte principale de devoir s'intégrer à l'architecture de Modeleur 2.0 basée sur un bus logiciel commun à tous les modules. Le module de visualisation doit donc être en mesure de communiquer avec les autres modules (le GUI et la BD) uniquement via des événements échangés via le bus.

1.3.2 Avantages du design retenu

Voici les deux principaux avantages que comporter le design retenu:

1.3.3 Possibilité de substituer l'engin graphique

Il est aisé de substituer une implémentation d'un algorithme de représentation par une autre. Il serait donc relativement simple de changer l'outil graphique actuel (VTK) pour un nouvel outil graphique sans avoir à détruire le code existant.

1.3.4 Facilité pour l'ajout des couches

Il est aisé d'ajouter une nouvelle couche d'affichage. Le couplage est minimal avec les autres entités du design. L'ajout d'une nouvelle couche se fait donc sans avoir à modifier plusieurs classes.

2 Recherche d'un outil graphique

2.1 Résumé de l'analyse des besoins

La recherche d'un outil graphique ne peut être faite sans évaluer à priori les fonctionnalités que l'outil aura à réaliser. Cette section résume donc les différents besoins qui ont été énumérés avant même que la recherche ne débute. Plusieurs de ces besoins seront par la suite directement impliqués dans l'élaboration des critères de sélection.

Tout d'abord, l'acquisition d'un outil de haut niveau serait un atout à ne pas négliger. Ce dernier permettra de développer plus rapidement et plus simplement les fonctionnalités désirées et réduira ainsi le temps de développement et de maintenance de la partie graphique du logiciel.

En second lieu, il est impossible de parler de système graphique sans parler des différentes transformations qui lui sont associées. Il s'agit ici de la rotation, de la translation et des zooms avant et arrière. Toutes ces transformations devront faire partie des fonctionnalités supportées par l'outil choisi.

Il est également indispensable d'être en mesure de faire du picking sur les différents éléments interactifs du système.

2.2 Critères de sélection

Une fois la recherche des outils terminée, une liste de critères a été élaborée.

Voici le contenu de cette liste :

- Outil de haut niveau.
- Gratuit ou peu cher.
- Open source.
- Indépendant de la plate-forme.
- Indépendant du système de fenêtrage.
- Permet de faire du picking.
- Permet la sauvegarde du résultat sous les formats BMP et PostScript.
- Permet l'affichage de texte.
- Permet de mixer du 2D avec du 3D.
- Support disponible. (Exemples, Livres, Mailing-List, Tutorial)
- Possibilité de court-circuiter le pipeline d'affichage.
- Fonctionne bien sous Windows ou par l'intermédiaire de Cygwin.
- Possibilité de faire porter de l'information à la forme, à la couleur ou à l'épaisseur.

2.3 Raisons pour le choix de VTK comme outil graphique

L'outil Visualization Toolkit (VTK) a été choisi pour ses nombreuses qualités. En effet, VTK est indépendant de la plate-forme et du système de fenêtrage. Il permet la sauvegarde du résultat sous les formats BMP et PostScript. Il est extensible, permet d'afficher du texte et capable de mixer des éléments 2D avec des éléments 3D. Il permet de faire du picking très facilement. La caméra et les sources lumineuses sont gérées de manière implicite. Une nouvelle version du produit est compilée toutes les nuits. De plus, VTK est gratuit et Open source.

VTK a toutefois quelques désavantages non négligeables. Il est très lourd et peu rapide.

3 Visualization Toolkit (VTK)

3.1 Installation

L'installation de VTK sous Windows peut être faite de deux façons. La première manière, la plus simple, consiste à utiliser le programme d'installation disponible sur le CD fourni avec le livre *The VTK User's Guide* ou sur le site officiel de Kitware situé à l'adresse http://public.kitware.com/VTK/get-software.php. Cette approche installe la version 4.2 pré-compilée du produit. La seconde manière consiste à compiler soit-même VTK. CMake doit donc être utilisé pour générer un projet compatible avec le compilateur C++ qui sera utilisé pour la compilation. Il permet également de spécifier divers options de compilation. Une fois ces étapes réalisées, il s'agit ensuite de compiler le projet préalablement généré. Cette approche est nécessaire pour installer une version plus récente que 4.2 puisqu'il n'y a pas de programme d'installation pour les dernières versions.

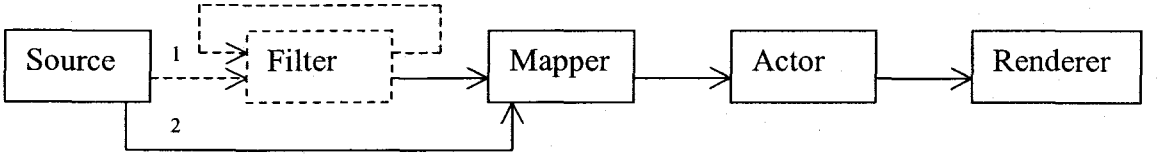
Il est possible de trouver plus d'informations sur les deux approches d'installation aux pages 8 à 13 du livre *The VTK User's Guide*.

3.2 Introduction de son fonctionnement

VTK est un système très vaste et il peut être facile de s'y perdre. Cette section a donc été rédigée pour faciliter et accélérer la compréhension d'un nouvel utilisateur.

VTK est basé sur le paradigme par flux de données. Dans ce paradigme, les données circulent dans un réseau de modules. Ces modules sont en fait des algorithmes qui modifient les données qui entrent dans ceux-ci.

Le diagramme suivant illustre la structure typique d'un programme utilisant VTK :



Voici la signification de chacune des composantes du diagramme :

Source : Source correspond à la source des données caractérisant l'objet à afficher. La sortie d'une Source peut être utilisée comme entrée dans un filtre ou directement dans un Mapper. Les classes vtkConeSource, vtkSphereSource et vtkUnstructuredGrid sont tous des exemples de Source.

Filter : Filter correspond à un filtre qui traite les données qui entrent dans ce dernier pour obtenir des données modifiées en sortie. Un filtre peut avoir une ou plusieurs entrées et une ou plusieurs sorties. Les filtres ne sont pas des éléments essentiels au bon fonctionnement du pipeline VTK. La classe vtkShrinkFilter est un exemple de filtre.

Mapper : Mapper sert à interpréter les données qui entrent dans celui-ci. Cette étape du pipeline convertit les données en primitives graphiques comme des points, des lignes et des polygones.

Actor : Actor correspond à l'élément de la scène. C'est avec ce dernier qu'il est possible d'interagir, de positionner et de modifier la couleur et l'opacité de l'objet à afficher.

Renderer : Renderer est la scène contenant tous les acteurs. Il s'agit ici d'un viewport qui sera contenu dans une fenêtre. Une fenêtre peut ainsi contenir plusieurs Renderer.

Les éléments en pointillé dans le schéma sont facultatifs.

3.3 Gestion de la mémoire

La gestion de la mémoire est faite par compteur de référence. Cette approche a été retenue pour minimiser l'utilisation de la mémoire. Il est donc intéressant de conserver ce concept bien en tête lors de l'écriture de code utilisant VTK. Il est possible, comme dans l'exemple suivant, de créer une seule Source et un seul Mapper pour plusieurs acteurs différents.

Exemple de gestion de la mémoire efficace :

```
vtkConeSource *cone = vtkConeSource::New();
cone->SetResolution( 10 );

vtkPolyDataMapper *mapper = vtkPolyDataMapper::New();
mapper->SetInput( cone->GetOutput() );

// Ces deux acteurs utiliseront le même cone et le même mapper.
vtkActor *actor1 = vtkActor::New();
actor1->SetMapper( mapper );
vtkActor *actor2 = vtkActor::New();
actor2->SetMapper( mapper );
[...]
actor2->Delete();
actor1->Delete();
mapper->Delete();
cone->Delete();
```

Il est également bon de s'habituer dès le début à ne pas utiliser les opérateurs new et delete du standard avec les objets VTK. Les pages 17 à 24 du livre *The VTK User's Guide* résument bien le fonctionnement du pipeline VTK et les principaux composants du système.

3.4 Contrôle de l'affichage

Le contrôle de l'affichage de VTK est fait de manière implicite. En effet, chaque instance de vtkRenderWindow possède par défaut un vtkRenderWindowInteractor. Cet objet permet de modifier la position de la caméra et de la plupart des acteurs contenus dans la scène. Divers styles d'interaction sont disponibles. Ces derniers répondent aux évènements de la souris et du clavier de manière différente.

Les pages 41 à 44 du livre *The VTK User's Guide* décrivent de façon détaillée la manière d'interagir avec la scène et la procédure à suivre pour changer le style d'interaction.

3.5 Affichage d'image

L'affichage d'une image peut être fait à l'aide d'une texture appliquée sur un plan. Pour réaliser cette tâche, il faut simplement charger l'image qui doit être affichée avec un objet vtkImageReader2 qui sera créé par un objet vtkImageReader2Factory. La factory s'occupe de créer le bon objet dérivé de la classe vtkImageReader2. La suite des opérations est identique à la procédure pour charger une image de type Bitmap qui est détaillée à la page 54 du livre *The VTK User's Guide*.

3.6 Picking

Comme le contrôle de l'affichage, VTK permet de faire du picking sur les acteurs de la scène de façon implicite. Chaque instance de vtkRenderWindowInteractor possède par défaut un objet dérivé de vtkAbstractPicker. VTK supporte différentes manières de faire de la sélection sur les acteurs. Ces méthodes sont encapsulées dans des classes dont l'opération de sélection peut être déterminée de façon matérielle ou logicielle. Il est claire que les méthodes matérielles sont plus performantes, elles fournissent toutefois moins d'informations que celles logicielles. Les pages 54 à 57 du livre *The VTK User's Guide* expliquent ce sujet.

VTK fournit également une gamme de 3D widgets. Ces widgets peuvent être utilisés pour modifier la dimension ou l'emplacement d'un objet dans la scène. Le livre *The VTK User's Guide* décrit ces derniers aux pages 67 à 71.

3.7 Variation de la couleur en fonction d'un scalaire

VTK permet de faire varier la couleur d'une surface en fonction d'un scalaire quelconque. C'est la classe vtkLookupTable qui permet de faire ce genre d'opération. Les pages 76 et 77, 86 et 87 du livre *The VTK User's Guide* sont intéressantes pour en savoir davantage.

4 Prototype liant VB et VTK

4.1 Énumération des contraintes

La principale contrainte qui devait être respectée lors du développement du prototype unissant l'interface usager réalisée en Visual Basic .NET et VTK est directement liée au bus logiciel. En effet, les évènements de la souris et du clavier peuvent être utiles à plusieurs modules desservis par le bus. Il est donc indispensable de rediriger ces évènements vers le bus pour que tous les intéressés soient mis au courant.

La seconde contrainte est de ne pas spécialiser les classes de VTK. Cette technique avait été utilisée avec les classes de C++ Views et ce fut une erreur. Cette approche n'est donc pas une solution à envisager avec VTK.

4.2 Approche retenue

L'approche retenue pour faire le lien entre l'interface usager et VTK est constituée de fenêtres VB et d'une DLL. Les fenêtres VB créées connaissent uniquement une fonction

associée à une DLL. Cette fonction correspond à la fonction d'envoie d'évènements du bus. Elle constitue également un wrapper sur des objets connus à l'intérieur de la DLL seulement. Les fenêtres n'ont donc aucune idée de la représentation interne utilisée par la DLL.

Pour ce qui est de la DLL, elle n'a aucune idée du genre de fenêtre qui l'utilise. La seule chose qu'elle connaît est le Handle de la fenêtre que VTK doit utiliser pour faire son affichage. Il est également indispensable que chaque fenêtre ait un identifiant unique qui permettra de retrouver l'objet qui doit être utilisé lors d'un envoie d'évènement.

Les points intéressants de cette approche sont que ni l'interface et ni le système graphique n'a conscience de ce qui se passe chez l'autre. Le lien entre les deux est donc très faible. De plus, elle permet d'intégrer très facilement le bus logiciel.

5 Design du module de visualisation

5.1 Présentation

Le design illustré par le diagramme de classe à l'annexe B a été conçu pour faciliter l'ajout de nouvelle couche d'affichage. RGCouche implémente donc la classe abstraite qui sert de base aux différentes couches.

Ce design permet également la substitution d'une implémentation d'un algorithme de représentation par une autre. Il est donc relativement simple de changer l'outil graphique actuel (VTK) pour un nouvel outil graphique sans tout détruire le code existant.

5.2 Description des rôles

Voici la liste des classes contenues dans le diagramme de l'annexe B ainsi que la description de leurs rôles respectifs :

GUI	: GUI est le module de l'interface usager.
Fenetre	: Fenetre correspond à une fenêtre VB. L'attribut fenetreId est nécessaire lors de l'envoi d'évènements sur le bus.
Groupe	: Groupe sert à appliquer les transformations (rotation, translation, zoom avant et arrière) faites à une fenêtre à toutes les autres fenêtres du groupe.
VTK	: VTK est le module du système graphique.
BD	: BD est le module de la base de donnée.

BDGstDonnees : BDGstDonnees crée et sauvegarde les données contenues dans la base de donnée. C'est donc le travail du gestionnaire de charger et de sauvegarder les couches d'affichage, les espaces de représentation, les maillages, etc.

RGGstEspaceRepresentation : RGGstEspaceRepresentation permet d'ajouter, d'enlever et de retrouver des espaces de représentation. Chaque espace est identifié par un numéro unique. Des méthodes sont fournies pour itérer sur les espaces de représentation contenus dans le gestionnaire.

RGEspaceRepresentation : RGEspaceRepresentation est une classe abstraite servant d'interface à l'espace d'affichage d'une fenêtre. C'est l'espace qui contient les différents plans et c'est lui qui gère plusieurs évènements de souris et de clavier.

RGCouche : RGCouche est la classe abstraite de toutes les couches d'affichage.

RGCouchePlan : RGPlan correspond à un plan dans lequel il est possible d'ajouter et d'enlever des couches d'affichage. Des méthodes sont fournies pour itérer sur les couches contenues dans le plan. Cette classe implémente le Design pattern du Composite décrit aux pages 163 à 173 du livre *Design patterns*.

RGAlgoRepresentation : RGAlgoRepresentation est la classe abstraite de tous les algorithmes.

RGRepresentationImp : RGRepresentationImp est la classe abstraite de toutes les implémentations possible d'imaginer. Les classes dérivées ont donc le mandat de fournir l'implémentation des primitives graphiques. Cette classe implémente le Design pattern du Bridge élaboré aux pages 151 à 161 du livre *Design patterns*.

5.3 Lacunes et fonctionnalités manquantes

Les paragraphes suivants listent des lacunes et des fonctionnalités manquantes en donnant une solution pour les résoudre lorsque possible.

RGCouche contient un attribut affichable. En ce moment, il n'y a pas de moyen de changer la visibilité d'un acteur présent dans la scène VTK. La manière la plus évidente

de procéder est d'utiliser la classe RGRepresentationVTKImp pour modifier l'état de l'acteur qui sera préalablement stocké dans l'attribut donnee de la couche.

RGEspaceRepresentation contient une méthode handleMessage. Cette méthode est toutefois trop ressemblante aux fonctions de réception des évènements de Windows et ainsi trop associée à ce système d'exploitation. Il serai donc souhaitable de trouver une manière pour éviter de lier les espaces de représentation à un seul système d'exploitation.

6 Conclusion

6.1 Résumé

Au cours de ce rapport, il a été question de la recherche d'un outil graphique pour remplacer l'ancien inadéquat compte tenu des nouveaux besoins. Nous avons également pu connaître plus en profondeur Visualization Toolkit et ses particularités. Le prototype liant VB et VTK a de plus été introduit et décrit afin de mieux le cerner. Finalement, le design du module de visualisation nous a été décrit et nous a permis de connaître ses lacunes et ses fonctionnalités manquantes.

6.2 Perspectives futures

Les prochaines étapes consistent à faire ou à trouver les correctifs pour les lacunes et les fonctionnalités manquantes énumérées en 5.3.

7 Glossaire

Espace de représentation : l'espace de représentation est l'endroit où se fait l'affichage. Normalement un espace de représentation est lié à une fenêtre.

Plan : les plans servent à contenir des couches. Il y a différents types de plans (plan arrière, plan médian, plan avant, plan interractif).

Couche : les couches servent à représenter de l'information.

Picking : l'action de sélectionner un point dans une image à l'aide de la souris, et de pouvoir interagir avec l'image.

8 Bibliographie

Kitware, Inc. *The VTK User's Guide*. New York: Kitware, Inc., 2003.

Gamma, Erich; Helm, Richard; Johnson, Ralph; Vlissides, John. *Design Patterns, Elements of reusable object-oriented software*. Reading: Addison-Wesley Publishing Company, 1995.

9 Annexes

9.1 Annexe A - Liste des outils graphiques

Visualization ToolKit (VTK)

VTK est basé sur le paradigme par flot de données. Dans ce paradigme, les données circulent dans un réseau de modules. Ces modules sont en fait des algorithmes qui modifient les données qui entrent dans ceux-ci. La gestion de la mémoire est faite par compteur de référence. Cette approche a été retenue pour minimiser l'utilisation de la mémoire.

Producteur : GE Corporate Research & Development
Version : 4.2.1
Date : 18 février 2003

Avantages :

- Indépendant de la plate-forme.
- Indépendant du système de fenêtrage.
- Open source et gratuit.
- Extensible.
- Permet la sauvegarde du résultat sous les formats BMP, JPEG, PNG, PPM, TIFF et PostScript.
- Permet d'afficher du texte.
- Capable de mixer du 2D avec du 3D.
- Permet de faire du Texture mapping.
- Capable de travailler avec une portion des données.
- Gestion implicite de la caméra et des sources lumineuses.
- Permet de faire des applications à fenêtre multiple et/ou à plusieurs Viewports.
- Permet de faire facilement du picking.
- Nouvelle version compilée toutes les nuits.
- L'affichage n'est pas forcément fait par OpenGL. (ex : OpenGL pour les PC, XGL pour les Sun)

Inconvénients :

- Pas très rapide.
- Très lourd.
- Non thread-safe (OpenGL non plus)

Note :

- Lorsqu'il y a beaucoup de données, il est préférable de désactiver les Display List d'OpenGL. Voir les FAQ sur le site Kitware.

Support :
- Support professionnel offert par Kitware, Inc.
- Mailing-list.

Livres :
- The Visualization Toolkit User's Guide, ISBN 1-930934-08-4, publié par Kitware.
- The Visualization Toolkit, An Object-Oriented Approach To 3D Graphics, 3rd edition, ISBN 1-930934-07-6, publié par Kitware.

Tutoriaux :
- http://cherokee.ncsa.uiuc.edu/~semeraro/PPT/VTK_TUTORIAL/frame.htm
- http://www.fysik.dtu.dk/CAMP/vtktutorial.html

Référence :
- http://public.kitware.com/VTK/

Open Visualization Data Explorer (OpenDX)

OpenDX est la version Open Source et gratuite du produit d'IBM Visualization Data Explorer. C'est un framework qui permet aux usagers de voir leurs données de la manière qu'il le souhaite.

Producteur : IBM
Version : 4.2.0
Date : 20 mai 2002

Avantages :
- Portable.
- Open source et gratuit.
- Extensible.
- Permet la sauvegarde du résultat sous les formats GIF, RGB, TIFF, YUV et PostScript.
- GUI intégré basé sur X Windows et Motif permettant de faire de la programmation visuelle.
- Permet de faire des manipulations, des transformations et des animations.
- Supporte les ordinateur uni-processeur et multiprocesseur.
- Peut s'exécuter de manière distribuée. (Grappe d'ordinateurs ou Cluster)
- Permet la création de module avec Module Builder.
- L'affichage est fait par OpenGL de façon matérielle. Si OpenGL n'est pas pris en charge, OpenDX possède son propre Software-Renderer.
- Prétend être de haute performance.
- Gestion implicite d'une cache.

- Permet de faire facilement du picking.
- Permet l'affichage d'isosurface.

Inconvénients :
- La documentation en ligne et fournie avec le code source n'est pas très étoffée.
- Possibilité de problèmes et bugs.
- IBM n'offre pas de support officiel.
- Ne semble pas être en mesure d'être compilé sur Windows (CygWin) sans modification.

Notes :
Visualization and Imagery Solutions, Inc (VIS) vend des versions d'OpenDX pré-compilées pour Windows.
25 $ US pour la version téléchargeable.
35 $ US pour la version sur cd-rom.

Support :
- Support professionnel offert par VIS.
- Mailing-list.

Livre :
- OpenDX Paths to Visualization, publié par VIS, Inc.

Références :
- http://www.research.ibm.com/dx/
- http://www.opendx.org/
- http://www.vizsolutions.com/

Open Inventor

Open Inventor utilise un graphe acyclique directionnel, ce qui signifie que le résultat est directement dépendant de l'ordre dans lequel le système parcours les nœuds du graphe.

Producteur : Silicon Graphics
Version : 2.1.1

Avantages :
- Indépendant de la plate-forme.
- Indépendant du système de fenêtrage.
- Open source et gratuit.
- Permet la sauvegarde du résultat sous le format PostScript.
- L'affichage est fait par OpenGL.
- Interaction par évènements simples.
- Permet de faire une sélection d'objet de haute performance.
- Puissant et efficace.

Inconvénients :
- Reste de bas niveau en comparaison avec VTK et OpenDX.
- L'ordre du parcours des nœuds du graphe influence le résultat final.

Livres :
- The Inventor Mentor, ISBN 0-201-62495-8, écrit par Josie Wernecke.
- The Inventor Toolmaker, ISBN 0-201-62493-1, écrit par Josie Wernecke.
- The Open Inventor C++ Reference Manual, ISBN 0-201-62491-5.

Références :
- http://www.sgi.com/software/inventor/
- http://www.tgs.com/

IRIS Explorer

IRIS Explorer 5.0 est un logiciel à part entière. Il utilise Open Inventor pour l'affichage et permet la création de modules en C et en Fortran à l'aide de l'outil Module Builder.

Producteur : Numerical Algorithms Group
Version : 5.0

Tutorial :
- http://www.scs.leeds.ac.uk/iecoe/tutorial/main-frm.html

Référence :
- http://www.nag.co.uk/Welcome_IEC.html

Advanced Visual Systems (AVS)

AVS n'est pas un produit mais un producteur qui offre plusieurs solutions aux industries.

Voici une liste de quelques-uns de leurs produits :
- OpenViz
- VizWorks
- AVS/Express
- Gsharp
- AVS/PowerViz

Référence :
- http://www.avs.com/

Lymb

Lymb est un pluggin pour Netscape. L'affichage est réalisé par OpenGL, Iris GL ou XGL.

Producteur : GE Research and Development Center

Référence :
- http://www.crd.ge.com/~lorensen/hyperlymb/hyperlymb.html

Visage

Visage est un logiciel pour explorer, analyser et visualiser des informations.

Producteur : MAYA Design Group

Référence :
- http://www.maya.com/Visage/

VISAGE (VISualization, Animation, and Graphics Environment)

VISAGE est un système qui permet de voir rapidement un grand volume de données.

Producteur : GE Research and Development Center

Référence :
- http://www.crd.ge.com/esl/cgsp/fact_sheet/rapiddat/index.html

Simple DirectMedia Layer (SDL)

SDL est une API multimédia multi-platforme et gratuite. Il a été développé pour donner un accès simple au clavier, à la souris, aux manettes de jeux, au cd-rom, au système audio et à l'affichage matériel de scène 3D via OpenGL.

Version : 1.2.5
Date : 5 octobre 2002

Référence :
- http://www.libsdl.org/index.php

Interactive Data Language (IDL)

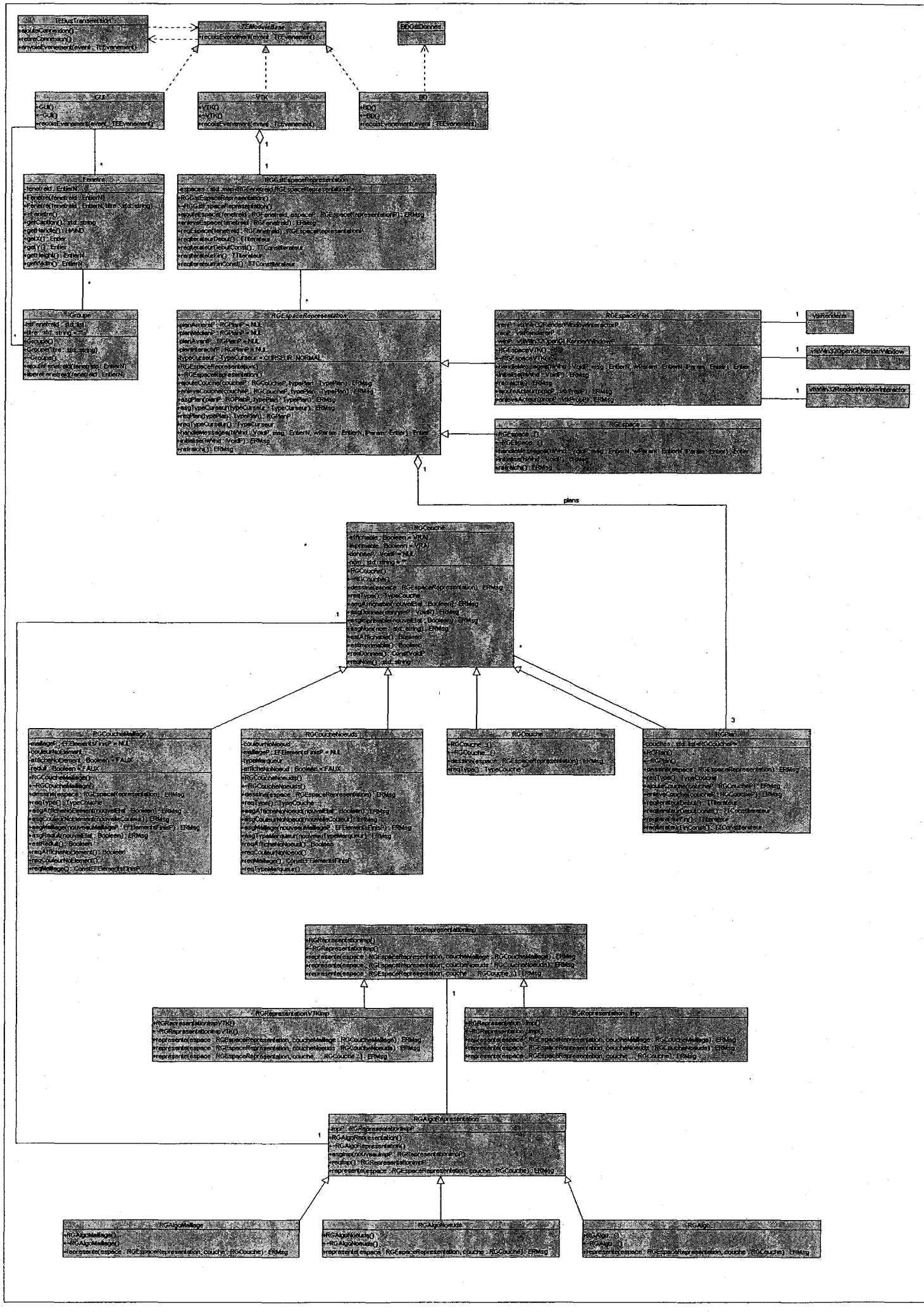
IDL est un logiciel permettant l'analyse et la visualisation de données.

Producteur : Research Systems, Inc.
Version : 5.6
Date : 28 octobre 2002
Coût : 2 350 $ US – 3 350 $ US

Référence :

- http://www.rsinc.com/idl/

9.2 Annexe B – Diagramme des classes


TEBusTransmetteur
TEModuleBase
BDGdiDonnee
GUI
VTK
BD
Fenetre
RGGdiEspaceRepresentation
RGCouleur
RGEspaceRepresentation
RGEspaceVtk
RGEspace
plans
RGCouche
RGCoucheMaillage
RGCoucheNoeuds
RGCouche
RGPlan
RGRepresentationImp
RGRepresentationImpVTK
RGRepresentationImp
RGAlgoRepresentation
RGAlgoMaillage
RGAlgoNoeuds
RGAlgo

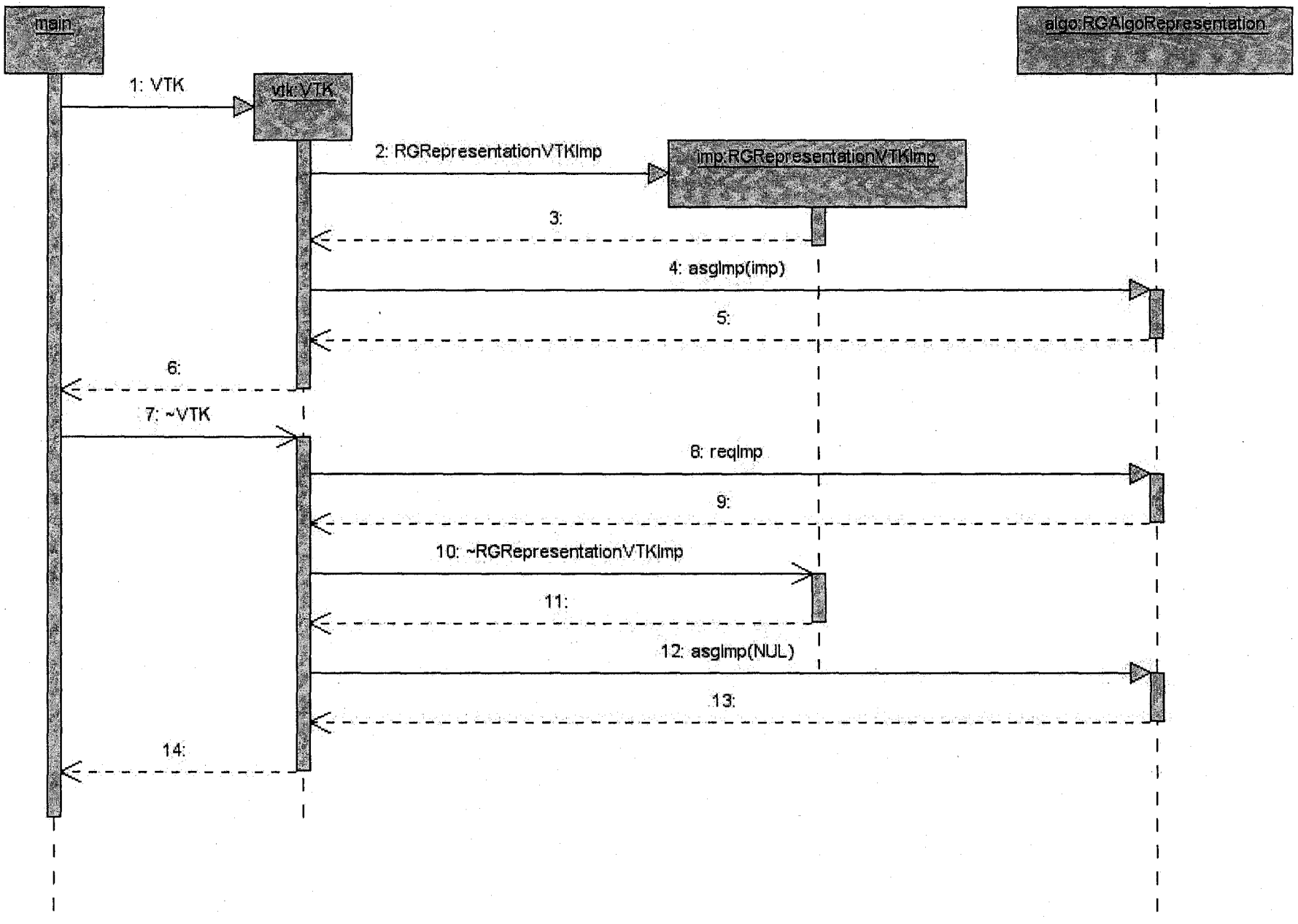
9.3 Annexe C - Diagrammes de séquence

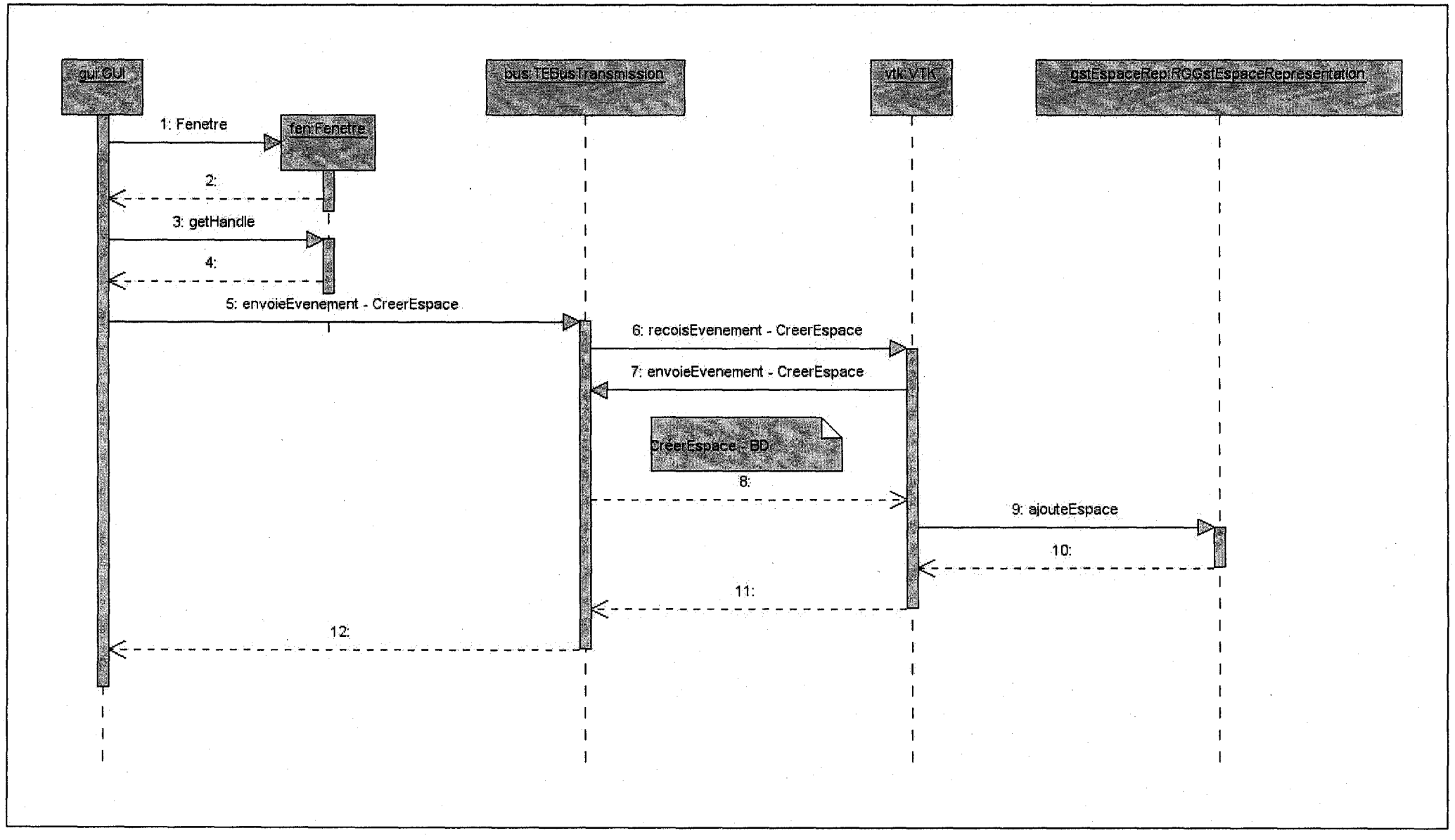
Diagramme 1 : Assignation implémentation VTK aux algorithmes

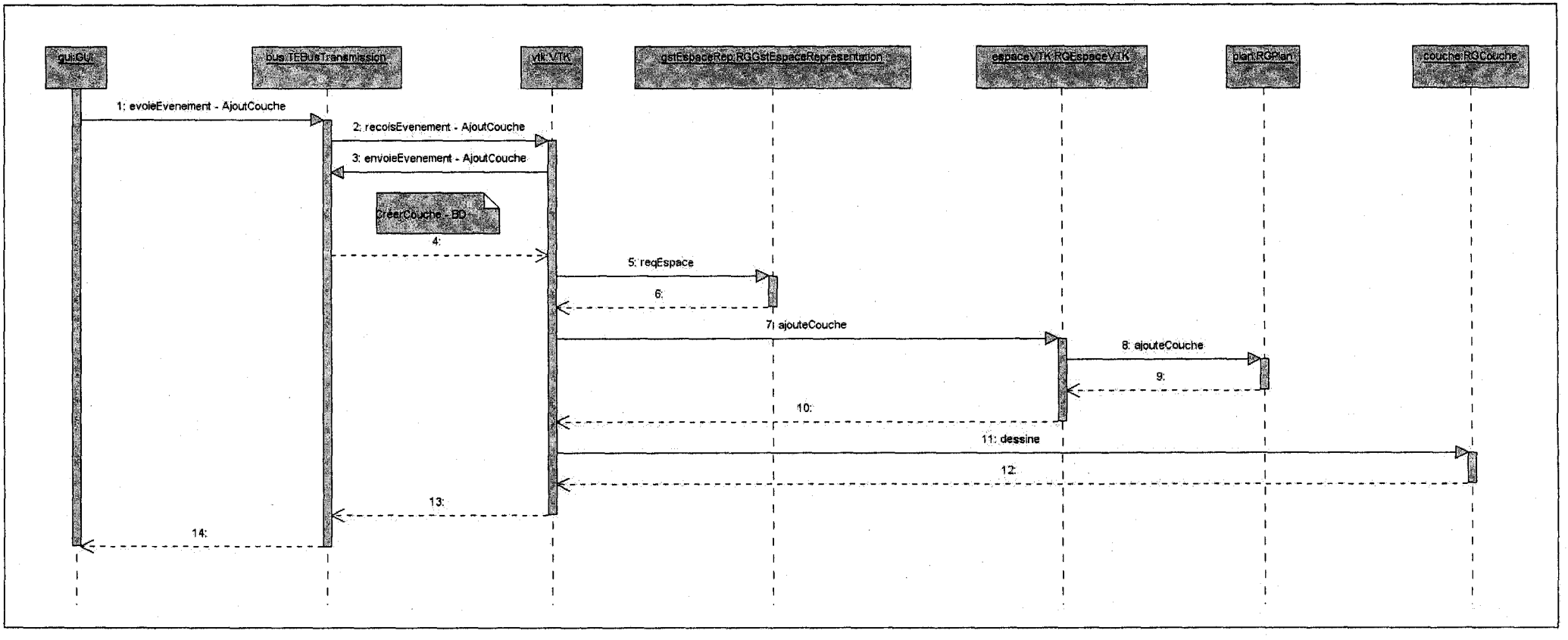
Diagramme 2 : Création d'un espace de représentation

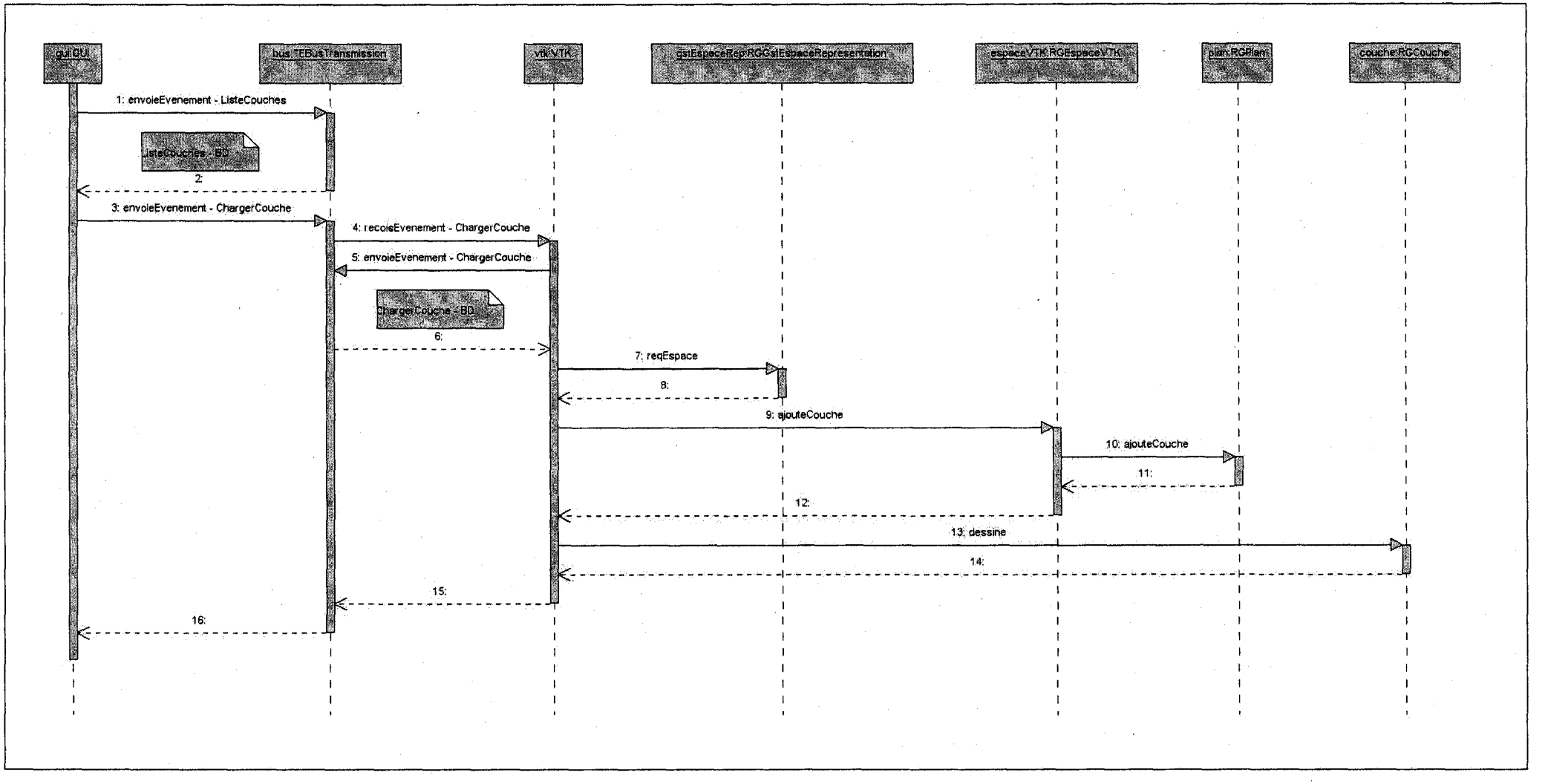
Diagramme 3 : Création d'une couche

Diagramme 4 : Chargement d'une couche


main
vtk:VTK
imp:RGRepresentationVTKImp
algo:RGAlgoRepresentation
1: VTK
2: RGRepresentationVTKImp
3:
4: asgImp(imp)
5:
6:
7: ~VTK
8: reqImp
9:
10: ~RGRepresentationVTKImp
11:
12: asgImp(NUL)
13:
14:


gui:GUI
bus:TEBusTransmission
vtk:VTK
gstEspaceRep:RGGstEspaceRepresentation
1: Fenetre
fen:Fenetre
2:
3: getHandle
4:
5: envoieEvenement - CreerEspace
6: recoisEvenement - CreerEspace
7: envoieEvenement - CreerEspace
CréerEspace - BD
8:
9: ajouteEspace
10:
11:
12:


gui:GUI
bus:TEBusTransmission
vtk:VTK
gstEspaceRep:RGGstEspaceRepresentation
espaceVTK:RGEspaceVTK
plan:RGPlan
couche:RGCouche
1: evoieEvenement - AjoutCouche
2: recoisEvenement - AjoutCouche
3: envoieEvenement - AjoutCouche
CréerCouche - BD -
4:
5: reqEspace
6:
7: ajouteCouche
8: ajouteCouche
9:
10:
11: dessine
12:
13:
14:


gui:GUI
bus:TEBusTransmission
vtk:VTK
gstEspaceRep:RGGstEspaceRepresentation
espaceVTK:RGEspaceVTK
plan:RGPlan
couche:RGCouche
1: envoieEvenement - ListeCouches
listeCouches - BD
2:
3: envoieEvenement - ChargerCouche
4: recoisEvenement - ChargerCouche
5: envoieEvenement - ChargerCouche
ChargerCouche - BD
6:
7: reqEspace
8:
9: ajouteCouche
10: ajouteCouche
11:
12:
13: dessine
14:
15:
16:

Rapport de développement logiciel

Mise en place de la base de données relationnelle

Présenté par :

Samuel Ouellet
Eric Larouche

2003-12-19

1. Introduction

1.1 Objectifs

Ce rapport discute de la première phase de conception de la base de données relationnelle pour Modeleur 2.0. Il constitue un résumé de tout le travail d'analyse et de développement réalisé ainsi que les voies à emprunter pour la suite du développement. Ce rapport permettra donc de comprendre le cheminement réalisé jusqu'à maintenant en ce qui a trait au développement de la base de données relationnelle, et d'identifier quelles seront les prochaines étapes.

1.2 Contexte

Dans la première version de Modeleur les informations étaient contenues dans une base de données objet interne au logiciel. Il était donc très difficile d'accéder à ces données par des sources externes. C'est pour cette raison que la décision d'utiliser une base de données relationnelle a été envisagée.

Les bases de données relationnelles ont l'avantage de prendre en charge beaucoup de fonctionnalités utiles au fonctionnement de Modeleur, tout en permettant l'accès aux données par une source externe.

Les bases de données relationnelles utilisent une interface commune basée sur le langage SQL pour accéder aux données. Les bases de données relationnelles permettent de faire des requêtes sur les données et d'ainsi ne charger qu'une partie des données. Elles ont également l'avantage de prendre en charge la gestion des usagers et de permettre le travail partagé (multi usagers) sur une même base de données.

1.3 Structure

Ce rapport présente plusieurs aspects qui ont été analysés lors de cette phase développement. Pour chacun des aspects, une brève présentation est faite, suivie d'un aperçu du travail effectué et d'un résumé des discussions. Finalement, les renseignements techniques et les pistes de solutions sont expliqués en détails.

Les aspects discutés sont les suivants :

- *Analyse des outils de design de base de données*
- *Analyse des systèmes de base de données*
- *Design de la nouvelle base de données de Modeleur*
- *Analyse des coordonnées GIS*

- *Installation de la base de données PostgreSQL/PostGIS*
- *Développement du traducteur de projet*
- *Développement du chargement d'un maillage de la nouvelle base de données*
- *Analyse des problèmes de compatibilité entre les bases de données*

2 Recherche d'outils de travail

2.1 Outils de design de base de données

Avant de débuter le design de la nouvelle base de données, il fallait trouver le meilleur outil pour développer sa structure. L'utilisation d'un tel logiciel permet de faciliter la création et la modification d'une base de données.

2.1.1 Résumé des discussions

Quelques outils de design ont été retenus afin d'être évalués selon plusieurs critères de sélection (voir ANNEXE 1 – Outils de design de bases de données). Il y a d'abord eu une première réunion où seul les outils les plus envisageables ont été retenus.

Par la suite, ces logiciels ont fait l'objet d'une analyse plus poussée en testant les fonctionnalités de chacun. Il est ressorti que l'outil le plus adéquat pour les besoins du projet était « Case Studio 2 Full ».

Il a l'avantage d'être complet et de permettre de garder une trace de l'évolution du design avec un système de sauvegarde par version. Il a aussi l'avantage de supporter tous les systèmes de base de données et d'optimiser le design selon le système choisi. Pour plus de détails, voir en référence : Base de données - Rapport 1 et 2.

2.1.2 Utilisation de l'outil Case Studio 2

Le logiciel Case studio 2 est assez simple d'utilisation. Pour ouvrir le design actuel de la base de données, il faut appuyer sur « Ouvrir », sélectionner le type de fichier : « Projects (.dp2) » et ouvrir le projet de design.

Actuellement le gestionnaire de projets se nomme « Version DB ». Une fois le gestionnaire ouvert, il est possible d'extraire une version en la sélectionnant puis en appuyant sur « Out ». Il est possible de l'enregistrer en appuyant sur « In » dans le gestionnaire de version ou en fermant une fenêtre qui a été modifiée. Il est fortement suggéré de faire une nouvelle version lorsqu'il y a des changements importants.

Pour générer le script de la base de données, il faut appuyer sur l'éclair ou {F9}. Pour convertir la modèle dans un autre système de base de données, il faut aller dans le menu « model → database conversion » ce qui créé un nouveau modèle. La conversion de type est ajustable dans le bouton « Setup ». Les usagers sont modifiables dans le menu « model ».

2.2 Systèmes de base de données

Pour être en mesure d'utiliser une base de données relationnelle publique il fallait avant toute chose explorer les différents systèmes disponibles sur le marché pour connaître leurs différences. Il n'est pas nécessairement avantageux de forcer l'utilisateur de Modeleur à installer une base de données spécifique. Il est donc souhaitable d'être compatible avec tous les systèmes de base de données.

2.2.1 Résumé des discussions

Une étude détaillée a été faite afin de déterminer quelle base de données il serait préférable d'utiliser. Tous les systèmes de base de données ont été explorés et comparés (Voir ANNEXE 2 – Systèmes de bases de données). Le choix pour la base de données principale a été PostgreSQL parce qu'elle est standard, complète, gratuite et qu'elle permet d'utiliser les coordonnées de type GIS.

Les fonctionnalités d'OpenGIS (Geographical Information System) sont un ajout au standard SQL (Structured Query Language) qui permet d'insérer et de traiter des objets géographiques dans une base de données. Il a été décidé que l'utilisation des GIS était un critère important pour le choix de la base de données. Toutefois, il y a encore relativement peu de bases de données qui supporte les fonctionnalités GIS.

Pour les bases de données qui ne supportent pas les données GIS, il faudrait intercepter les commandes spatiales et les traduire. Ceci implique de faire le traitement habituellement fait par les fonctions GIS de la base de données directement dans le logiciel. Pour rester cohérent avec la partie GIS, il faudrait alors mettre les données géographiques dans une colonne de type « varchar » du même nom que les colonnes GIS. Enfin, il faudrait que le format de GIS soit utilisé en respectant le fonctionnement des projections.

2.2.2 Choix des bases de données supportées

Il a été décidé que la base de données PostgreSQL serait la base de données utilisée pour le développement.

La base de données Oracle sera utilisée comme seconde base de données offrant toutes les capacités spatiales (GIS). Oracle est une base de données utilisée par certains utilisateurs de Modeleur 1.0, et son support dans Modeleur 2.0 est fortement souhaité.

La base de données Access sera également supportée car elle constitue la base de données la plus répandue (en raison de la suite Microsoft Office). Par contre, elle ne supporte pas les objets GIS et n'est pas standard.

Il s'agit donc des trois systèmes de base de données qui seront implantés en premier. Les autres systèmes pourront être validés par la suite car leur utilisation devrait être similaire.

3 Design

3.1 Design de la nouvelle base de données de Modeleur

Après avoir complété les recherches préliminaires, il a fallu mettre sur pied la base de données relationnelle pour enregistrer les données des projets Modeleur. Cette conception s'est faite en modélisant les données actuellement enregistrées et en les structurant sous forme de tables. Il en est résulté le modèle représenté dans le projet CaseStudio appelé « Version DB ». Celui-ci a évolué suite à plusieurs réunions et discussions.

3.1.1 Résumé des discussions et développements

Le design réalisé représente les principaux types de données qui seront contenus dans le logiciel Modeleur 2.0. Les partitions, les maillages, les champs, les espaces de représentation et les séries y sont représentés.

Il reste à intégrer les habillages d'écran et les composants graphiques (iso-lignes, iso-surfaces, etc.). Ceci devrait être assez simple une fois décidé comment ils seront enregistrés dans la base de données. La mise en place des iso-lignes pourrait se faire en utilisant les MULTILINESTRING et celle des iso-surfaces en utilisant les MULTIPOLYGON, des fonctionnalités GIS.

Les systèmes de référence (projections) ne seront plus une entité indépendante, comme c'était le cas dans l'ancienne base de données. Ils seront plutôt associés à chaque objet par l'intermédiaire de leur table information.

Actuellement il n'y a pas beaucoup de méta données (« meta data ») qui sont conservées sur les données. Il a été suggéré d'utiliser une gamme d'informations comparable à celle du logiciel m3cat. Pour utiliser cette approche, des tables devraient être ajoutées au design.

3.1.2 Description du design

Le design de la base de données a été validé et apparaît satisfaisant dans son ensemble. Il y a quelques points particuliers qu'il est important de spécifier.

3.1.2.1 Séries

Le design de la partie « serie » dans la base de données est fortement inspiré du maillage. Les séries sont représentées sous la forme d'éléments afin de faciliter l'interpolation. Comme les algorithmes pour travailler avec les éléments existent déjà, il est avantageux de s'en servir pour effectuer le traitement des séries.

3.1.2.2 Échelle

La table « echelle » a pour but de mettre un « tag » sur les nœuds, en leur associant un niveau de précision et un niveau d'erreur. Ces valeurs devront être indiquées à l'insertion des données. La méthode utilisée pour déterminer le niveau de précision/d'erreur reste à déterminer, de même que l'information qui sera contenue dans ces attributs.

Au niveau de l'affichage, il sera possible d'effectuer le traitement plus rapidement. Par exemple, un maillage pourrait être construit uniquement avec les nœuds qui correspondent à un certain niveau de précision.

Au niveau des calculs, il sera possible de ne considérer que les nœuds correspondant à un certain niveau de précision, ce qui accélérera le calcul, en augmentant en contrepartie le niveau d'erreur.

3.1.2.3 Boucles fermées

Le design actuel comporte un problème en ce qui a trait à la présence de boucles fermées crées par les tables « nœud_arete » et « arete_face ». Ces tables constituent des risques quant à l'intégrité de la base de données car elles permettent d'introduire des incohérences.

En effet, à cause de ces tables, l'arête d'un élément pourrait faire référence à un nœud qui lui n'est pas relié à cet élément. Pour empêcher ce problème, il a été décidé d'utiliser des « triggers » qui vérifieront avant d'introduire la valeur dans l'une de ces tables que l'élément référé est le même du côté de l'arête et du côté du nœud. Ces « triggers » ne sont pas encore implantés dans la base de données.

3.2 Analyse des coordonnées GIS

L'utilisation des coordonnées GIS a apporté plusieurs problèmes de conception qu'il a fallu analyser tout au long du développement. Le principal problème est que l'on ne peut pas mettre des données de différentes projections dans une colonne GIS. En fait, il faut, selon le standard, créer la colonne en lui spécifiant le type de projection (srid) qu'elle contiendra. Par la suite, les données d'une autre projection ne peuvent pas y être insérées.

3.2.1 Résumé des discussions

Pour résoudre ce problème cinq possibilités ont été envisagées :

- ajouter une nouvelle colonne par projection pour toutes les tables hébergeant des données GIS.

- faire une table pour chaque projection et y enregistrer tous les éléments qui sont dans cette projection ensemble dans cette même table GIS.

- construire une table par projection pour chaque objet hébergeant des données GIS.
- ne pas respecter le standard et mettre toutes les données dans la même colonne.
- convertir toutes les données enregistrées dans une même projection.

Aucune de ces possibilités ne constitue une solution idéale. Cependant, la première solution a été retenue parce qu'elle est la plus propre. Pour plus de détails, voir ANNEXE 3 – Enregistrer des données GIS.

Afin d'éviter que le nombre de colonnes n'augmente trop rapidement, il a été proposé d'utiliser les schémas de la base de données pour séparer les projets. Les schémas sont, pour une base de données, l'équivalent des répertoires pour un système d'exploitation. Ils permettent de subdiviser une base de données en plusieurs parties indépendantes.

Comme dans un projet il ne devrait pas y avoir plus de dix projections, le nombre de colonnes ne deviendra pas trop élevé. Par contre, comme certaines bases de données ne supportent pas les schémas (comme Access), il faudra parfois regrouper tous les projets ensemble et se retrouver avec plus de colonnes.

3.2.2 Renseignement technique

Suite à l'adoption de cette solution, de nouveaux problèmes sont apparus. En effet, comme il faut maintenant créer des colonnes GIS en temps réel, il faut que le logiciel sache comment créer ces colonnes.

Le problème est que pour créer une colonne GIS dans une table, il faut appeler une fonction spéciale et lui passer le nom de la base de données. Ceci est contraire au principe d'ODBC puisque l'usager n'est pas censé devoir connaître le véritable nom de la base de données.

Le driver ODBC est en réalité une porte avec un nom constant qui pourrait pointer sur n'importe quelle base de données. Ceci permet de pouvoir modifier le nom de la base de données, le système de base de données ou l'emplacement de la base de données sans avoir à modifier les programmes qui l'utilisent.

Pour résoudre le problème, il doit être possible de trouver le nom de la base de données. Ceci peut se faire en extrayant l'information dans les registres de Windows. On y retrouve en effet toutes les informations d'un DNS ODBC dont entre autre le type de système de base de données et le nom de la base de données.

4 Installation

4.1 Installation de la base de données PostgreSQL/PostGIS

L'installation de la base de données principale de test a été très difficile à effectuer. Il a fallu beaucoup d'essais pour réussir à bien faire l'installation et à mettre en place la procédure d'installation de toutes ses composantes.

4.1.1 Résumé des discussions

Il serait très utile pour plusieurs raisons d'être en mesure de fournir un script d'installation de la base de données avec le programme Modeleur. Ceci permettrait aux usagers d'avoir en mains une base de données utilisable par Modeleur 2.0 sans avoir à configurer manuellement tous les logiciels périphériques.

Voici deux solutions qui sont envisageables pour compléter cette tâche:

4.1.1.1 Solution 1 – tout reconstruire

On pourrait tout d'abord créer un script qui fait tous les « make install » et les modifications automatiquement. Ce script serait très difficile à réaliser car il y a plusieurs programmes différents à installer et plusieurs configurations à faire.

Cette solution pourrait avoir comme avantage que la toute dernière version des logiciels pourrait être extraite d'Internet. Par contre, s'il s'avère que la version à jour d'un logiciel n'est plus compatible avec l'un ou l'autre des composants, ceci devient un désavantage.

4.1.1.2 Solution 2 – copier tout

La deuxième possibilité consiste à copier directement tous les fichiers aux bons endroits comme le font les « make install ». Pour cela, il suffit de retracer tous les fichiers installés et de faire un script pour les copier aux bons endroits. Bien sûr, il faudra dans ce cas s'assurer de faire la configuration du système. Cette deuxième solution est plus facilement envisageable et résistera mieux aux changements.

4.1.2 Renseignement technique de l'installation

L'installation de PostGIS a été très difficile et comporte encore quelques problèmes à chaque installation sur une nouvelle machine. Les versions utilisées ainsi que la marche à suivre sont décrites dans le fichier « Installation PostGIS.txt », reproduit à l'ANNEXE 4 – Installation de PostgreSQL/PostGIS.

Pour simplifier, il est suggéré de repartir du code source déjà compilé et de seulement faire les « make install » de chaque programme. L'installation doit être faite sous un environnement Unix comme cygwin pour Windows.

La base de données PostgreSQL pourra être bientôt directement compilée sous Windows mais l'extension de PostGIS n'est toujours pas prévue. Il ne sera donc pas possible de faire le transfert complet vers Windows, du moins à court terme.

5 Développement du prototype

5.1 Développement du traducteur de projet

Le traducteur de projet permet d'aller lire des données dans l'ancienne base de données pour ensuite les introduire dans la nouvelle structure relationnelle.

Ceci permet de commencer l'implantation de la sauvegarde des données dans la base de données relationnelle. Du même coup, le traducteur constitue le début d'un programme qui sera utilisé pour transformer les projets de Modeleur 1.0 en projet de Modeleur 2.0.

5.1.1 Résumé des discussions

5.1.1.1 Utilisation de OTL

Le code qui a été développé utilise une librairie qui interface avec ODBC : la librairie OTL 4.0. Ceci permet de manipuler une interface utilisant les « streams » (semblables à ceux du C++) pour interagir avec la base de données. Cela permet d'éviter d'avoir à utiliser directement les fonctionnalités ODBC.

OTL a été développé par Sergei Kuchin (http://otl.sourceforge.net/home.htm). Il a en quelque sorte combiné les « streams » C++ et le langage SQL pour former un langage avec des « data streams ».

L'utilisation de OTL facilite grandement l'interaction avec la base de données. Il ne s'agit que d'inclure le fichier otlv4.h et d'utiliser les opérateurs '>>' et '<<' pour lire et écrire dans la base de données via des requêtes SQL.

5.1.1.2 Maillage

Actuellement, il y a uniquement la partie de la sauvegarde du maillage qui a été mise en place. Cette partie est des plus importantes car elle représente la base de plusieurs autres informations d'un projet.

Le maillage a une structure assez complexe parce qu'il regroupe beaucoup de sous-ensembles. Il est aussi très intéressant pour juger de la performance car c'est l'un des plus grands groupes de données.

5.1.2 Résumé des développements

Le développement du traducteur est actuellement limité au traitement des maillages.

Il s'est révélé que la version du code de sauvegarde utilisant l'opération « insert » du langage SQL pour l'ajout de données est beaucoup trop lente pour être utilisable. En effet, il faut plus de 5 minutes pour sauvegarder un maillage de mille nœuds en utilisant cette méthode. Le temps de sauvegarde augmente rapidement lorsque le nombre de nœuds augmente.

Pour résoudre ce problème, une méthode non standard (méthode qui n'est pas offerte pas le langage SQL) a été utilisée dans une deuxième version du code de sauvegarde. Cette version utilise la commande COPY disponible dans PostgreSQL pour importer directement un fichier de plusieurs lignes dans une table en une seule opération.

L'utilisation de cette commande permet de réduire de beaucoup le temps de sauvegarde d'un maillage. Avec cette méthode, le traitement d'un maillage de cent mille nœuds prend un peu moins de 30 minutes.

5.1.3 Piste de solution pour le futur

5.1.3.1 Ne plus sauvegarder les arêtes et les faces

Actuellement la sauvegarde d'un maillage est beaucoup trop lente pour être utilisable telle quelle. Une façon d'accélérer le traitement serait d'enregistrer uniquement les nœuds, les éléments et les liens entre les deux, en ignorant les arêtes et les faces.

Ces trois opérations constituent uniquement le quart du temps de traitement. Cette modification ramènerait le temps traitement des cents milles nœuds à 7,5 minutes.

Il serait plus rapide de recalculer les arêtes et les faces dans le logiciel plutôt que de les extraire de la base de données. Cependant, il ne serait plus possible d'enregistrer de l'information sur des arêtes ou des faces.

5.1.3.2 Récupérer les id automatiquement après l'insertion

L'étape qui consiste à aller chercher chacun des id pour les nœuds et les éléments insérés est celle qui demande le plus de temps. Pour diminuer ce temps il faudrait trouver une façon de pouvoir récupérer tous les id générés de manière automatique après l'insertion.

Il serait peut-être possible de diminuer légèrement le temps de traitement en demandant tous les id suivants dans une seule requête au lieu de un par un. Voici à quoi pourrait ressembler une telle requête :

ex : SELECT nextval('element_element_id_seq'), nextval('element_element_id_seq'),...;

5.1.3.3 Améliorer l'écriture dans les fichiers

Il est possible d'optimiser l'écriture dans les fichiers qui serviront de source à la commande COPY de PostgreSQL. La fréquence de ces écritures pourrait être diminuée en augmentant leur contenu. Par exemple, plusieurs insertions pourraient être mises dans un tampon avant d'être écrites dans le fichier. Présentement, il y a une écriture dans un fichier pour chacune des données à ajouter. C'est seulement une fois que toutes les écritures ont été faites, que la commande COPY est appelée avec le fichier source en paramètre.

Il faudrait également changer le mécanisme qui utilise la fonction dos2unix pour convertir les fichiers avant d'appeler la commande COPY de PostgreSQL. Si possible, il faudrait éviter de passer par l'intermédiaire d'un fichier, et utiliser plutôt un « stream », ce qui éviterait la problématique de conversion de fichiers.

5.2 Développement du gestionnaire de données

5.2.1 Résumé des discussions

5.2.1.1 Rôle du gestionnaire de données

Le gestionnaire de données : BDGestionnaireDonnee sert d'interface entre les différents modules et les données contenues dans les tables de la base de données. Ce module a pour rôle de répondre aux requêtes de création, de chargement, de sauvegarde et de destruction des objets de données. Ces requêtes sont reçues via le bus d'événements.

Par exemple, un module désirant chargé une donnée envoie un événement de chargement, lequel contient un pointeur à void. Après l'opération de chargement, ce pointeur pointera vers un objet c++ qui aura été créé pour héberger la donnée. L'opération de chargement d'une donnée provoque le chargement de toutes les données qui en sont dépendantes.

5.2.1.2 GDAlgoIo

Lors d'une requête, le gestionnaire de données utilise une « object factory » pour procéder à la création de l'objet d'algorithme GDAlgoIo approprié au traitement demandé. Ce GDAlgoIo pourra lui-même demander à « l'object factory » de créer un objet GDAlgoIo pour exécuter un sous traitement.

Il existe présentement des classes GDAlgoIo pour les composants graphiques : GDAlgoIo Espace, GDAlgoIoPlan, GDAlgoIoCoucheMaillage, GDAlgoIoCoucheNoeuds. De telles classes devront être développées pour tous les objets de données que le gestionnaire de données aura à héberger.

Voici un exemple qui permet de clarifier la discussion :

Le module de visualisation demande le chargement d'un plan graphique qui contient une couche d'information de type maillage.

Lieu du traitement : Gestionnaire de requêtes
Le gestionnaire de données reçoit la demande de chargement d'un plan graphique. Il procède à la création d'un objet GDAlgoIoPlan (via « l'object factory) sur lequel est appelée la méthode de chargement avec l'id du plan à charger.

Lieu du traitement : GDAlgoIoPlan
Cette méthode procède à la création d'un objet RGPlan et au chargement de ce plan et de ses attributs. Le chargement de ce plan nécessite de charger une couche de type maillage. Il y a création d'un objet GDAlgoIoCoucheMaillage (via l'object factory) sur lequel est appelée la méthode de chargement avec l'id de la couche à charger.

Lieu du traitement : GDAlgoIoCoucheMaillage
Cette méthode procède à la création d'un objet RGCoucheMaillage et au chargement de la couche et de ses attributs. Le chargement de cette couche nécessite de charger un maillage. Il y a création d'un objet GDAlgoIoMaillage (via l'object factory) sur lequel est appelée la méthode de chargement avec l'id du maillage à charger.

Lieu du traitement : GDAlgoIoMaillage
Cette méthode procède à la création d'un objet EFElementsFinis (maillage) et au chargement du maillage.

Le diagramme de classes actuel et celui qui pourrait devenir le futur diagramme de classes est disponible à l'ANNEXE 5 – Diagrammes de classes.

5.2.2 Résumé des développements

Pour valider le fonctionnement des différents modules, il a été nécessaire d'implanter le chargement du maillage. De cette façon, il y a eu démonstration qu'il était possible d'aller rechercher l'information dans la nouvelle base de données relationnelle.

Il a fallu recréer un maillage dans l'état exact où il était lors de sa sauvegarde. Cette partie a permis de faire le lien avec le module de visualisation, lui permettant d'afficher un maillage obtenu à partir de la nouvelle base de données.

Le chargement du maillage est fait dans la classe GDAlgoIoMaillage. Le chargement du maillage consiste à créer un objet maillage et à le remplir avec toutes les informations le concernant dans la base de données. La première étape est d'insérer tous les nœuds du maillage. Pour ce qui est des éléments, deux solutions on été testées.

La première solution consiste à faire une requête à chaque élément pour leur demander la liste de leurs nœuds. Cette solution s'est avérée assez lente à cause des requêtes fréquentes à la base de données.

La seconde solution consiste à faire une seule grande requête sur tous les éléments appartenant à un maillage, et en les triant par ordre d'éléments et par ordre de nœuds. De cette façon, le travail de classement des données est fait dans le logiciel au lieu d'être fait

dans la base de données. Cette solution est beaucoup plus rapide et offre des temps de traitement acceptables.

5.2.3 Pistes de solution pour le futur

Le chargement n'est pas vraiment problématique car il est standard et compatible avec la plupart des bases de données. Il aura quand même à subir quelques modifications.

En effet, il a été décidé d'utiliser des vues pour faire les « INNER JOIN » dans la base de données au lieu de les faire dans la requête elle-même. Ceci permettra de simplifier la requête de l'extérieur.

5.3 Analyse des problèmes de compatibilité entre les bases de données

Au cours du développement du code plusieurs problèmes de compatibilité avec les bases de données sont survenus. Souvent, en développant avec un système de base de données puis en testant ensuite sur un autre, il y avait des différences qui influençaient la façon de faire les requêtes. Par exemple, la base de données Access présente beaucoup de différences particulières en comparaison avec les autres systèmes de base de données.

5.3.1 Résumé des discussions

Suite à cette découverte, le design de BDConnection et des GDAlgoIo a été repensé pour répondre à cette nouvelle problématique.

5.3.1.1 BDConnection

L'objet BDConnection a actuellement uniquement un rôle d'initialisation. Il deviendra une interface qui implantera des versions compatibles avec le standard SQL (supporté par toutes les base de données). Les opérations de lecture et d'écriture seront donc faites via cette classe.

Une implantation spécialisée de cette interface pourra ensuite être faite pour chacune des bases de données. Cette optimisation sera intéressante lorsqu'il y aura une façon différente ou une version plus rapide pour accomplir une certaine action pour une base de données en particulier.

5.3.1.2 GDAlgoIo

Dans la même optique, certains GDAlgoIo deviendront des interfaces qui seront implantées de façon différente pour certains types de base de données.

La sauvegarde d'un maillage constitue un bon exemple de cette approche. La version qui utilise l'opération SQL « insert » est standard, mais elle est très lente. Elle constituerait donc la version de base pour la sauvegarde pour toutes les bases de données.

D'un autre côté, une implantation basée sur la commande COPY de PostgreSQL est un exemple d'une implantation d'un GDAlgoIo spécifique à chaque base de données. Cette structure de classe permettra de bien séparer le code contenant les requêtes SQL et de rendre bien structuré les différences de code nécessaires à chaque type de base de données.

Pour être en mesure de créer le bon type de BDConnection il faudra connaître le type de système de base de données utilisé. Encore une fois c'est contraire au principe d'ODBC mais l'information pourra être récupérée de la même façon que pour le nom de la base de données tel qu'il a été discuté à la section 3.2.2.

5.3.2 Problèmes de compatibilité rencontrés

5.3.2.1 Retourner les id après l'insertion

Dans la plupart des bases de données, les séquences sont utilisées pour assigner un numéro unique à une clé primaire. Il existe une technique relativement standard à la plupart des bases de données pour obtenir le numéro de séquence après une insertion.

Dans le cas de Access, c'est plutôt un numéro interne qui est incrémenté. Pour l'obtenir, il faut utiliser la fonction « @@IDENTITY » qui retourne le dernier numéro généré par cette connexion dans toute la base de données. La même fonction pourrait être utilisée pour les bases de données SQL Server.

5.3.2.2 Différence dans la gestion des types

Il y a une différence entre les types pris en charge par les bases de données. Dans le cas de Access, les numéros automatiques sont limités à 32 bits tandis qu'avec PostgreSQL, il est possible d'utiliser des BigSerial de 64 bits. Ceci a comme impact principal que les projets Access se rempliront plus rapidement que les projets PostgreSQL.

Une autre différence entre ces deux bases de données est la conversion faite pour les données de type booléen. Dans le cas de PostgreSQL, les booléens sont extraits en tant que caractère tandis qu'Access retourne une valeur numérique.

5.3.2.3 Conversion des types PostgreSQL dans OTL

La lecture des données avec OTL a causé des problèmes lors de la conversion d'un type. Les conversions de type sont gérées par le pilote ODBC de la base de données utilisée. Dépendamment de l'implantation de ces pilotes le type de conversion d'une donnée peut changer d'une version à l'autre.

Par exemple, deux versions successives des pilotes ODBC de PostgreSQL ne géraient pas la conversion des BigInt de la même manière. Dans le version 2.5, cette conversion était faite vers une chaîne de caractères. Dans la version 3.0, la conversion des BigInt est maintenant faite vers des doubles. Il est d'ailleurs possible que cette conversion soit de nouveau changée car celle-ci n'est pas encore appropriée. En effet, la représentation d'un BigInt par un double entraîne une perte de précision pour des valeurs élevées.

Dans l'implantation actuelle, lorsque le type change aussi radicalement il est nécessaire de modifier le code. Afin d'éviter cela, il faudrait donc mettre au point un mécanisme qui permettrait de choisir automatiquement le type de variable à utiliser pour extraire une donnée.

Pour y arriver il faudrait trouver les associations faites au travers de OTL. Pour PostgreSQL il semble qu'il soit possible de connaître le type a l'aide de « Explain ». Ceci reste à valider.

6 Conclusion

6.1 Résumé

Les deux premières étapes de la mise en place de la base de données relationnelle ont été d'analyser différents produits disponibles sur le marché.

Dans le cas de l'outil de design, une licence de Case Studio 2 Full a été achetée et est utilisée pour faire le design. Pour ce qui est des bases de données, elles devraient ultimement être toutes supportées, il a été déterminé que les bases de données GIS seront beaucoup plus avantageuses pour tirer profit des nouvelles fonctionnalités de Modeleur.

Le design de la base de données est assez avancé mais devra être complété par l'ajout des tables pour les autres types d'entité. Pour ce qui est des données GIS la façon de les utiliser a été définie. Le développement de code s'est fait autour de deux pôles, le traducteur et le gestionnaire de données.

6.2 Perspectives futures

Le traducteur devra être complété afin d'y intégrer les sections relatives aux autres structures de données, notamment les champs et les partitions.

Une solution devra être trouvée pour diminuer le temps de traitement beaucoup trop élevé pour la sauvegarde des maillages. Le temps de sauvegarde ne devrait pas excéder celui de Modeleur 1.0.

Le gestionnaire de données devra être modifié pour être en mesure de supporter les multiples bases de données tel qu'il a été décrit dans ce document.

Dans la solution actuelle, il y a création d'une copie en mémoire pour tous les objets. Il est possible d'imaginer un système qui serait plutôt basé sur la notion d'itérateurs sur les données. Une analyse préliminaire comparant la solution actuelle avec cette solution avec itérateurs a d'ailleurs été faite (voir ANNEXE 6 – Analyse de l'utilisation d'itérateurs).

7 Glossaire

Open GIS: Geographical Information System. C'est une méthodologie standard pour interagir avec des données de type géographique.

OTL: Librairie permettant d'interfacer avec une base de données en utilisant des « streams ».

ODBC: Open DataBase Connectivity (ODBC). C'est un API (Application Programming Interface) qui permet aux programmeurs d'interfacer avec toutes les bases de données de manière identique.

Projection: Technique visant à projeter la surface de la terre ou une portion de celle-ci sur une surface plane.

SQL: Structured Query Language. C'est le langage standard pour interagir avec une base de données.

ANNEXE 1 – Outils de design de bases de données

Analyse #1							
Programme de design	PowerDesigner	Dezign for databases	Case Studio 2 Full	Case Studio 2 Lite	dbPAL	Table designer	Visual Case
Compagnie	Sybase	Datanamic	Charonware	Charonware	It-map	Coolstrategy	Artiso Corp
Prix	2 995 $	149 $	329$ - 235$ U	165$ - 117$ U	1 985 $	40 $	89$ U
Reverse Engineering	X	109$ / les 4 DB	X			X	
Oracle	X	X	X	X	X		X
SQL Server	X	X	X	X	X	X	X
PostgreSQL		X	X	X			
MySQL		X	X	X	X		X
Access	X	X	X	X	X	X	X
Sybase		X	X	X	X		X
DB2	X	X	X	X	X		
InterBase		X	X	X	X		
Ingres			X	X			
Informix	X	X	X	X			
Demo	45 jours	Nombre limite d'entité	6 entité	6 entité	Oui	30 jours	30 jours
	40+ DB... Quelle?						

Analyse #2					
Programme de design	Dezign for databases	Case Studio 2 Full	Case Studio 2 Lite	DDS	ERCreator
Compagnie	Datanamic	Charonware	Charonware	Chili Source	modelcreator
Prix	149 $	329$ - 235$ U	165$ - 117$ U	80$ - 40$ U	149 $
Reverse Engineering	109$ / les 4 DB	X			X
Oracle	X	X	X	X	X
SQL Server	X	X	X	X	X
PostgreSQL	X	X	X	X	
MySQL	X	X	X	X	X
Access	X	X	X	X	X
Sybase	X	X	X	X	
DB2	X	X	X	X	X
InterBase	X	X	X	X	X
Ingres		X	X	X	
Informix	X	X	X	X	
Demo	Nombre limite d'entité	6 entité	6 entité	20 jours	30 jours

ANNEXE 2 – Systèmes de bases de données

DB	Oracle / Lite	SQL Server	PostgreSQL	MySQL	Access	ASE	DB2	InterBase	Informix
Compagnie	Oracle	Microsoft	UC (Berkeley)	MySQL AB	Microsoft	Sybase	IBM	Borland	IBM
Prix Processeur	15 000 $	4 999 $	Gratuit	Gratuit	Office	995 $	873 $	200 $	
Prix par usager	5 000 $	1489 $ / 5	Gratuit	Gratuit	Office	195 $	324 $	150 $	
Prix developer	?	499 $	Gratuit	Gratuit	Office	195 $	48 $	?	
Prix éducation	?	1059$ / 10	Gratuit	Gratuit	Office	?	?	?	
SQL	Près SQL	?	Près SQL 92 et 99	Imp ANSI SQL 99	Non standard	?	?	Comforme SQL 92	Objet
Stock LOB	Oui	?	Oui (bytesa)	Oui (blob)	Non (OLE?)	?	Oui	Oui (MySQL)	
Système	All	Win (NT et +)	All (native Unix)	All	Win + Mac	Win NT4 + Unix	Win + Unix	Win + Unix	
Installation	Setup	Setup	Code ou cygwin	Setup	Setup	Setup	Setup	Setup	
Backup	Dump	Dump	Dump	Dump (Offline)	Copie	?	?	?	
Update	Setup/Rare	Setup/Rare	Code/Frequent	Code/Frequent	Setup/Rare	Setup/Rare	Setup/Rare	Setup/Rare	
ODBC	Oui	Oui	Oui	Oui	Oui	Oui	Oui	Oui	
GIS	Oui (Spatial)		Oui (postGIS 3D)	Oui (v 4.1)			Oui		
				Pas subquery					Info?

ANNEXE 3 – Enregistrer des données GIS

Solutions envisagées

Solution A : Avoir une colonne pour chaque projection

Faire une seule table pour chaque élément utilisant les GIS mais qui contient plusieurs colonnes pour enregistrer la donnée GIS.

Il faudra alors soit :

-Aller chercher toutes les colonnes de la table et trouver celle qui contient une valeur.
-Associer une projection à la table de l'objet (maillage et autre) pour spécifier dans quelle projection aller chercher les données de la table GIS.

Désavantage :
On ne peut pas facilement s'assurer qu'une seule des colonnes est utilisée pour un enregistrement de la table. Il faudrait utiliser des triggers.

Solution B : Avoir une table pour chaque projection

Faire plusieurs tables pour enregistrer les éléments qui sont dans la même projection ensemble dans une même table GIS.

Il faudra alors :
La table de l'objet (maillage et autre) se retrouvera à pointer vers plusieurs tables. Il faudrait donc avoir aussi plusieurs tables qui font les liens. Par exemple entre la table élément et les différentes tables de nœuds pour chaque projection. Les schémas pourraient être utilisés pour séparer les tables par projection.

Désavantage :
Peut faire un grand nombre de tables identiques si plusieurs projections sont utilisées.
Oblige à ramener l'information de la projection au niveau de l'objet. Il y a donc une duplication de l'information.

Solution C : Avoir une table pour chaque objet

Faire une table différente pour chacune des tables GIS en dessous d'un objet.

Il faudra alors soit :
Utiliser des noms de tables générés automatiquement que l'on utilise dans nos requêtes. Enregistrer les différents noms de tables dans une table pouvant faire les associations. Les schémas pourraient être utilisés pour séparer les tables d'un objet.

Désavantage :
Il en résulte un nombre effroyable de table.

Solution D : Ne pas utiliser le standard de GIS et permettre de tout enregistrer dans la même colonne

Conserver le modèle de base de données actuel.

Il faudra alors :
Contourner le processus de création des colonnes GIS.

Désavantage :
Nous ne pourrions pas utiliser les fonctions des GIS pour rechercher les données et perdons alors tous leur avantages.

Solution E : Ramener toutes les informations enregistrées dans une projection donné

Toutes les données peut alors être enregistrées dans la même table.

Il faudra alors :
Convertir les données reçues avant d'être enregistrées.

Désavantage :
Perte de précision car les données initiales sont modifiées avant d'être enregistrées.

Problèmes communs

Design très mauvais parce que la structure même de la base de données est variable.

Solution A : Des nouvelles colonnes seront rajoutées pour une nouvelle projection.
Solution B : Des nouvelles tables seront rajoutées pour une nouvelle projection.
Solution C : Des tables seront créées à chaque insertion de données.

Structure de la base de données rend celle-ci impossible à lire par une source extérieure.

Solution A : Complexifie légèrement la lecture des données GIS.
Solution B : Complexifie grandement la lecture car il y a plusieurs tables similaires.
Solution C : Complexifie grandement la lecture car il faut trouver directement sa donnée et non plus une table qui contient toute les données.

Liste variable de colonnes ou de tables qui rend difficile la formation des requêtes SQL.

Solution A : La requête sera différente selon le nombre de projections utilisées ou selon la projection utilisée par l'objet demandé.
Solution B : La requête sera spécifique à la projection. Les schémas pourrait être utilisés pour regrouper les différentes projections (ex : projection1.nœud, projection2.nœud).
Solution C : La requête sera spécifique à l'objet. Les schémas pourrait être utilisés pour regrouper les différents objets(ex : objet1.nœud, objet2.nœud).

Il faut effectuer la génération de tables ou de colonnes automatiquement lorsqu'elles n'existent pas.

Solution A : Si une projection est utilisée pour la première fois il faut l'ajouter à la table des projections, et rajouter la colonne pour y enregistrer les données.
Solution B : Si une projection est utilisée pour la première fois il faut l'ajouter à la table des projections, et créer une nouvelle hiérarchie de tables puis y enregistrer les données.
Solution C : Pour chaque donnée enregistrée il faut créer une nouvelle hiérarchie de tables puis y enregistrer les données.

ANNEXE 4 – Installation de PostgreSQL/PostGIS

Version testée :

Postgresql	7.3.3-1
geos	cvs (25 juin 2003)
proj4	4.4.7
postgis	cvs (25 juin 2003)

Avant tout il faut avoir le "make" de GNU d'installé sous Cygwin (voir Cygwin/Devel)

1. Allez chercher le programme cygipc a l'adresse :

http://www.neuro.gatech.edu/users/cwilson/cygutils/cygipc/index.html
cd /
tar xvjf <file name>;
ipc-daemon --install-as-service

2. Downloader le code source de postgresql dans cygwin.

Il se retrouve alors dans le dossier /usr/src
Modifier : C:\cygwin\usr\src\postgresql-7.3.3-1\src\include\optimizer\cost.h
Trouver la ligne : extern double cpu_index_tuple_cost;
Remplacer par : extern DLLIMPORT double cpu_index_tuple_cost;

3. Mettre le code de geos dans le dossier /usr/src/postgresql($version)/contrib

Site de geos : http://geos.refractions.net
Adresse cvs : cvs -d :pserver:cvs@geos.refractions.net:/home/cvs/postgis

Aller dans /usr/src/postgresql($version)/contrib
cvs -d :pserver:cvs@geos.refractions.net:/home/cvs/postgis login
Le password est : cvs
cvs -d :pserver:cvs@geos.refractions.net:/home/cvs/postgis checkout geos

4. Installer GEOS

cd geos
./autogen.sh
./configure
chmod 750 depcomp
make
make install
cd ..

5. Downloader proj4

Site de proj4 : http://www.remotesensing.org/proj/
Extraire dans le dossier contrib

---

6. Installer PROJ4

cd proj-$version
./configure
make
make install
cd ..

---

7. Downloader postGIS > 0.7.5

Site de postGIS : http://postgis.refractions.net/
Adresse cvs : cvs -d :pserver:cvs@postgis.refractions.net:/home/cvs/postgis

Aller dans /usr/src/postgresql($version)/contrib
cvs -d :pserver:cvs@postgis.refractions.net:/home/cvs/postgis login
Le password est : cvs
cvs -d :pserver:cvs@postgis.refractions.net:/home/cvs/postgis checkout postgis

---

8. Installer PostGIS

Retourner dans le dossier de postgresql (cd ..)
./configure --enable-multibyte --with-CXX --prefix=/usr --sysconfdir=/etc --docdir=/usr/doc/postgresql-$version
(ou $version = 7.2, 7.3, ...)
make
cd contrib
cd postgis
Modifier le makefile :
Trouver la ligne : SHLIB_LINK += -lstdc++ -L$(GEOS_DIR)/lib -lgeos
Remplacer par : SHLIB_LINK += -lstdc++ -L$(GEOS_DIR)/lib -lgeos --driver-name g++

Trouver la ligne : SHLIB_LINK += -L$(PROJ_DIR)/lib -lproj
Remplacer par : SHLIB_LINK += -L$(PROJ_DIR)/lib -lproj --driver-name g++
Ajouter "using namespace geos;" au fichier : postgis_geos_wrapper.cpp
make

make install

---

9. Démarrage de postgres
Variables d'environnement :
PGDATA = /usr/local/pgsql/data
PGHOME = /usr/local/pgsql
PGHOST = localhost
PGLIB = %PGHOME%/lib

Si vous rencontrez des problèmes de dll introuvable. Faire une copie des dll "pq.dll" et "postgis.dll" de /lib au dossier /lib/postgresql

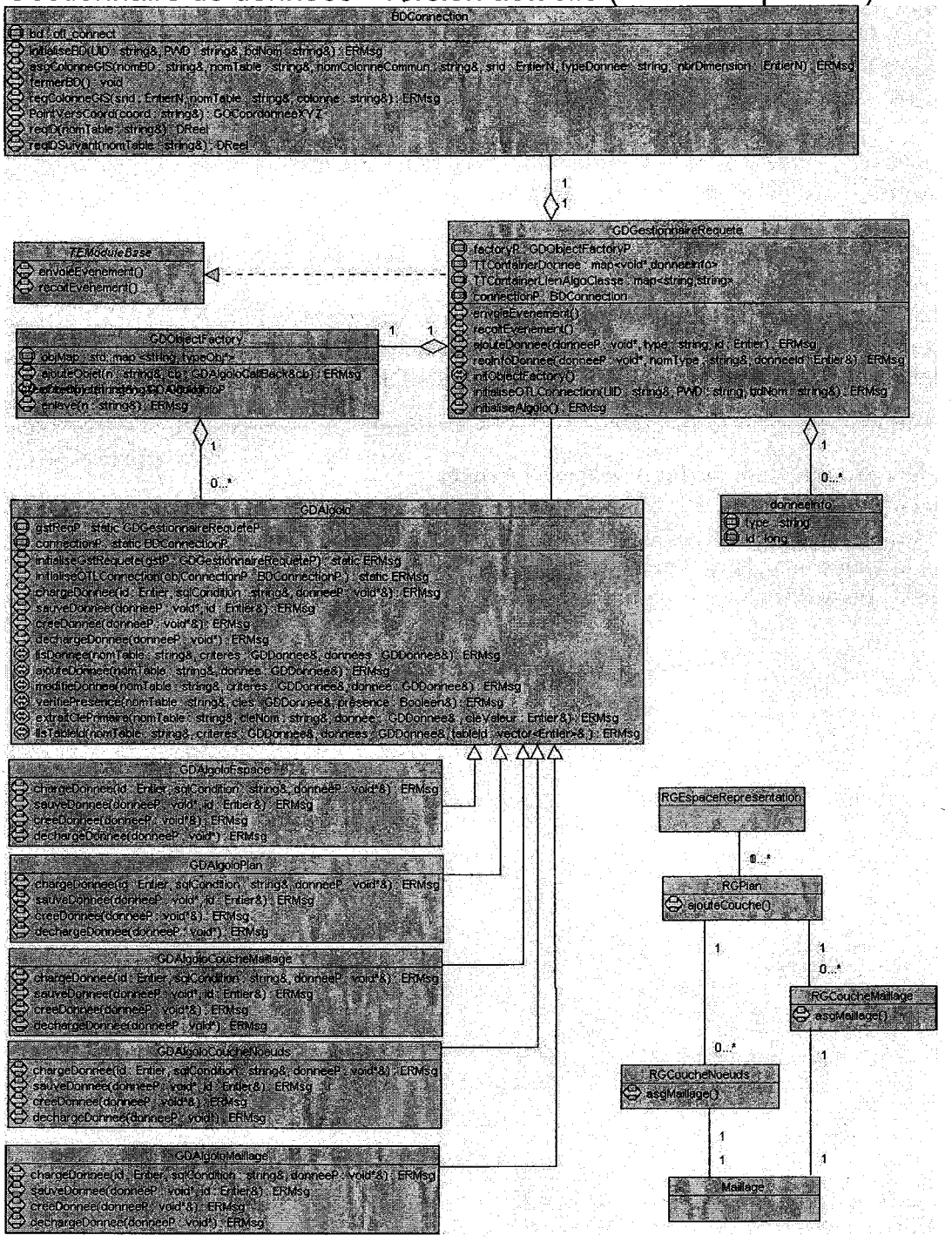
Verifier que ipc-daemon est demarre. Sinon erreur "shmget(): Function not implemented".
Faire "initdb -D {repertoire}". (Par defaut utilise le repertoire de PGDATA)
Faire "postmaster -i" pour demarrer le serveur de la base de donnees
Faire ensuite "createdb {nomDB}" (Par defaut nomDB = user de la machine)

---

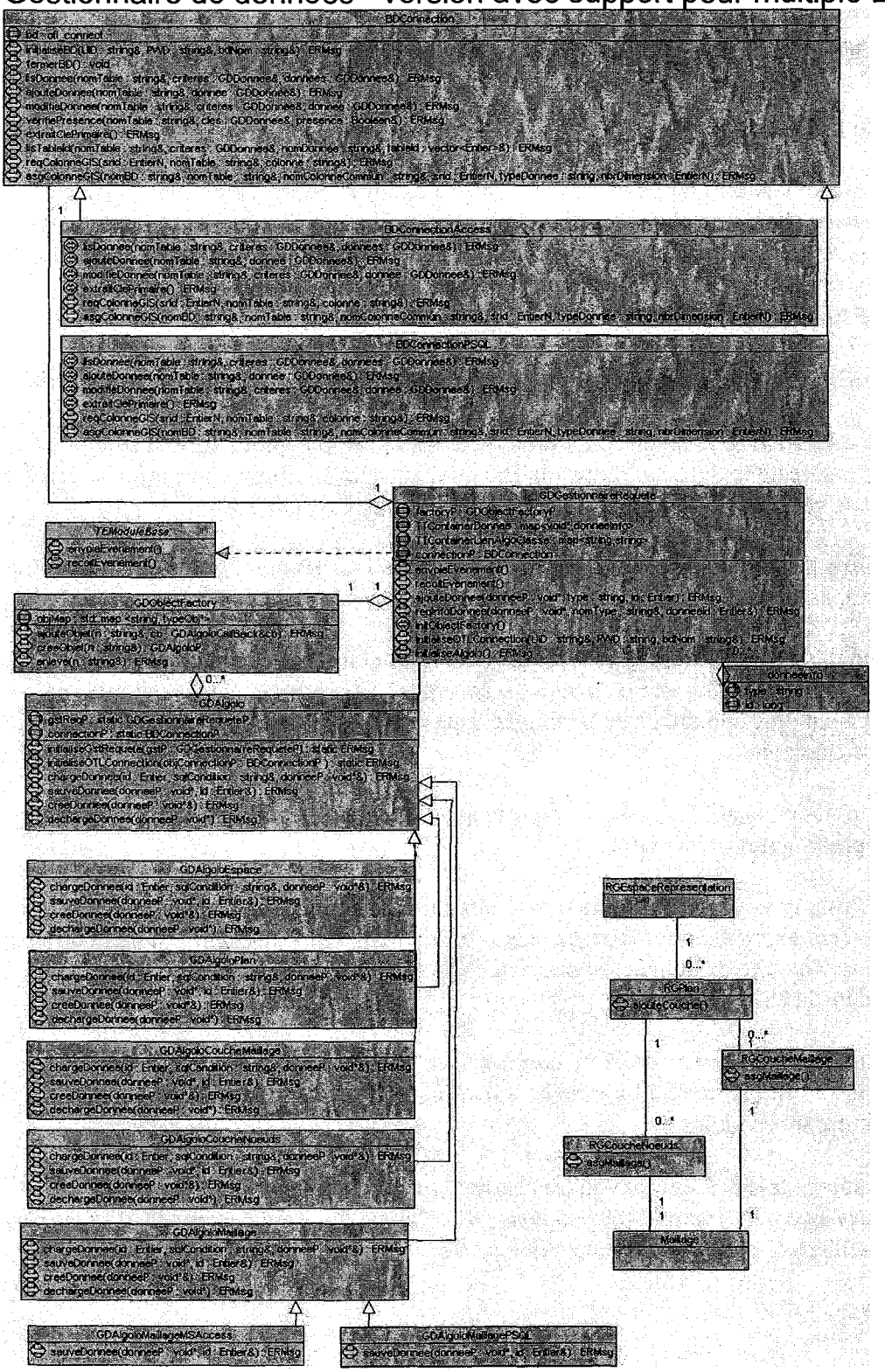
10. Rendre une base de données géoréférencée
Aller dans le répertoire de postgis.
createlang plpgsql yourtestdatabase
psql -d yourtestdatabase -f postgis.sql
psql -d yourtestdatabase -f spatial_ref_sys.sql

ANNEXE 5 – Diagrammes de classes

Gestionnaire de données - version actuelle (version implantée)



Gestionnaire de données - version avec support pour multiple BD



ANNEXE 6 – Analyse de l'utilisation d'itérateurs

Contexte actuel :

Si le module de visualisation demande le chargement d'un espace, il obtient un pointeur vers un espace chargé de plans, les plans eux-mêmes chargés de couches et les couches chargées potentiellement d'un maillage. L'outil de visualisation peut donc se servir de l'objet espace sans se poser de question. Il a accès à tous les attributs de l'espace de même que tous ceux des éléments qu'il contient.

Désavantages : BD couplée avec des itérateurs; pas d'objets en mémoire

Dans une version avec itérateur, il faudrait qu'à toutes les fois qu'un attribut soit demandé, une requête SQL soit faite à la BD pour extraire l'information : information qui ne serait pas gardée…

Pour les gros jeux de données (millions de nœuds), ceci va générer un très grand nombre de requêtes, si toutes les données doivent être chargées.

Étant donné le couplage avec la base de donnée, il faudrait qu'à chaque fois qu'un attribut aie à être modifié, une requête SQL soit faite. C'est parce que les modules ne manipuleraient plus une image de la donnée dans la BD, mais aurait plutôt un lien direct vers la vraie donnée .

Si on est toujours couplé à la BD, ça signifie que toute modification est sauvegardée; ce qui n'est pas toujours souhaitable.

En effet, l'usager veut parfois manipuler la structure de donnée sans être couplé à la base de donnée (par exemple, pour faire du calcul sur un maillage temporaire). Il ne pourra pas le faire si la structure de donnée est couplée en lecture/écriture avec la BD; il écrira toujours dans la base de données.

Chaque structure de donnée doit être couplée avec la base de données; pour pouvoir être écrite et lue. C'est contraire à l'approche qui voulait qu'on aie le moins de couplage possible entre les modules.

Dans beaucoup de cas, l'usager voudra charger tout son maillage. Il n'a donc pas d'avantages à ne pas le faire d'un seul coup. S'il s'avère qu'il n'a besoin que d'une partie de son maillage, il pourrait faire une requête GIS.

Avantages : BD couplée avec des itérateurs; pas d'objets en mémoire

Pour l'aspect visualisation graphique, ce serait plus fluide, si on fonctionnait avec des itérateurs pour les structures de données à afficher (par exemple pour faire un pan).

On a pas à tenir les autres modules au courant des changements, car il y a une zone centrale qui est la BD, partagée par tout le monde.

Solution intermédiaire

La solution avec itérateur pourrait être utilisée pour les gros jeux de données : exemple maillage, semi de points, champs, alors que la solution actuelle, pourrait être employée pour les jeux de données plus petits.

Rapport de développement logiciel

Mise en place du langage interprété

Présenté par :

Kévin Solinski (Telecom Lille)

06-01-2004

Versions du document

Date	Auteur	Commentaires	Version
19/12/03	Kévin Solinski	Version initiale	1.00
06/01/04	Kévin Solinski	Modifications (chapitre 8)	1.10

1 Notations utilisées

Avant de commencer la lecture de ce rapport, il est nécessaire de faire quelques précisions sur les notations utilisées.

- Un mot souligné de façon discontinu renvoie vers le glossaire.

Exemple : C++

- Les termes en anglais sont en italiques.

Exemple : *plug-in*

- Des remarques concernant certaines explications sont mises en forme de la manière qui suit :

Exemple :

Remarque : Texte de la remarque.

- Les blocs de code C++ ou Python sont mises en forme de la manière qui suit :

Exemple :

```
// code C++

# ou code Python
```

- Les noms (fonctions, méthodes, classes, etc...) qui font partie de code C++ ou Python sont introduits de la manière suivante :

Exemple :

La méthode `NomDeLaMethode` a pour fonction de ...

2 Introduction

2.1 Objectifs

Ce rapport a pour objectif de présenter l'avancement du dossier relatif à l'intégration du langage interprété dans Modeleur 2.0.

Ce rapport est principalement destiné aux personnes susceptibles de reprendre le travail qui a déjà été fait dans ce dossier.

2.2 Contexte

De nos jours, la plupart des logiciels sont personnalisables et extensibles (*plug-in*). Ceci dans le but de répondre à toutes les attentes des utilisateurs, en particulier celles qui n'ont pas été clairement exprimées.

C'est dans ce contexte qu'il a été décidé d'intégrer un langage interprété dans Modeleur 2.0.

Voilà les caractéristiques auxquelles doit répondre ce langage :

- *Le langage interprété doit donner la possibilité aux utilisateurs de modifier le logiciel Modeleur pour qu'il corresponde mieux à leurs besoins.*

Concrètement cela signifie que les usagers de Modeleur seront capables d'ajouter eux-mêmes les fonctionnalités dont ils ont besoin (à l'aide de scripts ou de *plug-in* qu'ils créeront).

Les usagers de Modeleur pourront également modifier l'interface graphique du logiciel. Pour cela, ils n'auront qu'à éditer des fichiers texte (scripts) pour les modifier. Aucun besoin de recompilation et de reconstruction du logiciel n'est nécessaire. Ceci est très intéressant car la reconstruction d'un logiciel à partir de son code source est souvent pénible, longue et compliquée. De plus, le code source (ici en C++) n'est pas toujours livré avec le logiciel.

Ils pourront également utiliser et modifier certaines fonctionnalités de base de Modeleur, puisqu'ils auront accès aux ressources cachées du logiciel via l'utilisation de scripts.

- *Le langage interprété sera également utilisé pour la création d'une calculatrice qui sera configurable à l'aide de scripts.*

Là encore, c'est la possibilité d'utiliser toutes les ressources de Modeleur à l'aide des scripts qui offre un intérêt non négligeable dans l'élaboration de la calculatrice. En effet, les usagers pourront faire des calculs importants en utilisant toutes les ressources mathématiques qui sont à disposition dans le code source C++ de Modeleur (Intégrales, dérivées, ...).

- *Le langage interprété permettra de sauvegarder les manipulations des usagers de Modeleur, pour pouvoir les rejouer plus tard.*

Cela correspond au principe des macros que l'on peut utiliser dans des logiciels tels que Microsoft Word et Microsoft Excel.
Pour Modeleur, cela correspond à l'enregistrement des événements qui circulent sur le bus pour pouvoir ensuite rejouer ces événements et ainsi ré-exécuter les manipulations de l'usager.

- *Le langage interprété devra donner la possibilité d'utiliser la programmation visuelle pour configurer certaines parties de Modeleur.*

Cette dernière fonctionnalité permettrait aux personnes novices en programmation de configurer Modeleur à l'aide d'outils visuels. Ainsi, ils ne leur seraient pas nécessaire de connaître la syntaxe du langage, seules quelques notions d'algorithmie leur seraient nécessaires.

- *Enfin, un langage de scripts permet de lancer une batterie de tests pendant le développement de Modeleur, ce qui est un avantage certain.*

2.3 Structure

La structure de ce document suit les différentes étapes qui furent abordées durant la mise en place du langage interprété dans Modeleur 2.0 :

- Recherche d'un langage interprété ;
- Choix des outils Python ;
- Répercussions du choix Python sur Modeleur ;
- Analyse du module (de scripts et de macros) ;
- Implémentation du module de scripts ;
- Améliorations à apporter.

3 Recherche d'un langage interprété

3.1 Analyse des besoins

Dans la première partie de ce document, il a été présenté l'intérêt d'utiliser un langage interprété. Maintenant il faut déterminer quelles sont les fonctionnalités que ce langage doit offrir.

Premièrement, il doit être interfaçable avec le C++. C'est à dire qu'il doit être possible d'utiliser ce langage interprété dans le langage compilé C++ et inversement.

Concrètement cela signifie qu'il doit être possible d'accéder à des classes, fonctions, et objets C++ à partir du langage interprété et que de la même manière il doit être possible d'accéder à des classes, fonctions, et objets du langage interprété à partir du C++.

Ce premier point implique également qu'il soit possible d'avoir à disposition un interpréteur du langage interprété à partir du C++ pour l'exécution de scripts.

Le deuxième critère auquel doit répondre ce langage interprété est l'existence d'un support pour l'interface graphique à même le langage. Cette interface graphique doit être très performante et doit utiliser les fenêtres natives de Windows.

Pour pouvoir créer rapidement des applications fenêtrées dans ce langage, il est nécessaire de trouver un bon outil de développement RAD pour cette interface graphique.

Le troisième critère du choix du langage est la simplicité globale de la solution tant pour les développeurs de Modeleur que pour ses usagers.

Concrètement ce dernier critère implique :

- que le langage soit simple à apprendre pour des non-programmeurs ;
- qu'il possède un outil de RAD simple d'utilisation ;
- qu'il possède toutes les caractéristiques que les programmeurs expérimentés peuvent en attendre (langage orienté objet par exemple) ;
- et qu'une documentation conséquente existe sur ce langage.

Enfin, la philosophie de licence du langage de scripts ne fut pas un critère important pour effectuer le choix. Seule la gratuité de la solution choisie était obligatoire, le fait qu'elle soit libre étant bien évidemment un avantage de plus.

Remarque :
La portabilité de la solution ne fut pas un critère déterminant dans le choix du langage puisque seule la compatibilité avec Windows était requise.

3.2 Comparaison des différents langages

	VB.NET	*Tcl*	*Python*	*Perl*	*Cint*
Plateforme	*Windows*	Windows Linux Macintosh	Windows Linux Macintosh	Windows Linux Macintosh	Windows Linux Macintosh
Licence	*Propriétaire*	Libre	Libre	Libre	Libre
Open source	*Non*	Oui	Oui	Oui	Oui
Interface Graphique	Windows Form	Tk	wxPython	*Non*	*Non*
Interfaçage C++	*Oui, mais très complexe*	Oui	Oui, et simplement	Oui	Oui, et très simplement
Orienté Objet	Oui	*Non*	Oui	Oui	Oui
Niveau de Complexité	*Important*	Moyen	Moyen	*Important*	*Important*
Documentation de la solution	Oui	Oui	Oui	Oui	*Inexistante*
Mises à jour régulières	*Non*	Oui	Oui	Oui	*Non*

Qtscript, Jscript, Rexx, Ruby, Lua n'ont pas été retenu car peu d'informations ont été trouvées sur ces langages et qu'ils ne possèdent pas d'interface graphique et/ou d'interfaçage avec le C++.

Après avoir fait une recherche d'environ quinze jours sur les différents langages de scripts existants, il a été décidé de n'évaluer que les 3 solutions les plus intéressantes pour Windows : VB.NET, Tcl et Python.

3.3 Choix du langage

Tcl fut éliminé très rapidement, puisqu'il n'est pas orienté objet et qu'il s'interface plus difficilement avec le C++ que Python. Seules les solutions VB.NET et Python ont dû être testées en détails.

L'interface graphique des deux solutions furent donc testées à l'aide de prototypes. Sur ces premiers tests, c'est VB.NET qui fut le plus apprécié car il permet de créer toutes types de fenêtres Windows. De plus, le RAD de VB.NET est meilleur que celui de Python (Boa Constructor).

Cependant Python permet de faire pratiquement tous les types de fenêtres de Windows à l'aide de la librairie wxPython et du RAD Boa Constructor.

La deuxième série de tests concerna l'interfaçage du langage interprété avec le C++. Les prototypes Python furent réalisés assez facilement après avoir pris en main la librairie d'interfaçage (Boost Python).

Les prototypes VB.NET, eux, furent difficiles à réaliser car le moteur de scripts (l'interpréteur du langage) ne possède que très peu de documentation. De plus, l'utilisation du C++ pur n'a pas pu être possible, et les prototypes ont été écrits à l'aide de *Managed C++* (compatible .NET).

Remarque :
Le *Managed C++* n'est compilable que sous Visual C++.NET et Borland Builder 6.0.

Critère	.NET	Python	Choix
Interopérabilité avec le C++	**CONTRE:** L'interopérabilité avec le code C++ à partir d'un script n'est pas simple à mettre en place.	**POUR:** L'interopérabilité avec le code C++ est facile à mettre en place avec Boost-Python et Pyste.	Python
Niveau de couplage	**CONTRE:** Le couplage avec .NET est important (ex : bus en .NET), ce qui est contraire à l'idée de base de l'architecture.	**POUR:** Le couplage entre le C++ et Python reste minimal. Très facile de dissocier la partie Python de celle C++. Très facile d'interfacer avec le bus, sans rien ajouter.	Python
Courbe d'apprentissage (développeurs & utilisateurs)	**POUR:** Certains développeurs stagiaires pourraient avoir une expérience de .NET. **CONTRE:** Courbe d'apprentissage abrupte pour les développeurs (stagiaires) du projet (concepts .NET, framework, « scripting »).	**POUR:** Courbe d'apprentissage relativement simple pour les développeurs (stagiaires) du projet. **CONTRE:** Peu de développeurs (stagiaires) auront de l'expérience en Python.	Python
Documentation (Scripting)	**CONTRE:** La documentation pour accéder au côté « scripting » de .NET est rare et incomplète. La documentation Microsoft est typiquement ardue à comprendre pour les non-initiés.	**POUR**: Offre beaucoup de documentation sur le Web, et d'exemples pour répondre à nos besoins. Quelques livres sont disponibles.	Python
Déboguage	**POUR**:Offre des outils fiables pour le déboguage, tous regroupés dans un même IDE. **CONTRE**: Ne permet pas de déboguer le code source de .NET.	**POUR**: Offre quelques d'outils de déboguage. Les sources de toutes librairies peuvent être déboguées/modifiées si nécessaire. **CONTRE**: Les outils de déboguage ne semblent pas tout à fait mature. Pas d'IDE offrant une solution intégrée.	.NET
Interface graphique	**POUR:** Offre un excellent RAD. Offre un éventail étendu de composants pour les interface graphiques. Peut utiliser des COM.	**POUR:** wxPython est basé sur wxWindows, librairies qui a fait ses preuves. Affiche des boîtes de dialogue natives. Peut utiliser des COM **CONTRE:** Peu de choix de RAD disponible. RAD pas encore mature. Certains composants visuels ne seront peut être pas disponibles.	.NET

Plug-in	**POUR:** L'engin Visual Studio for Application (VSA) semble prometteur pour l'ajout de plug-in. **POUR** VSA n'est pas encore disponible. L'ajout de plug-in reste donc relativement difficile.	**POUR:** L'usager peut développer son plug-in dans le langage de son choix, publier ses événements et modifier l'interface usager.	Python
Portabilité	**CONTRE:** Est limité à la plateforme Windows; ce qui limite l'expansion vers d'autres systèmes.	**POUR:** Est portable sur les autres plateformes (Linux, Unix)	Python
Configuration	**POUR:**L'installation de Visual Studio installe tout ce qui est nécessaire. **CONTRE**: Nécessite Visual Studio pour le développement de plug-in.	**POUR:** Plusieurs outils/extensions sont disponibles gratuitement (Boost-Python, Pyste, wxPython). **CONTRE**: Plusieurs logiciels / librairies à installer	Python

Après cette série de tests, Python fut choisi pour les raisons suivantes :

- Il offre une meilleure interopérabilité avec le C++.
- Le niveau de couplage avec Python reste minimal contrairement à .NET qui ne possède que deux compilateurs C++ *Managed* (Visual C++ .NET et Borland Builder 6.0)
- Le courbe d'apprentissage de Python pour les développeurs et les utilisateurs est relativement simple.
- La documentation sur Python est riche et l'accès au code source simplifie grandement la tâche des développeurs.
- Le solution Python est Portable sur Windows, Dos, Mac, OS/2, BeOS, NeXTStep, Unix, Linux, et toutes les autres variantes de Unix.
- De plus, il n'est pas nécessaire aux développeurs qui souhaitent modifier les sources ou créer un *plug-in* d'acheter des logiciels propriétaires (Visual Studio, ou Builder).

Voir Annexe 1 : VB.NET vs Python
(paragraphe 13.1, page 53)

4 Le choix des outils « Python »

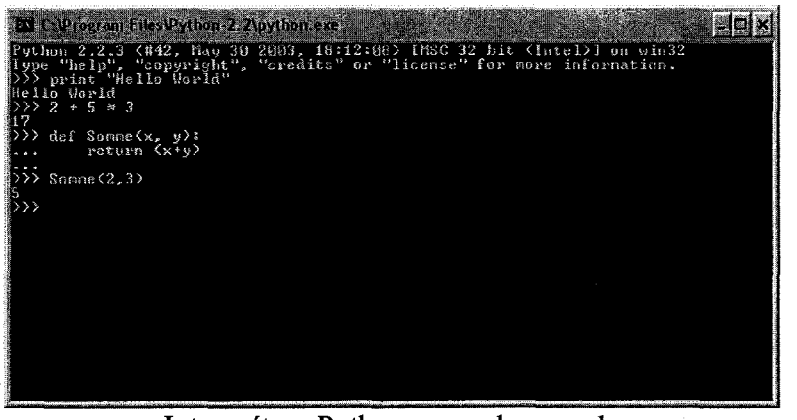
Comme l'a présenté la partie précédente, Python a besoin de « librairies » pour répondre aux besoins des usagers et développeurs de Modeleur (Voir paragraphe 3.1, page 6).

Cette troisième partie va donc présenter très succinctement Python et les outils (librairies) utilisés par l'équipe de Modeleur 2.0.

4.1 Le langage Python

Voici quelques caractéristiques de Python :

- Libre : les sources sont diffusés sous une licence compatible GPL (utilisation commerciale autorisée) ;
- Interprété (compilé en bytecode comme java ou interprété à la volée sans passer par une phase de compilation) ;
- Orienté Script ou Fonction ou Objet (au choix ou en même temps) ;
- Compatible Linux, Windows, Mac, Java (Jython permet de créer des classes java), autres (en compilant les sources) ;
- Liens possible avec du code C ;
- Nombreux modules et fonctions en standard (listes, regexp, I/O, Internet...) ;
- Nombreux modules et librairies additionnelles (Graphisme, base de données, GUI, PDF...) ;
- Syntaxe simple, claire et efficace.



Interpréteur Python en mode console

Remarque :
Il existe un document qui aide à l'installation de Python et de ces librairies sur un poste : « **Installation poste développeur.doc** » qui se trouve sur \\Caxipi\Green\Documentation\

4.2 Interface graphique : wxPython

wxPython est un Toolkit qui permet de construire des interfaces graphiques portables sur Unix, et Windows. wxPython est basé sur la bibliothèque C++ : wxWindows.

L'utilisation de wxPython se fait très simplement. En effet, il suffit de l'importer dans un script Python pour pouvoir ensuite l'utiliser.

Remarque :
Le principe de fonctionnement de wxPython ressemble à celui des MFC.

4.3 Interfaçage avec le C++ : Boost Python

4.3.1 Définition succincte

Boost Python est une librairie C++ *open_source* qui permet d'interfacer le C++ avec Python. Ainsi il est possible d'utiliser toutes les classes, fonctions, instances C++ en Python et inversement.

Remarque :
La version actuelle de Boost (1.30.2) ne prend pas encore en charge la version 2.3 (ou supérieure) de Python, c'est pourquoi la version utilisée de Python est la 2.2.3

Concrètement, il faut écrire du code C++ « Boost Python » qui indique quels éléments vont devoir être partagés en Python. Ces éléments seront ensuite incorporés dans une DLL qui pourra être importée sous Python.

Par exemple, le fichier suivant (World.h) contient du code C++ « normal* » :

```
#ifndef WORLD_H
#define WORLD_H

#include <iostream>
#include <string>

struct World
{
    void set(std::string msg) { this->msg = msg; }
    std::string greet() { return msg; }
    std::string msg;
};

#endif
```

* en comparaison au code C++ « Boost Python ».

Pour partager ce code en Python, il faut générer un fichier C++ « Boost Python » (ici hello.cpp) qui servira à créer une DLL. Cette DLL sera ensuite vue par Python comme étant un module Python à part entière.

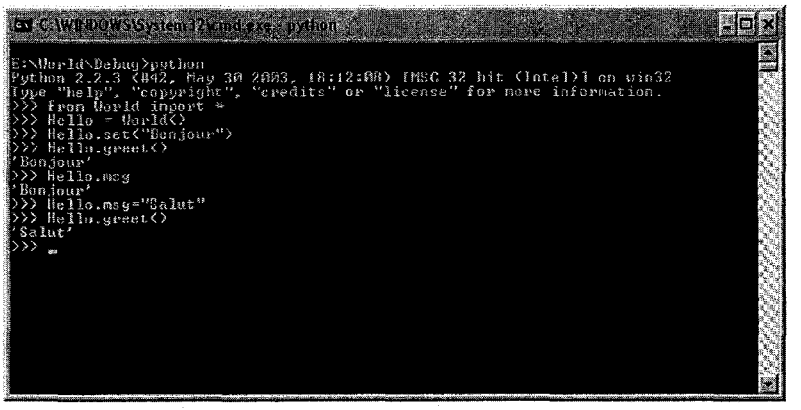
Voilà le fichier C++ « Boost Python » généré automatiquement avec Pyste (voir paragraphe 4.3.3), on peut remarquer que sa syntaxe indentée rappelle Python.

Fichier World.cpp

```
// Includes ==================================================================
#include <boost/python.hpp>
#include "world.h"

// Using =====================================================================
using namespace boost::python;

// Module ====================================================================
BOOST_PYTHON_MODULE(World)
{
    class_< World >("World", init<  >())
        .def(init< const World & >())
        .def_readwrite("msg", &World::msg)
        .def("set", &World::set)
        .def("greet", &World::greet)
    ;

}
```

4.3.2 Utilisation dans python

L'utilisation de la DLL (World.dll) générée avec les fichiers précédents (World.h et World.cpp) se fait ainsi :



4.3.3 Pyste

Pyste est un script Python qui permet de générer le fichier C++ qui va contenir le code de partage. Pour fonctionner Pyste a besoin d'un fichier texte qui lui indique quels sont les éléments C++ à partager en Python, et dans quels fichiers les trouver.

Fichier World.pyste:

```
Class("World", "World.h")
```

Voilà les lignes de commande qu'il faut exécuter pour générer le fichier World.cpp :

Fichier exec_PY_World.cmd:

```
set INCLUDE=

set PYSTE_INCLUDE=%PYSTE_INCLUDE% -I
↳                                "C:\Program Files\Microsoft Visual Studio .NET\Vc7\include"
set PYSTE_INCLUDE=%PYSTE_INCLUDE% -I "C:\Program Files\boost-1.30.2"
set PYSTE_INCLUDE=%PYSTE_INCLUDE% -I "C:\Program Files\Python-2.2\include"

python "C:\Program Files\boost-1.30.2\libs\python\pyste\src\pyste.py" %PYSTE_INCLUDE% --
module=World World.pyste
```

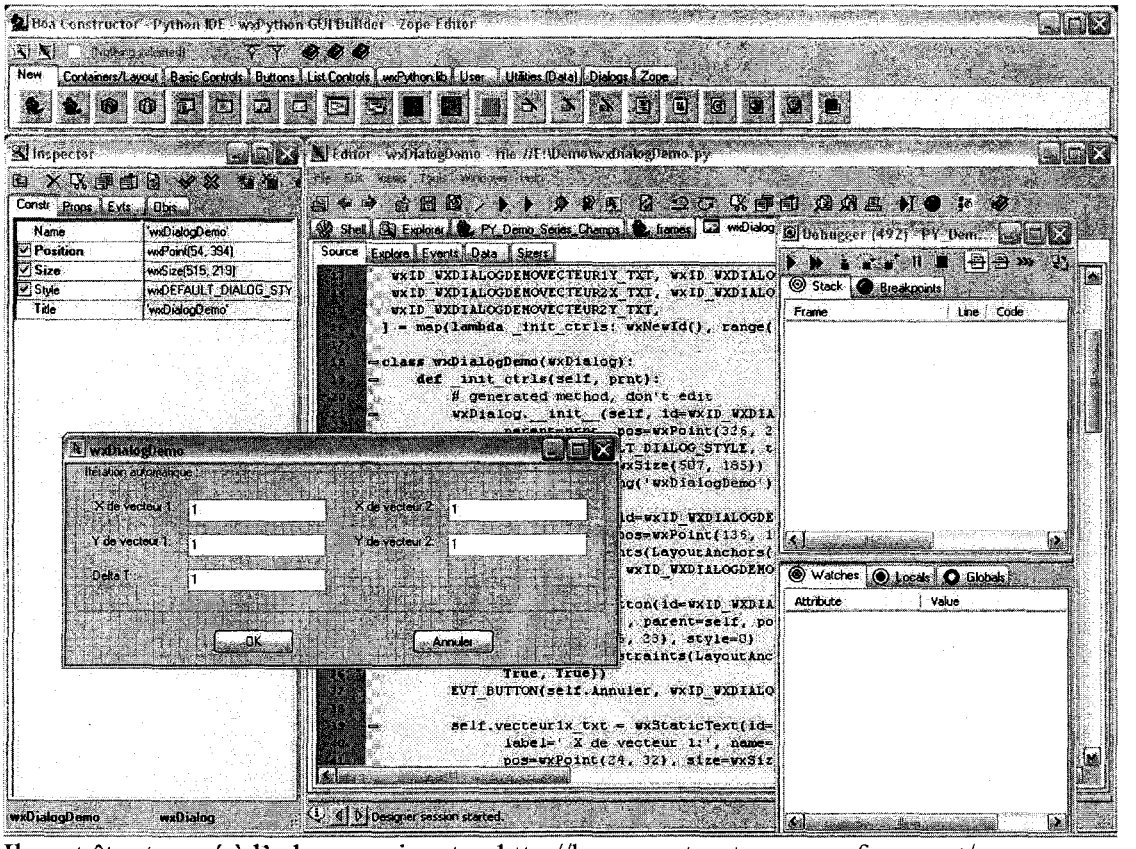
Remarques :
Seule la déclaration (fichier .h) est utilisée par Pyste pour générer le fichier C++ « Boost Python ».

Voir Annexe 2 : Utilisation de Pyste
(paragraphe 13.2, page 60)

4.4 Divers outils

4.4.1 Boa Constructor

Boa Constructor est un outil de développement RAD Python. Il permet en particulier de créer des boîtes de dialogue en wxPython. C'est un très bon éditeur, et c'est également un des meilleurs débogueurs Python.



Il peut être trouvé à l'adresse suivante : http://boa-constructor.sourceforge.net/

4.4.2 Extensions Windows pour Python

Ces extensions permettent d'avoir accès à l'API de Windows sous Python, ainsi il est possible d'utiliser des ActiveX à partir de Python ou même d'utiliser les MFC sous Python. Il peut être trouvé à l'adresse suivante :
http://starship.Python.net/crew/mhammond/

4.4.3 Les autres outils testés

Pythonwin

Cet éditeur, fourni avec les extensions Python, ne prend pas en charge les accents, ce qui est plutôt pénible pour l'écriture de commentaires. Cependant, il est assez complet et mérite que l'on s'y intéresse. Il peut être trouvé à l'adresse suivante : http://www.Python.org/Windows/Pythonwin/

SPE

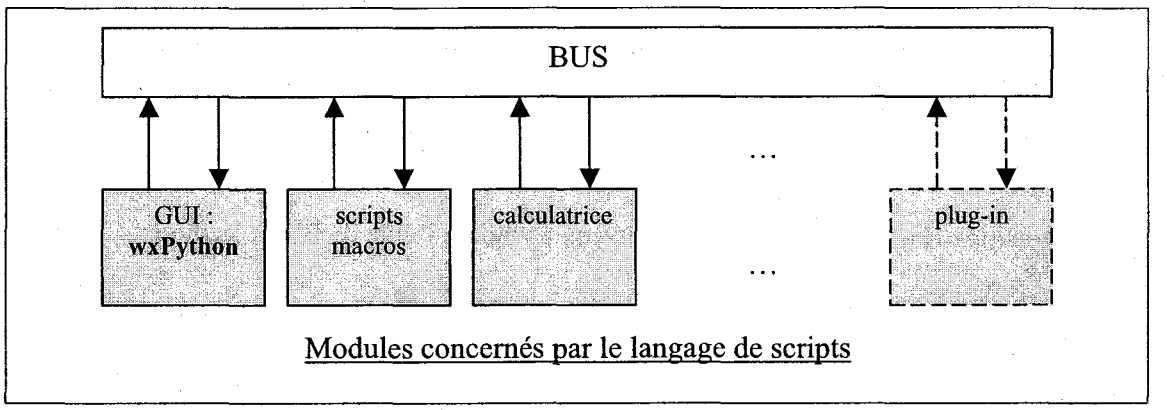
Cet éditeur assez complet est un peu bogué (la fonction annuler ne fonctionne pas très bien par exemple), mais mérite qu'on s'y attarde pour voir s'il ne possède pas d'autres améliorations. Il peut être trouvé à l'adresse suivante : http://spe.pycs.net/

Hap Debugger

Ce débogueur étant l'un des plus complets, il méritait d'être cité ici. Il peut être trouvé à l'adresse suivante : http://hapdebugger.sourceforge.net/

5 Modeleur et Python

Dans cette quatrième partie, il sera présenté les répercussions qu'a Python sur certains modules de Modeleur.



Modules concernés par le langage de scripts

On peut noter que le langage de scripts servira également à pouvoir configurer certains aspects du logiciel tels que les partitions génériques et les champs analytiques par exemple.

5.1 Partage de classes C++ en Python

5.1.1 Le pont entre le bus C++ et l'envoi d'événements en Python

Le bus C++ distribue les événements que chaque module définit. Chaque module reçoit tous les événements qui circulent sur le bus, libre à lui ensuite de traiter ces événements ou non.

Les événements de chaque module « Modeleur » sont définis en C++, cependant il est nécessaire de pouvoir les utiliser en Python puisque c'est Python qui définit l'interface graphique de Modeleur.

Deux solutions se sont présentées pour répondre à ce besoin :

- *Créer les événements à partir de C++ et appeler une fonction C++ exportée en Python (à l'aide de Boost Python).*

Cette première solution n'a pas été retenue car elle imposait de devoir créer pour chaque événement une fonction C++ qui ait pour tâche de l'envoyer sur le bus.

Exemple :

```
EnvoieEvenementAAA();
EnvoieEvenementAAB();
EnvoieEvenementAAC();
```

Cette fonction devant ensuite être exportée en Python à l'aide de Boost Python.

- *Créer les événements à partir de Python.*

Cette seconde solution consiste à exporter toutes les classes d'événements C++ en Python à l'aide de Boost Python et de Pyste. Ensuite l'appel d'une même méthode C++ `envoieEvenement` permet d'envoyer un événement sur le bus. Cette méthode ne fait qu'envoyer des événements de base `TEEvenement` sur le bus C++, mais comme tous les événements C++ ont pour parent l'événement de base, la méthode `envoieEvenement` envoie tous les événements sur le bus quelque soit leur type. C'est cette seconde solution qui a été retenue, du fait qu'elle ne nécessite pas la création d'un grand nombre de fonctions en C++.

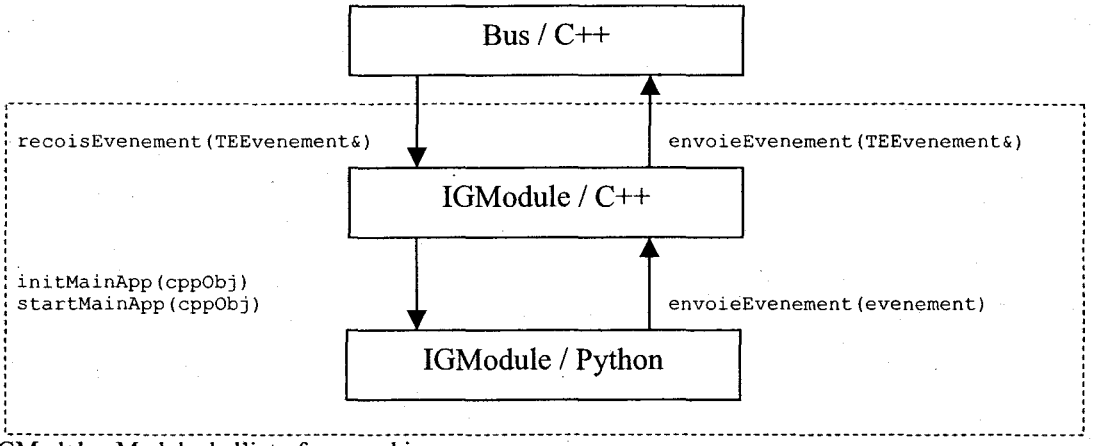
5.1.2 Module de base

D'autres classes sont également exportées en Python. Il s'agit des classes relatives à la création de modules « Modeleur », en particulier de `TEModuleBase` et de ses méthodes `envoieEvenement` et `recoisEvenement`. L'exportation en Python de cette classe permet de créer des modules « Modeleurs » uniquement à l'aide de Python, ce qui peut être très intéressant pour l'écriture de *plug-in* par exemple.

5.2 L'interface graphique

Un autre module qui dépend directement du choix du langage de scripts est l'interface graphique puisque celle-ci doit être écrite dans ce même langage. C'est la librairie wxPython qui a été choisie pour réaliser l'interface graphique de Modeleur.

L'interface graphique est responsable de l'envoi des événements du module de l'interface graphique, elle doit donc être capable de créer des événements à partir de Python.



IGModule : Module de l'interface graphique.

Le module C++ de l'interface graphique dérive d'un module de base C++ (`TEModuleBase`). Cette partie C++ peut donc envoyer et recevoir des événements sur le bus C++.

```
class DLL_IMPEXP(MODULE_TRANSMISSION_EVENEMENT) TEModuleBase
{
public:
…
            ERMsg envoieEvenement   (TEEvenement&);
            ERMsg recoisEvenement   (TEEvenement&);
…
}

class DLL_IMPEXP(MODULE_INTERFACE_GRAPHIQUE) IGModule : public TEModuleBase
{
…
}
```

Pour envoyer des événements à partir de l'interface graphique wxPython, il faut pouvoir avoir accès à la méthode `envoieEvenement` de `IGModule` en Python. Pour se faire, on exporte cette méthode C++ en Python à l'aide de Boost Python. Il faut également pouvoir avoir accès à l'instance C++ de `IGModule` à partir de Python, c'est pourquoi la partie C++ de `IGModule` appelle une fonction Python `initMainApp` exportée à l'aide de Boost Python. Cette fonction Python a pour rôle d'enregistrer l'instance C++ de `IGModule` dans un singleton Python.

Pour envoyer des événements à partir de wxPython, il suffit d'appeler la fonction Python `envoieEvenement` de `IGModule` qui va appeler la méthode `envoieEvenement` de l'instance C++ de IGModule. Cette instance est contenue dans un singleton Python.

5.3 Scripts (et macros)

Le choix du langage interprété a bien évidemment le plus de conséquences sur l'écriture et la gestion des scripts (et des macros).

Cette gestion des scripts et des macros a été conçue dans un Module spécifique : le module de scripts. Plus d'informations sur ce module sont disponibles dans la suite de ce document.

5.4 Gestion modulaire (plug-in)

Les répercussions du langage interprété se retrouvent dans la quasi totalité des modules « Modeleur » (et de ses *plug-in*).

Ceci pour deux raisons :

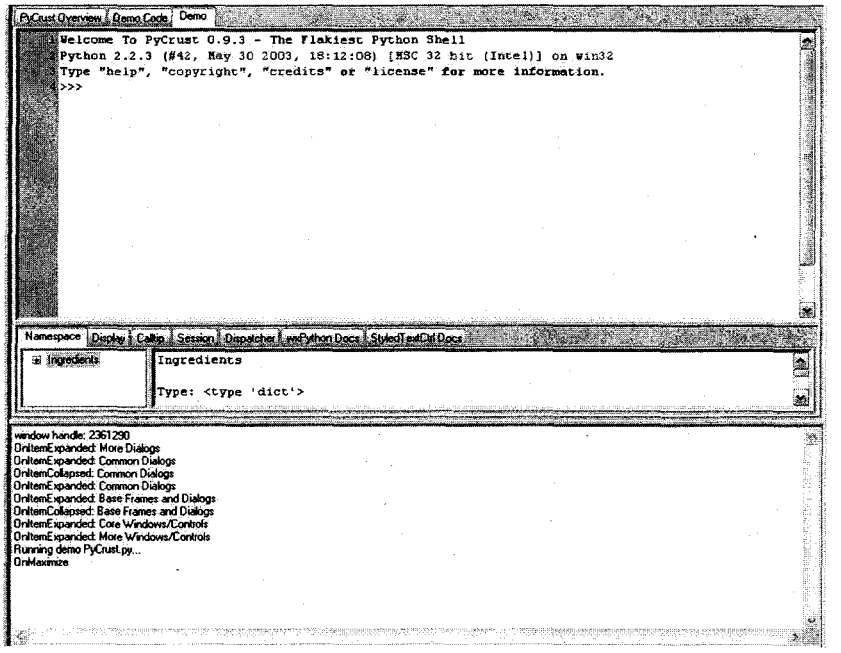
- l'interface graphique étant en wxPython, il est nécessaire que les modules « Modeleur » qui souhaitent avoir une interface graphique le fassent en Python. Ces modules doivent en effet définir des événements spécifiques à l'interface graphique pour par exemple gérer la création, l'activation, la désactivation des entrées de Menus dont ils ont besoin.

- certaines fonctionnalités C++ ayant besoin d'être exportées en Python, il est nécessaire d'utiliser Boost Python pour pouvoir utiliser ces classes C++ en Python.

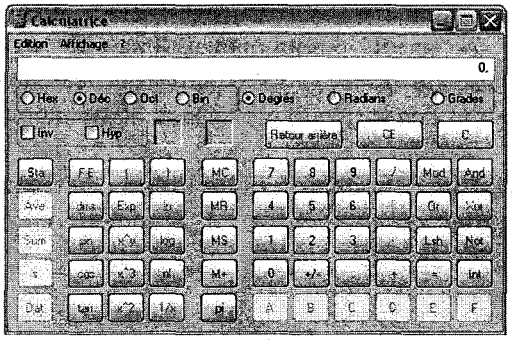
5.5 La calculatrice

La calculatrice de Modeleur utilisera le langage Python ce qui permettra d'avoir accès à toutes les fonctions, et classes C++ à partir de la calculatrice. Ainsi toutes sortes de calculs pourront être effectués.

Pour réaliser la calculatrice dans Modeleur, il est possible de réutiliser l'interpréteur Python tel quel pour les utilisateurs expérimentés. Cela pourrait s'implanter en utilisant PyCrust qui est fourni avec wxPython (voir l'aperçu qui suit).



La calculatrice devra également être facile à utiliser pour les utilisateurs novices, c'est pourquoi il faudra recréer une calculatrice qui sera basée sur un interpréteur Python, mais qui s'approchera plus de la calculatrice Windows.



6 Le Module de scripts - Spécifications fonctionnelles

6.1 Introduction

L'activité « Système de scripts » a pour but de permettre à l'usager d'automatiser le fonctionnement de Modeleur et de fournir l'accès à la plupart des fonctionnalités « cachées » de Modeleur. L'usager peut ainsi créer des scripts qu'il sera ensuite capable de rejouer.

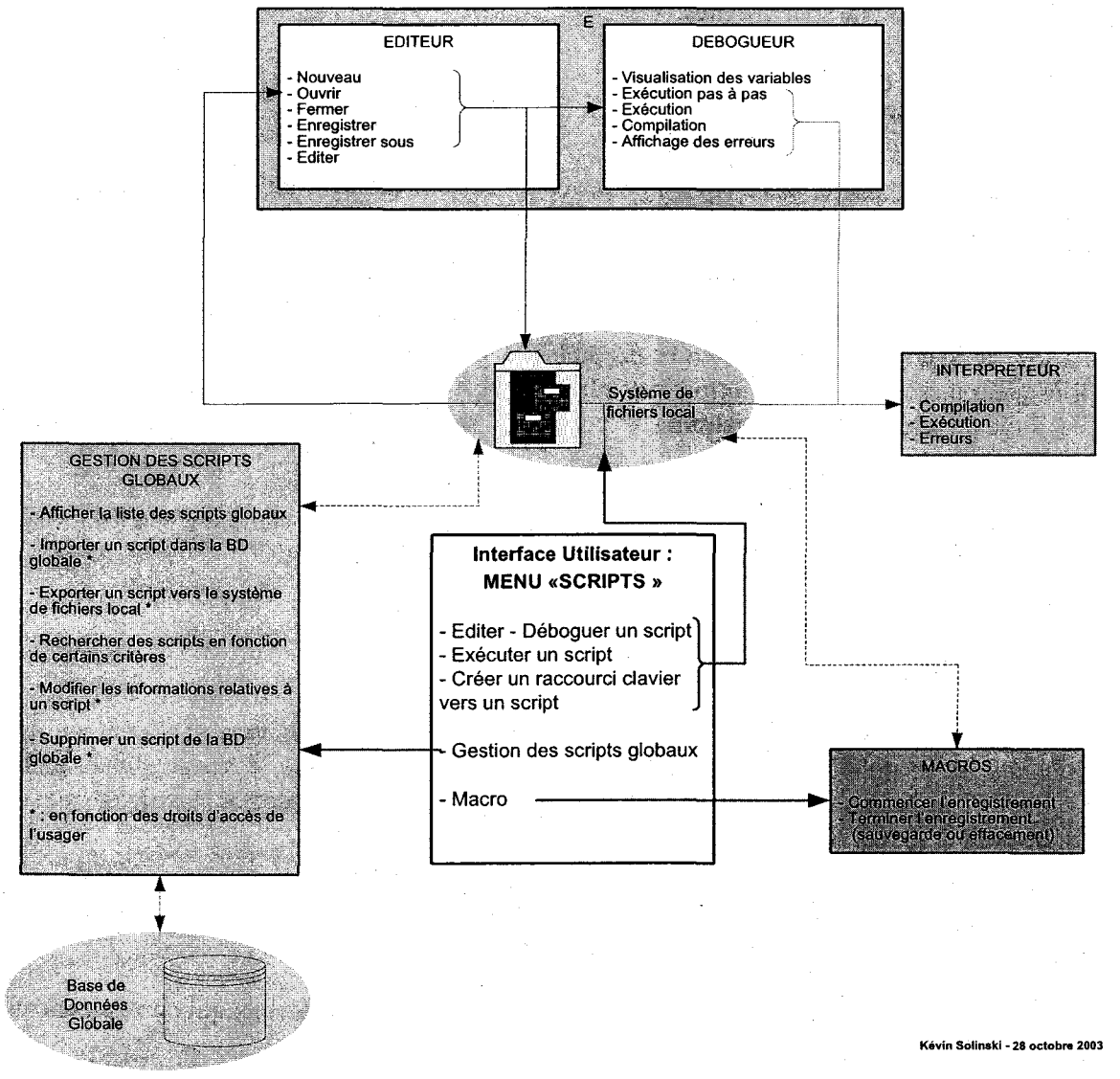
Pour écrire un script, l'usager a deux choix : soit il édite un script et écrit lui-même le contenu, soit il crée une macro. **Une macro est un script** qui enregistre automatiquement toutes les actions produites dans Modeleur pendant une période de temps définie par l'usager. Pour que l'usager puisse éditer un script, la version 2.0 de Modeleur sera constituée d'un IDE externe contenant un éditeur (et un débogueur).

Après avoir sauvegardé ses scripts sur un disque dur, l'usager peut les gérer à l'aide du système de fichiers de son système d'exploitation. Il bénéficie ainsi de toute la souplesse et facilité que ce dernier peut lui procurer. Il peut également exporter des scripts globaux vers son système de fichiers local, s'il est autorisé à le faire (droits d'accès).

Il peut s'il le souhaite et s'il y est autorisé importer des scripts locaux vers la base de données globale. Ainsi, ses scripts deviennent globaux et sont partagés entre plusieurs usagers à l'aide de la base de données globale. Seuls les usagers qui y sont autorisés peuvent importer ou détruire des scripts dans la base de données globale.

Enfin, l'usager bénéficie d'un débogueur qui lui permet de tester les scripts qu'il crée. Ce débogueur externe interagit directement avec Modeleur et fait partie d'un IDE qui contient également l'éditeur de code.

SCHEMA DES SPECIFICATIONS FONCTIONNELLES RELATIVES AU SYSTEME DE SCRIPTS



6.2 Gestion des scripts globaux

La gestion des scripts globaux se fait à l'aide d'une liste qui affiche les différents scripts qui se trouvent dans la base de données globale. Seuls les scripts correspondant à certains critères de recherche sont affichés.

6.2.1 Importer

L'usager peut importer des scripts locaux vers la base de données globale, il est pour cela invité à choisir un script Python à l'aide d'une boîte de dialogue de sélection de fichiers Python. Ce script est ensuite compilé pour avoir l'assurance de ne stocker que des scripts Python dans la base de données globale.

Après cette vérification, une boîte de dialogue de modification est affichée pour que l'usager puisse entrer des informations sur son script (nom, date, type, description). Après la validation de ces informations, le script est sauvegardé dans la base de données globale.

L'usager ne peut importer un script dans la base de données globale que s'il a les droits d'accès qui permettent cette opération.

6.2.2 Exporter

L'usager peut exporter un script de la base de données globale vers son système de fichiers local, pour cela il sélectionne dans une liste d'affichage le script global qu'il souhaite exporter vers son espace de travail local. Une boîte de dialogue l'invite ensuite à sélectionner le répertoire dans lequel son script sera copié.

6.2.3 Rechercher

L'usager peut entrer une requête SQL pour pouvoir afficher les scripts qui correspondent à certains critères de recherche.

6.2.4 Modifier

L'usager peut modifier un élément de la base de données globale en le sélectionnant, puis soit en double-cliquant dessus, soit en cliquant sur le bouton Modifier de la boîte de dialogue de gestion des scripts globaux.

La boîte de dialogue de modifications des informations relatives à un script lui permet ensuite de modifier le nom, la date, le type, la description de l'élément sélectionné.

6.2.5 Supprimer

L'usager peut supprimer un ou plusieurs script(s), s'il dispose des droits d'accès nécessaires.

6.2.6 Glisser-déposer et Raccourcis-clavier standards

L'usager peut utiliser le « glisser – déposer » ou les raccourcis classiques de son système d'exploitation (ctrl+v, ctrl+c, ctrl+x sous Windows) pour copier des scripts locaux vers la base de données globale.

L'usager est ensuite interrogé par une boîte de dialogue qui lui demande de remplir les informations relatives à chaque script qu'il a décidé d'importer dans la base de données globale.

Voir Annexe 3 : Exemple d'interface graphique pour le module de scripts (paragraphe 13.3, page 66)

6.3 Gestion des scripts locaux

6.3.1 Edition – Déboguage

L'usager sélectionne un script local à l'aide d'une boîte de dialogue classique de sélection de fichiers. Ce script est ensuite chargé dans un éditeur externe qui reste en liaison avec Modeleur pour pouvoir être déboguer dans ce dernier.

6.3.2 Exécution

L'usager sélectionne un script local à l'aide d'une boîte de dialogue classique de sélection de fichiers. Ce script est ensuite exécuté dans l'interpréteur de scripts de Modeleur. L'interpréteur de Modeleur fournit à l'usager des informations sur d'éventuelles erreurs dues à un script bogué.

6.3.3 Raccourcis clavier

L'usager peut choisir un raccourci clavier pour l'exécution d'un script qu'il aura sélectionné à l'aide d'une boîte de dialogue de sélection de fichiers. Ainsi, à chaque appel du raccourci clavier, un certain script sera appelé.

6.3.4 Macros

6.3.4.1 Commencer l'enregistrement

L'usager peut enregistrer les actions qu'il produit dans une macro. Pour cela, il doit débuter l'enregistrement de ces actions.

6.3.4.2 Terminer l'enregistrement

De la même manière, l'usager doit indiquer à Modeleur quand il souhaite que l'enregistrement de ces actions prenne fin. Quand l'enregistrement est terminé, il est invité à nommer la macro qu'il vient de créer, s'il le désire, il peut décider d'effacer cette macro (annulation de l'enregistrement).

Cette capture des actions de l'usager crée un script automatiquement (macro), ce script peut ensuite être modifié par l'éditeur de code précédemment cité.

6.3.4.3 Pause / reprise de l'enregistrement

L'usager peut interrompre l'enregistrement de la macro lorsqu'il ne souhaite pas enregistrer certaines de ses actions.

6.4 IDE externe

6.4.1 Editeur

Un éditeur externe est intégré à Modeleur pour faciliter la modification des scripts par l'usager.

6.4.1.1 Caractéristiques nécessaires pour l'intégration dans Modeleur

Cet éditeur doit être capable de se lancer avec en argument le script à ouvrir, il doit également pouvoir recevoir à tout moment le nom d'un script à ouvrir.

6.4.1.2 Nouveau

L'éditeur permet à l'usager de créer un nouveau script.

6.4.1.3 Ouvrir

L'éditeur permet à l'usager d'ouvrir un script existant.

6.4.1.4 Editer

L'éditeur fournit à l'usager toutes les fonctionnalités d'un bon éditeur de code : auto indentation, coloration des mots clé, auto complétion, etc...).

6.4.1.5 Enregistrer

L'usager peut sauvegarder les modifications apportées à son script lorsqu'il le souhaite.

6.4.1.6 Enregistrer sous

L'usager peut sauvegarder sous un autre nom le script qu'il était en train de modifier. Cette fonctionnalité permet d'enregistrer les modifications dans un nouveau script sans avoir à écraser l'ancien.

6.4.1.7 Fermer

L'usager peut fermer un script.

6.4.1.8 Quitter

L'usager peut quitter l'éditeur de script externe à tout moment pour revenir dans Modeleur.

6.4.2 Débogueur

Après avoir créé ou modifié un script, l'usager peut tester le bon fonctionnement de son script à l'aide d'un débogueur externe intégré à l'éditeur de code.

6.4.2.1 Caractéristiques nécessaires pour l'intégration dans Modeleur

Ce débogueur doit être capable d'exécuter les « scripts à déboguer » dans l'interpréteur Python de Modeleur.

6.4.2.2 Compiler

L'usager peut compiler pour voir s'il n'y a pas d'erreurs de syntaxe.

6.4.2.3 Exécuter pas à pas

L'usager peut exécuter ligne par ligne son script et voir les problèmes éventuels.

6.4.2.4 Visualisations de l'état des variables

L'usager peut, pendant le déboguage, voir la valeur (l'état) des différentes variables Python.

6.4.2.5 Exécuter

L'usager peut exécuter le script en entier, pour éventuellement s'arrêter sur une ligne fautive lors de l'apparition d'un bogue.

6.4.2.6 Affichage des erreurs

L'usager bénéficie d'informations utiles lors du déboguage telles qu'un bon affichage des erreurs que renvoie l'interpréteur.

7 Macros - Analyse

Dans cette partie, il sera énuméré les divers problèmes rencontrés lors de l'analyse des macros du module de scripts et les solutions retenues.

Avant de présenter les différentes solutions étudiées pour l'enregistrement des événements, il est bon de rappeler la direction prise lors de l'analyse des macros. Notamment, il fut décidé qu'une macro serait un script Python. Ce qui permet aux utilisateurs de Modeleur d'avoir la possibilité de modifier leur macro (en changeant la valeur des paramètres des événements inclus dans une macro).

7.1 L'enregistrement d'une macro

Une macro est construite dès que l'utilisateur souhaite enregistrer toutes les manipulations qu'il effectue pour pouvoir ré-exécuter ces manipulations en une seule commande.

Pour connaître les commandes que l'utilisateur effectue, il est nécessaire d'enregistrer les événements qui circulent sur le bus. De cette affirmation découle plusieurs problèmes :

- *Comment enregistrer les événements qui circulent sur le bus ?*
- *Comment n'enregistrer que les événements pertinents ?*
- *Comment recréer les événements contenus dans une macro ?*

7.1.1 L'enregistrement des événements

Un événement est l'instance d'une classe C++ dérivée de la classe C++ de l'événement de base : `TEEvenement`. Enregistrer les événements consiste donc à trouver le moyen de sauvegarder tous les attributs de l'objet C++ à un instant donné.

7.1.1.1 1ère Solution : Enregistrement des événements en C++

La première solution pour sauvegarder les attributs d'un objet C++ est de définir une fonction qui puisse enregistrer ses attributs dans un fichier.

Le problème de cette solution est de savoir comment l'on peut connaître en C++ le nom de la classe et de ses attributs pour pouvoir les réutiliser dans une chaîne de caractères. Ces chaînes de caractères étant ensuite utilisées pour la création du fichier de macro.

Exemple :

```
class Test
{
public :
   Test();
   ~Test();
   int nbTest;
};
```

Comment sauver l'instance de la classe `Test` qui a pour attribut `nbTest=5` ?

- Tout d'abord, il faut se demander comment on va pouvoir récupérer le nom de l'attribut `nbTest` de la classe `Test` pour pouvoir l'utiliser dans une chaîne de caractères (qui servira à l'écriture du fichier de macro).

  Une première piste de solution consisterait à utiliser un attribut qui contiendrait la concaténation des noms utiles pour l'enregistrement.

  Exemple :

```
   strAttr = "Test;nbTest;";
```

- Ensuite, il faut une méthode qui permette de renvoyer la valeur d'un attribut (ici `nbTest`) à partir d'une chaîne de caractères.

  Exemple :

```
std::string Test::reqAttribut(std::string str)
{
   std::string ret;
...
...
        return ret;
}
```

  Ainsi en appelant `reqAttribut("nbTest")`, on obtiendrait comme valeur de retour le nombre 5.

Cette solution n'a pas été retenue du fait qu'il faille modifier en conséquence chaque classe d'événement.

7.1.1.2 2nde Solution : le *Marshalling* Python

La deuxième solution étudiée consistait à vouloir utiliser la technique du *Marshalling* que Python propose à travers l'utilisation de son module « pickle ».

Le *Marshalling* consiste à enregistrer une instance d'une classe Python dans un fichier pour pouvoir ensuite recharger cette instance en Python.

Le *Marshalling* de Python semblait donc être assez intéressant, puisque seul l'exportation en Python des classes d'événements C++ était requis : aucun système de sauvegarde d'événement n'étant alors à inventer.

Cependant un problème est apparu : la technique du *Marshalling* sauvegarde l'instance d'un objet Python dans un fichier dont le contenu est illisible. Ce qui ne permet pas à un utilisateur de modifier directement la valeur d'un attribut d'un événement.

La solution du *Marshalling* de Python n'a donc pas été retenue.

7.1.1.3 3ème Solution : Enregistrement des événements en Boost Python

Puisque le *Marshalling* de Python ne fonctionnait pas comme on l'entendait, il a été décidé d'en reprendre les principes de bases pour créer une solution répondant au problème cité précédemment. Cette solution sera écrite en C++ à l'aide de Boost Python pour que les usagers de Modeleur ne puissent pas modifier des choses trop critiques.

Une première tentative d'implémentation de cette solution a été réalisée. Elle consiste à ne travailler en C++ qu'avec les événements de base `TEEvenement` pour pouvoir les passer à une classe C++ « Boost Python » qui se charge de sauvegarder chaque attribut de l'événement dans une chaîne de caractères qui sera lue en Python.
Voir la partie concernant l'amélioration de l'enregistrement des événements dans le paragraphe 9.2 de ce document (page 44).

Remarque :
L'une des questions qui est survenue dans la phase d'analyse de l'enregistrement des macros fut le fait de savoir comment faire connaître les événements des modules non implémentés ou des *plug-in* au système d'enregistrement des événements. La solution consiste à ne travailler en C++ qu'avec la classe de base des événements `TEEvenement`.

7.1.2 Evénements à ne pas sauvegarder

Certains événements peuvent ne pas avoir à être sauvegardés. Pour cela, deux pistes ont été évoquées :

- Marquer à l'aide d'un *flag* (indicateur) les événements à sauvegarder dans les macros,
- Inclure dans l'événement courant l'identifiant de l'événement source (celui qui a appelé).

Ex : Evénement A >> Evénement B
>> Evénement C

B et C ont pour source A, A n'a pas de source, A doit donc être sauvegardé. B et C ne doivent pas être sauvegardés car ils sont des sous événements de A.

C'est la première solution qui a été choisie du fait de sa simplicité. Un attribut `sauverEvenement` a donc été ajouté à la classe base `TEEvenement` qui permet de déterminer si les macros doivent enregistrer cet événement ou non.

Remarque :
La question suivante avait été posée lors d'une réunion concernant les macros : Comment gérer le fait que l'on puisse enregistrer une macro pendant que l'on en enregistre une autre (macro récursive) ? Ce problème se résout en rendant les événements de macros (commencer l'enregistrement, terminer l'enregistrement, etc...) non enregistrables, ainsi une macro ne peut être constituée d'autres macros.

7.2 Description du fichier macro généré

Pour mieux comprendre comment une macro est construite, on va analyser la structure retenue pour la génération du script Python.

7.2.1 Exemple de fichier macro complet

```
#-------------------------------------------------------------------------------
# macro_gen_auto.py
# macro générée automatiquement
#-------------------------------------------------------------------------------

from PY_Erreur import ERMsg
import PY_Script
from PY_TransmissionEvenement import TEEvenement

#-------------------------------------------------------------------------------

def Fct1()
    """ Evenement : TEEvenement """
    ev = PY_TransmissionEvenement.TEEvenement()
    ev.typeFin = None
    ev.source = None
    ev.destinataires = None
    ev.codeRetour = None
    return ev

#-------------------------------------------------------------------------------

def main():
    try:
        msg = PY_Script.envoieEvenement(Fct1())
    except:
        print "exception dans macro_gen_auto.py"
    return 0


if __name__ == '__main__':
    #-------------------------------------------------------------------------------
    # --- Program Entry Point
    main()
```

7.2.2 L'entête

La première partie du script consiste à simplement indiquer son nom :

```
#-------------------------------------------------------------------------------
# macro_gen_auto.py
# macro générée automatiquement
#-------------------------------------------------------------------------------
```

7.2.3 Le bloc des « imports »

La seconde partie du script consiste à lister tous les modules python qui doivent être importés :

```
from PY_Erreur import ERMsg
import PY_Script
from PY_TransmissionEvenement import TEEvenement
```

7.2.4 La définition des fonctions

La troisième partie consiste à lister toutes les fonctions :

```
def Fct1()
    """ Evenement : TEEvenement """
    ev = PY_TransmissionEvenement.TEEvenement()
    ev.typeFin = None
    ev.source = None
    ev.destinataires = None
    ev.codeRetour = None
    return ev
```

Corps de l'événement

Une fonction crée un événement et en modifie les attributs si nécessaires, ensuite elle retourne l'événement créé.

Remarque :
Il a été préféré d'expliciter l'initialisation des attributs pour une meilleure lecture du script Python. On peut voir dans le code qui suit, un exemple d'initialisation des attributs d'un événement qui est loin d'être compréhensible pour un usager externe.

```
def Fct1()
    """ Evenement : TEEvenement """
    ev = PY_TransmissionEvenement.TEEvenement(None, None, None, None)
    return ev
```

Cette solution n'a pas été retenue du fait qu'il est impossible à un usager externe de savoir facilement à quoi correspond chaque paramètre de l'événement.

7.2.5 L'envoi des événements

La quatrième partie consiste à envoyer les événements retournés par chaque fonction.

```
msg = PY_Script.envoieEvenement(Fct1())
msg = PY_Script.envoieEvenement(Fct2())
```

La fonction `PY_Script.envoieEvenement` prend en paramètre l'événement retourné par une fonction `FctX` et l'envoie sur le bus de Modeleur.

7.3 Ré-exécution d'une macro

7.3.1 Ré-exécution d'un script

Puisqu'une macro est un script, pour pouvoir ré-exécuter une macro, il suffit de pouvoir exécuter un script.

Pour pouvoir exécuter un script, il a été décidé de créer un événement qui prend en paramètre le nom de fichier du script. Cet événement est ensuite traité par le module de scripts qui demande à l'interpréteur Python d'exécuter le script donné en paramètre.

7.3.2 Ré-exécution des événements

Comme on vient de le voir, c'est la fonction `PY_Script.envoieEvenement` qui est chargée d'envoyer les événements sur le bus de Modeleur, mais comment cela fonctionne?

L'instance C++ du Module de scripts est partagée en Python, ainsi il devient possible d'appeler la méthode d'envoie d'événements à partir de Python.

7.3.3 Macro hors contexte

Il avait été souligné qu'une macro exécutée hors contexte pouvait être source de beaucoup d'erreurs.

Par exemple, si la macro exécutée a besoin d'une fenêtre de visualisation pour travailler et qu'aucune n'est ouverte lors de son exécution, il y a de fortes chances pour qu'il y ait quelques problèmes. La macro est donc hors de son contexte d'exécution.

Une des solutions proposées pour répondre à ce problème consistait à traiter l'erreur retournée lors d'un échec d'envoi d'événement.

```
try:
    msg = PY_Script.envoieEvenement(Fct1())
except:
```

Dans la version actuelle du module de scripts, aucune démarche dans ce sens n'a été effectuée pour le moment. Ce problème reste donc d'actualité.

8 Module de scripts - design et implémentation

8.1 Evénements du module de scripts

Le module de scripts définit les événements suivants :

- `SCEvenConfigureCommandesIG` : cet événement est utilisé par l'interface graphique pour mettre à jour l'activation (respectivement la désactivation) des entrées de menu du module de scripts.

- `SCEvenCommandeExecuteScript` : cet événement exécute un script dans un interpréteur Python. Cet événement possède un attribut `nomScript` qui désigne le nom du script à exécuter.

- `SCEvenCommandeEditeScript` : cet événement ouvre un script dans un éditeur de code. Cet événement possède un attribut `nomScript` qui désigne le nom du script à éditer.

  Remarque :
  Le traitement de cet événement ne fait rien pour l'instant.

- `SCEvenCommandeDebogueScript` : cet événement ouvre un script dans le débogueur. Cet événement possède un attribut `nomScript` qui désigne le nom du script à déboguer.

  Remarque :
  Le traitement de cet événement ne fait rien pour l'instant.

- `SCEvenCommandeMacro` : cet événement, suivant la valeur de son attribut `type`, sert à :
  - débuter l'enregistrement d'une macro (`type == MACRO_DEBUT`) ;
  - terminer l'enregistrement d'une macro (`type == MACRO_FIN`) ;
  - mettre en pause l'enregistrement d'une macro (`type == MACRO_PAUSE`).

  Cet événement possède un attribut `nomScript` qui désigne le nom de fichier de la macro qui va être enregistrée.

  Remarque :
  L'événement `SCEvenCommandeMacro` n'est pas enregistré dans les macros puisque `asgSauverEvenement(FAUX)` est appelé dans le constructeur de `SCEvenCommandeMacro`.

8.2 Composition du module de scripts

Le module de scripts est constitué des classes suivantes :

`SCModule` : classe du module de scripts.
- `SCEvenXXX` : classes qui définissent les événements du module de scripts.
- `SCMacro` : classe qui définit la macro en cours d'enregistrement.
  - `SCImports` : classe qui définit les importations nécessaires à la macro.
  - `SCTraducteurPython` : classe qui permet de traduire un objet Python en un objet `SCMacroEvenement`.
    - `SCCorpsMacro` : classe qui crée le corps de l'événement à sauvegarder.
  - `SCMacroEvenement` : classe qui définit un des événements à sauvegarder dans le fichier de la macro :

Remarque :
L'identifiant des classes du module de scripts est : « SC ».

Voir Annexe 4 : Diagramme des classes (UML)
(paragraphe 13.4, page 68)

8.3 Fonctionnement de l'enregistrement des macros

Pour bien comprendre l'objectif que l'on veut atteindre, il est préférable de revoir le fichier que l'on souhaite générer : paragraphe 7.2.1, page 34.

1) Lors de la réception par le module de scripts (`SCModule`) d'un événement de début de macro, la méthode `traiteEvenCommandeMacroDebut` est appelée de `SCModule` ; ce qui crée une macro vide.

2) A chaque réception d'un événement (à sauvegarder), le module de scripts l'enregistre en appelant la méthode `ajouteEvenement` de la classe `SCMacro`.

- Cette méthode `ajouteEvenement` crée un nouvel `SCMacroEvenement`.
- Elle appelle ensuite la méthode `appelleSCTraducteurPython` pour remplir ce `SCMacroEvenement` à partir de l'événement à sauvegarder.
- La méthode `appelleSCTraducteurPython` appelle des méthodes de la classe SCTraducteurPython qui vont remplir certains attributs de `SCMacroEvenement`.
- La méthode `ajouteEvenement` complète le remplissage des attributs de `SCMacroEvenement`.
- Une fois `SCMacroEvenement` rempli, il est empilé à la liste des `SCMacroEvenement` de la macro.

- Chaque événement à sauvegarder est défini dans un module Python. Le nom de ce module Python est donc sauvegardé dans un objet (`SCImports`).
- Après avoir créé l'objet `SCImports`, la méthode `ajouteEvenement` l'empile à la liste des `SCImports` de la macro.

3) Lors de la réception par le module de scripts (`SCModule`) d'un événement de fin de macro, la méthode `traiteEvenCommandeMacroFin` est appelée de `SCModule` ; ce qui crée le fichier Python correspondant à la macro. Ce fichier est écrit à l'aide des opérateurs << des classes (`SCMacro` et `SCMacroEvenement`) et du contenu des listes `SCMacroEvenement` et `SCImports`.

Voir Annexe 5 : Diagramme de séquence (UML)
(paragraphe 13.5, page 70)

8.4 Classe SCModule

La classe `SCModule` constitue le module de scripts, elle dérive de la classe `TEModuleBase`, elle possède donc les méthodes `envoieEvenement` et `recoisEvenement` qui permettent au module de scripts de se connecter au bus.

La classe SCModule est constituée des attributs suivants :

- L'attribut `interpreteurP` est un interpréteur Python. Cet interpréteur est défini dans le projet Visual `boost::sup`. Il s'agit d'un pointeur qui ne doit jamais être nul (invariant de la classe). L'utilisation du pointeur a été préférée à la déclaration normale car elle permet de ne pas lier `SCModule.h` avec `boost::python.hpp`.

- L'attribut `macroP` est un pointeur sur la macro en cours d'enregistrement (`SCMacro`). Si ce pointeur vaut nul, c'est qu'il n'y a pas de macro en cours d'enregistrement.

Les méthodes importantes de la classe `SCModule` sont :

- La méthode `enregistreEvenement` de `SCModule` qui est appelée par `recoisEvenVirtuel` à chaque réception d'un événement. Cette méthode permet d'enregistrer le dernier événement dans la macro en cours d'enregistrement. Pour cela, elle appelle la méthode `ajouteEvenement` de l'attribut `macroP`.

- La méthode `effaceDernierEvenement` n'est pas utilisée pour le moment. Elle a été créée dans le but de permettre d'effacer le dernier événement sauvegardé dans une macro en cours d'enregistrement. Cette méthode sera probablement utilisée lors d'une demande d'annulation d'une action.

8.5 SCMacro

La classe SCMacro représente la macro qui est en cours d'enregistrement, cette macro peut être en pause (`enPause`) ou non. Elle comporte la liste des fichiers à importer en Python (liste de `SCImport`), la liste des événements à sauvegarder (liste de `SCMacroEvenements`), et le numéro du prochain événement à sauvegarder.

Si la macro n'est pas en pause, la méthode `ajouteEvenement` de `SCMacro` est appelée. Cette méthode enregistre le dernier événement qui circulait sur le bus dans la macro (si bien sûr l'événement doit être sauvegardé) sous forme d'un `SCMacroEvenement`.

Le `SCMacroEvenement` est rempli à l'aide d'un traducteur `SCTraducteurPython`. Après cet appel, le `SCMacroEvenement` contient les imports, le nom de l'événement et le corps de l'événement. Il ne manque que le nom de la fonction qui va contenir le corps de l'événement. Ce nom est créé à partir de l'attribut `noProchainEvenement` (qui est ensuite incrémenté).

A la fin de l'enregistrement de la macro (`SCMacro`), un fichier de macro est créé avec les informations contenues dans `SCMacro` (cet enregistrement se fait à l'aide de l'opérateur<<).

8.6 SCMacroEvenement

La classe `SCMacroEvenement` est constituée de quatre attributs :

- L'attribut `nomEvenement` représente le nom de l'événement
- L'attribut `nomFonction` représente le nom de la fonction
- L'attribut `fichierImport` représente le nom du module Python à importer pour créer l'événement
- L'attribut `chainesAEcrire` représente le corps de l'événement à écrire.

Cette classe sert à sauvegarder toutes les informations nécessaires à l'enregistrement d'un événement.
Elle est utilisée par la classe de macro (`SCMacro`) qui conserve une liste des événements à enregistrer sous un format spécifique, ce format étant `SCMacroEvenement`.

Quand la classe `SCMacro` crée le fichier de macro (par appel de son opérateur<<), l'opérateur<< de `SCMacroEvenement` est appelé.

8.7 SCimports

La classe `SCImports` comporte la liste des événements à importer en Python. Chaque événement est inclus dans un seul fichier. Il est donc inutile d'importer deux fois en Python un même événement d'un même fichier. Pour éviter cela, le couple d'attributs Evénement/Fichier est sauvegardé dans un `std::set`.
Pour pouvoir trier le couple d'attributs (et ne pas avoir de doublon), un test de comparaison (opérateur<) a été créé. Cet opérateur fonctionne à l'aide de la concaténation des deux attributs de cette classe.

8.8 SCTraducteurPython

Cette classe représente un package d'outils qui permettent de traduire le nom d'objets Python en string C++. Ces chaînes de caractères étant ensuite utilisées pour remplir les attributs (`nomEvenement`, `nomFonction` et `fichierImport`) de `SCMacroEvenement`.

8.9 SCCorpsMacro

Cette classe est utilisée par `SCTraducteurPython` et sert à remplir le corps de l'événement `SCMacroEvenement` (`chainesAEcrire`).

8.10 Interactions avec Python

8.10.1 Initialisation du module de scripts

Lors de l'appel de la méthode de traitement de l'événement d'initialisation `traiteEvenInitialise`, on importe `PY_Module_Script_Instance.py` dans l'interpréteur Python utilisé par le module de scripts.

Code C++ :

```
ERMsg SCModule::traiteEvenInitialise (TEEvenement & even)
{
...
      python::object mdul = interpreteurP->import_module("PY_Module_Script_Instance");

      python::object func = mdul.attr("__dict__")["initInstanceModuleScript"];
      python::object oret = func(boost::ref(*this));
      Entier ret = python::extract<Entier>(oret);
...
}
```

Ensuite, on appelle la fonction Python `initInstanceModuleScript` qui initialise un singleton avec l'instance du module C++.

Code Python :

```
def initInstanceModuleScript(instanceModuleScript):
    s = Singleton("moduleScript", instanceModuleScript)
    return 0
```

Pour récupérer l'instance du module C++ de scripts (le `this` de `SCModule`) en Python, il suffit de faire :

```
    instModuleScript = reqSingletonInstance("moduleScript")
```

8.10.2 Classes exportées vers Python

Seuls les événements du module de scripts et une partie de `SCModule` sont exportés vers Python à l'aide de Boost Python. La partie de `SCModule` exportée vers Python est la méthode C++ `envoieEvenement` qui permet d'envoyer un événement sur le bus.

9 Améliorations à apporter

9.1 Gestion des erreurs

Une meilleure gestion des erreurs doit être implémentée. Ceci touche les aspects suivants :

- les scripts du Module de scripts ;
- la compilation de scripts Python avant leur exécution ;
- l'interpréteur Python du projet Visual C++ « boost_sup ».

Remarque :
La gestion des erreurs de l'interpréteur Python pourrait être dans un premier temps de créer un fichier de log contenant toutes les erreurs retournées par Python.

9.2 Enregistrement des événements

L'enregistrement des événements pose les problèmes suivants :

- *on ne connaît pas le nom de l'événement que l'on enregistre. Or ce nom est nécessaire pour l'écriture du fichier de macro.*

L'enregistrement des événements se fait en partie en C++. Du côté C++, on voulait utiliser l'événement de base `TEEvenement` pour le traiter ensuite en Python afin d'en extraire les données.

Or, en passant un événement de type `TEEvenement` à Boost Python, Boost Python n'arrive pas à déterminer le type réel de l'événement. Il ne voit que le type de base et l'on n'arrive pas à obtenir le nom réel de l'événement en Python.

Une solution non acceptée consista a inclure dans chaque événement une méthode qui retourne un `boost::python::object` basé sur `this`, ainsi Boost Python arrive à déterminer le type réel de l'événement et on peut trouver le nom de cet événement en Python.

Exemple :

```
class SCEvenementTest : public TEEvenement
{
public:
   SCEvenementTest(TERecepteurs r = TOUS, TypeFin t = ATTEND_FIN_TRAITEMENT)
      : TEEvenement (r, t) {}
   virtual ~ SCEvenementTest() {}

   boost::python::object reqBoostPythonObject() {return boost::python::object(*this); }

};
```

La solution n'a pas été acceptée du fait qu'elle oblige l'inclusion (très lourde) de Boost Python dans toutes les classes d'événements (et donc dans la plupart des classes).

- *l'enregistrement des attributs « property » de Python ne fonctionne pas.*

9.3 Liste des améliorations à effectuer

La liste à jour des améliorations et corrections à apporter au module de scripts se trouve sur le serveur de *bug tracking* :
http://www.bugtracking.gre-chn.inrs-ete.uquebec.ca:10000/

9.4 Gestion des scripts (BD)

Il existe dans Modeleur deux types de scripts Python :
- les scripts locaux ;
- les scripts globaux (contenus dans la base de données de Modeleur).

9.4.1 Scripts globaux

La gestion des scripts globaux n'a pas été implémentée, seules les spécifications fonctionnelles ont été abordées sur ce sujet (voir paragraphe 6.2 - Gestion des scripts globaux, page 26).

9.4.2 Scripts locaux

La tâche de gérer les scripts locaux a été laissée au système de fichiers de Windows.

Ceci pose un problème avec les spécifications fonctionnelles de Modeleur, car il y est indiqué qu'il ne devrait y avoir qu'une seule façon de gérer les données : à l'aide d'une base de données locale.

L'inconvénient de cette solution est que la gestion des scripts locaux (fichiers) et globaux (BD) n'est pas la même, ce qui n'est pas souhaitable.

Le fait que les scripts locaux ne soient pas dans une base de données ne permettra pas
- de pouvoir faire des requêtes pour sélectionner un script local;
- de conserver un historique des versions des scripts locaux comme il est possible de le faire avec des scripts globaux.

Cependant, la gestion des scripts Python dans les éditeurs et débogueurs se faisant à l'aide de fichiers, il aurait fallu modifier le code source de l'interpréteur Python pour

qu'il puisse fonctionner avec une base de données locale, ce qui n'est pas acceptable puisqu'en modifiant le code source de Python, on perd l'intérêt d'utiliser un outil externe (maintenance, etc...).

Les principales difficultés qui s'opposent à l'utilisation d'une base de données locale à la place d'un système de fichiers local en Python sont l'importation des fichiers Python dans d'autres fichiers Python.

En effet, la commande `import` en Python importe un module Python dans un autre module Python. Cependant ces importations se font à l'aide du système de fichiers de Windows. Si l'on souhaite passer par un autre mécanisme, il faut impérativement modifier le code source de Python.

Remarque :
On retrouve le même problème dans l'importation de répertoire contenant des modules Python (`__init__.py`).

9.4.3 Versions des scripts

La gestion globale et locale des scripts entraîne diverses versions de scripts qui ne sont pas forcément les mêmes.

Il faudrait donc pouvoir donner à l'usager des fonctionnalités qui aideraient à conserver des liens de versions entre un script global et un script local. La gestion des versions et des scripts reste l'un des points majeurs à développer dans le module de scripts.

9.5 IDE : Editeur / Débogueur

La partie Editeur/Débogueur du module de scripts n'a pas encore été implémentée. Elle consistera à implanter un IDE externe dans Modeleur pour pouvoir y éditer (respectivement y déboguer) des scripts.

Pour le moment, seul un IDE répond à toutes les caractéristiques requises, il s'agit de Boa Constructor.

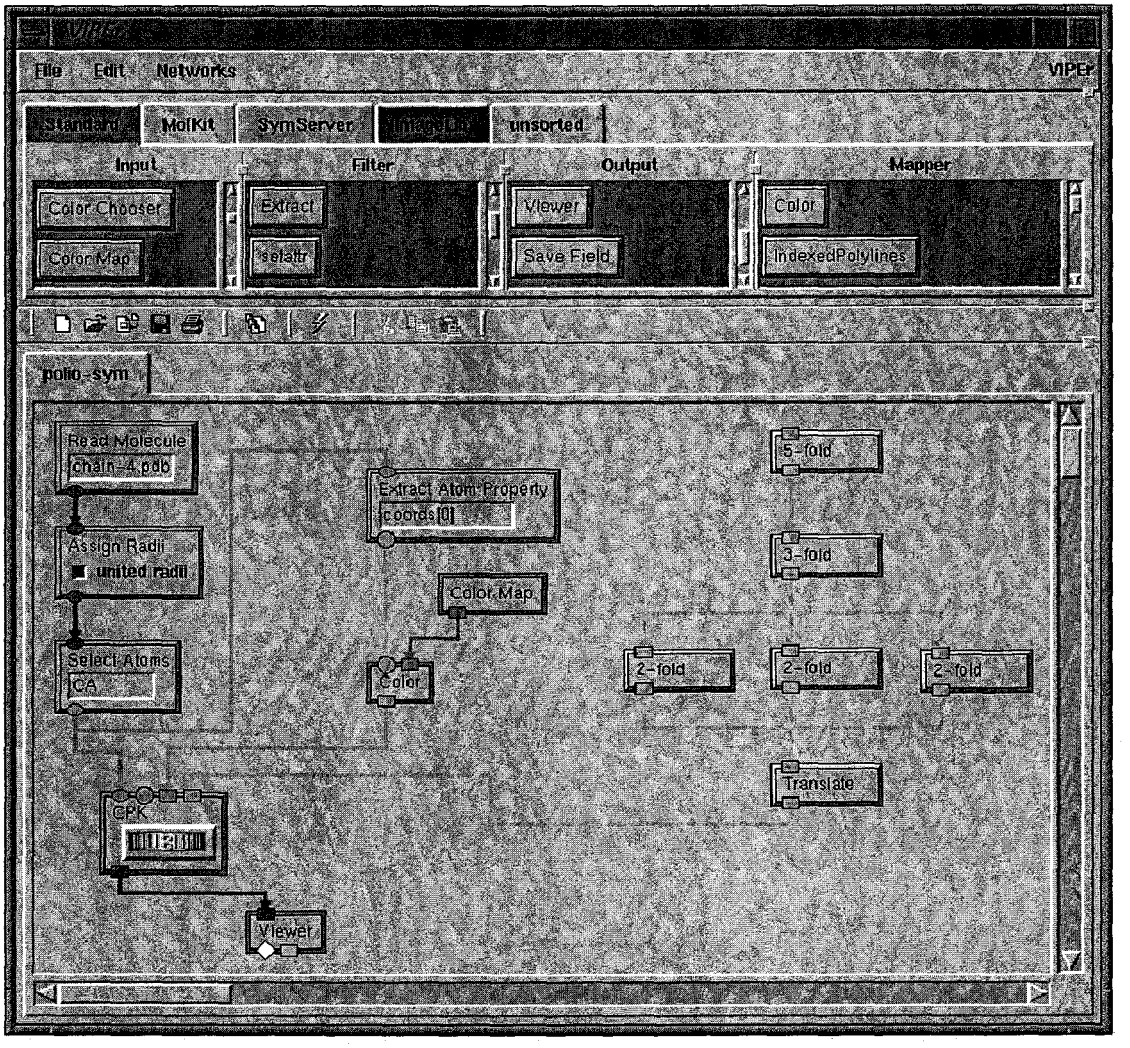
Remarques :
L'utilisation d'une base de données pour la gestion locale des scripts poserait quelques problèmes car les éditeurs de code travaillent tous avec des fichiers. Il faudrait donc pouvoir « passer » à ces éditeurs un fichier temporaire basé sur le script contenu dans la base de données locale.

De plus la fonction « enregistrer sous » des éditeurs a pour effet d'enregistrer un fichier dans le système de fichiers de Windows, ce qui n'est pas ce que l'on souhaite puisque l'on veut récupérer les modifications apportées au script dans la base de données locale. La seule solution pour qu'un éditeur fasse ce dont on attend de lui serait de modifier son code source, ce qui n'est pas souhaitable.

L'édition et surtout le déboguage de scripts à partir de Modeleur sont deux points importants du module de scripts qu'il reste à développer.

9.6 Programmation visuelle

La programmation visuelle en Python n'a pas été étudiée dans le cadre de ce dossier, cependant un outil intéressant a été trouvé sur Internet et il peut servir de base à une intégration future de la programmation visuelle dans Modeleur. Il s'agit de : « ***ViPEr**: a visual programming environment for Python* ».



Plus d'informations disponibles à cette adresse :
http://www.scripps.edu/~stoffler/proj/ViPEr/viper.html

10 Conclusion

10.1 Résumé

La mise en place du langage interprété dans Modeleur a débuté par le choix du langage. C'est une solution basée sur le langage Python qui a été choisie. Les composants de cette solutions sont les suivants :

- Python comme langage interprété ;
- Boost Python pour l'interopérabilité entre Python et le C++ ;
- wxPython pour l'écriture d'interface graphique en Python.

Après le choix du langage, certaines répercussions de ce choix sur le développement de certaines parties de Modeleur ont été analysées : bus d'événements, interface graphique, *plug-in*, calculatrice, scripts et macros.

Enfin, la partie « scripts / macros » de Modeleur a été abordée en détails à travers l'analyse et le développement effectués sur le module de scripts. Seule la partie relative aux macros a été développée dans le module de scripts

10.2 Perspectives futures

Concernant le module de scripts, il reste trois points à développer :

- la correction des bogues restant dans la partie « macro » du module de scripts,
- l'intégration d'un éditeur / débogueur de scripts dans Modeleur,
- la gestion des scripts globaux à l'aide de la base de données globale de Modeleur.

Concernant l'implantation du langage interprété, il reste deux points à développer :

- la calculatrice de Modeleur ;
- la gestion des erreurs renvoyée par l'interpréteur Python utilisé dans Modeleur.

11 Glossaire

ActiveX : code compatible Windows qui permet de réutiliser des éléments Windows. exemple : le navigateur Web de IE, la page de Acrobat Reader, etc...

API : Application Programming Interface : Interface de programmation d'applications, contenant un ensemble de fonctions courantes de bas niveau, bien documentées, permettant de programmer des applications de « Haut Niveau ». On obtient ainsi des bibliothèques de routines, stockées par exemple dans des DLL.

Assembly : conteneur physique des classes lors de la compilation d'un projet .NET (VB.NET, C#, MC++, ...). C'est souvent une DLL...

C++ : Le C++ est le langage de programmation (compilé) utilisé dans l'écriture du code source de Modeleur. C'est un langage Orienté Objet, multi-paradigme, très puissant et complexe.

CLR : le Common Language Runtime est l'équivalent de la machine virtuelle de java appliqué à .NET.

COM (Component Object Model) : Standard de gestion d'objets distribués, propre à Microsoft. (équivalent CORBA sous Microsoft).

DLL : Dynamic Linked Library. bibliothèque de liens dynamiques. Ensemble de routines extraite d'un programme principal, pour être partagées par plusieurs programmes, ou pour optimiser l'occupation de la mémoire (les DLL pouvant être chargées et déchargées à volonté). Utilisées essentiellement sous Windows désormais, autrefois utilisées par OS/2, elles peuvent aussi être exploitées dans certaines conditions sous Linux.

Langage compilé : Langage dans lequel le code d'un programme est traduit en langage binaire directement compréhensible par le processeur. Les programmes sont ainsi assez rapides, en tout cas généralement bien plus rapides que dans le cas des langage interprété (encore faut-il que le compilateur soit de qualité).

Langage interprété / langage de scripts : langage non compilé constitué d'une suite d'instructions interprétées qui sont exécutées de façon séquentielle par un programme intermédiaire appelé interpréteur de commandes. Bien que l'interprétation réduit l'efficacité de l'exécution, les langages de script sont souvent plus simples à apprendre que les langages compilés tout en fournissant de puissantes fonctionnalités.

Macro (Macro-commande) : commande formée par une succession d'autres commandes répétitives. Dans Modeleur, une macro est une suite d'événements enregistré dans un script Python.

Managed C++ (MC++ ou C++ Managé) : « Amélioration » du langage C++ par Microsoft qui comporte de nouveaux mots clés : __gc par exemple pour créer des classes avec garbage collector (ramasse-miettes) et qui est compatible avec le framework .NET.

MSIL (Microsoft Intermediate Language) : code intermédiaire généré par un compilateur .NET pour qu'il puisse être exécuté par le CLR.

Module C++ : Partie C++ d'un module (ou plug-in) Modeleur.

Module Python : un module Python est un élément qui fournit de nouvelles fonctions à un interpréteur Python. Dans Modeleur, scripts, macros et dll Python (.pyd) sont des modules Python.

MFC : Microsoft Foundation Classes : bibliothèque de classes d'objets destinées à construire des applications sous Windows de Microsoft.

OCX : Extension de Visual Basic à partir de sa version 4.

Open source : Définition particulière du logiciel libre, mise au point en 1998 par Eric Raymond, cherchant à adapter le principe à l'entreprise. Elle comporte neuf points (pour le moment) : la libre redistribution, la mise à disposition du code source, la possibilité de distribuer ses travaux dérivés, le respect du code source originel, l'absence de discrimination envers des personnes, l'absence de limitation sur le domaine d'application du logiciel, la distribution de la licence et sa non spécificité à un produit, et enfin le fait qu'elle ne contamine pas le travail des autres.

Philosophie de licence : Choix concernant la licence du logiciel. Il existe plusieurs grandes catégories de licence pour un logiciel : les logiciels libres (généralement associés aux logiciels open source), les logiciels gratuits, les logiciels payants.

Python : langage de script, libre, cousin de Perl dans son esprit, mais beaucoup plus fortement orienté objet, dont le développement a débuté en 1990. Sa syntaxe est décrite comme « élégante » par ses auteurs. La version 2.3 est sortie en 2003.

Plug-in (Plugiciel) : extension à une application qui vient sè loger dans l'application elle-même. Une fois installé, on peut utiliser le plug-in de façon tout à fait transparente.

RAD : *Rapid Application Developpement.* développement rapide d'application avec des outils modernes.

Script: Suite d'instructions simples, peu structurées, permettant d'automatiser certaines tâches en se passant d'un réel langage

Toolkit : ensemble d'outils réutilisables.

Sources de certaines définitions :http://www.linux-france.org/prj/jargonf/index.html

12 Bibliographie

Sites sur Python :

http://www.Python.org/

http://www.Python.org/doc/NonEnglish.html#french

http://Python.ilisys.com.au/doc/faq/Windows.html

http://lfe.developpez.com/tutoriel/Python/

http://www.ulg.ac.be/cifen/inforef/swi/Python.htm

http://wikiPython.flibuste.net/moin.py/FrontPage

http://www.Python-eggs.org/links.html

Sites sur wxPython :

http://www-106.ibm.com/developerworks/library/l-wxpy/index.html

http://www.wxPython.org/tutorial.php

http://wiki.wxPython.org/index.cgi/FrontPage

Sites sur Boost Python :

http://www.Python.org/cgi-bin/moinmoin/Boost_2ePython

http://www.Boost.org/libs/Python/doc/

http://mail.Python.org/pipermail/C++-sig/

http://www.Boost.org/libs/Python/pyste/

Outils Python intéressants :

http://boa-constructor.sourceforge.net/

http://starship.Python.net/crew/mhammond/

http://www.Python.org/Windows/Pythonwin/

http://spe.pycs.net/

http://hapdebugger.sourceforge.net/

13 Annexes

13.1 Annexe 1 : VB.NET vs Python

13.1.1 Présentation de la solution Python

Python seul ne serait pas d'un grand intérêt pour Modeleur 2.0, car Python (seul) n'est qu'un langage de scripts, il ne pourrait donc créer d'interfaces graphiques et s'interfacer avec du code C++. Cependant, à l'aide de certains outils, il devient un outil très performant :

- Boost Python : qui permet d'accéder à du code Python dans du code C++ et inversement.
- Pyste : qui permet de générer du code Boost Python à partir de C++.
- Extensions Win32 pour Python qui permettent d'augmenter les capacités de Python (par exemple, elles permettent d'utiliser un ActiveX sous Python).
- wxPython qui permet de créer une interface graphique à l'aide de code Python.

13.1.2 Présentation de la solution VB.NET

VB.NET, lui, n'a pas besoin d'autres ajouts pour fonctionner puisque VB.NET est une fusion de VB (interface graphique), de VBA (macros) et de VBscript.

Cependant, l'utilisation du logiciel « Visual Studio for Applications » (VSA) pourrait permettre d'améliorer très significativement le processus de *plug-in* et de modification de l'interface graphique de Modeleur. Toutefois, Microsoft n'a pas publié encore une version officielle de ce produit.

13.1.3 Comparaison VB.NET – Python

La comparaison s'est faite sur les critères suivants :

- configuration requise
- flexibilité du langage
- facilité du Déboguage de la solution
- facilité du Déboguage des scripts
- documentation
- portabilité
- interopérabilité avec le C++
- propriétés du langage
- interface graphique

Pour chaque critère, une solution a été désignée comme étant la plus avantageuse.

13.1.3.1 Configuration requise

	VB.NET	Python
Compilateur	Visual Studio .NET ou Builder 6.0	Compilateur C++ Installation de Python Installation de Boost Python Installation de Pyste Installation de wxPython
Coût :	Suivant la licence	Gratuit
Ajout possible	**VSA**	Visual Python (/!\ : pas visuel), wxDesigner, etc…
Coût :	Suivant la licence	Suivant la licence

Python qui est gratuit ne nécessite pas de compilateur compatible .NET, cependant les licences de **Visual Studio .NET** ayant déjà été achetées, cet argument n'est à prendre en compte que pour les utilisateurs « programmeurs ».

AVANTAGE	PYTHON

13.1.3.2 Flexibilité du langage

La solution basée sur Python étant libre, il est toujours possible de modifier les sources pour ajouter une fonctionnalité dont on n'aurait pas vu l'absence durant la phase de recherche du langage de scripts.

La solution VB.NET ne permet pas d'accéder aux sources du moteur de scripts, il n'est donc pas possible de le modifier.

AVANTAGE	PYTHON

13.1.3.3 Facilité du Déboguage de la solution

Avec Python, le « déboguage » d'un programme se fait avec l'accès aux sources (de Python, de Boost Python et de wxPython), ce qui est très appréciable.

	VB.NET	Python
Accès aux sources (très utile pour le déboguage)	Non	Oui

AVANTAGE	PYTHON

13.1.3.4 Facilité du Déboguage des scripts

	VB.NET	Python
Outils matures pour le déboguage de scripts	Oui	Non

AVANTAGE	VB.NET

13.1.3.5 Documentation

L'interpréteur de scripts VB.NET du framework .NET est très peu documenté : il n'y a même pas de fichier entête classique à disposition.

On peut également noter que très peu de documentation existe sur l'utilisation d'un interpréteur de scripts sous .NET et que les exemples de code sont plutôt rares.

	VB.NET	Python
Documentation sur la solution	VB.NET : suffisante VSA : seules les fonctions du framework sont expliquées de manière séparée. Il y a très peu d'articles sur la façon d'utiliser ces classes et très peu d'exemples.	Python : suffisante Boost Python : suffisante wxPython : à définir (mais beaucoup d'exemples)

AVANTAGE	PYTHON

13.1.3.6 Portabilité

	VB.NET	Python
Libre	Non	Oui
Code portable	Non	Oui
Compatible Linux	Non	Oui
Compatible MacOS	Non	Non (Boost Python étant actuellement incompatible avec cet OS)
Compatible UNIX	Non	Oui
Compatible OS/2	Non	Oui

AVANTAGE	PYTHON

13.1.3.7 Interopérabilité avec le C++

	VB.NET	Python
Accès à une classe C++ dans le langage de scripts	Oui dans le framework .NET (non testé) / ou dans un script	Oui à l'aide de Boost Python (création d'une dll)
Accès à une fonction C++ dans le langage de scripts	?	Oui à l'aide de Boost Python (création d'une dll)
Accès à une classe du langage de scripts dans C++	A première vue, cela paraît plutôt difficile puisque VB.NET doit toujours être compilé dans une Assembly* avant exécution. Ceci est donc possible seulement si le code VB.NET est exporté dans une dll.	Oui à l'aide de Boost Python (en « **live** » : pas de dll nécessaire)
Accès à une fonction du langage de scripts dans C++	?	Oui à l'aide de Boost Python (en « **live** » : pas de dll nécessaire)
Mise en œuvre de l'interopérabilité C++ / Langage	Compliqué	Simple
Implémentation d'un interpréteur	2 pages de code MC++	3 lignes de code C++!
Partage du même espace mémoire	Oui (instanciation du côté C++, inverse non testé)	Oui C++>>Python>>C++>>Python
Implémentation C++ natif / langage	Compliqué : mettre le code .NET dans un objet COM et l'appeler dans du code natif.	Simple : Boost Python

*Assembly : voir glossaire

Pour pouvoir accéder à du code C++ dans du code Python, il faut générer une DLL à l'aide de Boost Python. On inclut ensuite cette DLL dans Python.

Pour pouvoir accéder à du code Python dans du code C++, on appelle dans le code C++, les méthodes qui vont être définies dans le code Python à l'aide de fonctions Boost Python.

Pour pouvoir accéder à du code C++ dans du code VB.NET, on peut exporter une classe pour qu'elle soit accessible dans VB.NET ou on peut générer une DLL avec le code C++ (Managé) pour ensuite l'inclure dans du code VB.NET.

AVANTAGE	PYTHON

13.1.3.8 Propriétés du langage

	VB.NET	Python
Héritage multiple	Non (sauf pour les interfaces)	Oui
Surcharge des méthodes	A vérifier	Avec Boost Python
Surcharge des opérateurs	A vérifier	Natif dans Python
Typage	Statique	Dynamique
Base de données	ADO.NET, OLEDB, ODBC, ODBC.NET…	ODBC, Oracle, Informix, mSQL, mySQL
Extension possible du langage	Non	Oui
Nécessite le framework .NET	Oui	Non
Type de scripts	Compilés	Interprétés (en « live »)
Facile à apprendre pour les non-programmeurs	Oui	Oui
Nombre de Lignes de code pour un projet type	--	Beaucoup moins que les autres langages
Vitesse d'exécution	Plus lente (moteur VSA du framework .NET) que Python	Moins rapide que le C++, mais peut être amélioré pour s'en approcher.

NB : les programmes créés en .NET nécessitent d'être exécutés à partir de c:\ pour des raisons de sécurité…

AVANTAGE	PYTHON

13.1.3.9 Interface graphique

	VB.NET	Python
GUI	VB.NET (produit récent qui n'a pas encore toutes les fonctionnalités souhaitées : en particulier VSA n'est pas encore disponible sur le marché)	Librairie wxPython basée sur wxWindows (qui a déjà 10 ans, donc est un produit mature)
L'interface peut comporter des ActiveX, COM	Oui	Oui
L'interface peut comporter des barres d'outils flottantes	Oui	Oui pour wxWindows, mais pas testé sous wxPython
Boite de dialogue Windows	Natives	Natives
Possibilité de créer des interfaces Windows très complexes	Oui	Risque de tomber sur un élément que la librairie wxPython ne prend pas en charge.
RAD et IDE matures	Oui	Non
Logiciels pour la conception de boites de dialogue	Visual Basic .Net (mature) Builder 6.0 (mature)	wxDesigner : mature, mais que pour les boites de dialogues) Boa Constructor et PythonCard : produits non matures Visual Python (non visuel) pour Visual Studio .NET (éditeur / débogueur)

AVANTAGE	VB.NET

13.1.4 Résumé

En résumé, la solution Python possède tous les points requis (portabilité en plus), son seul défaut étant peut être son manque (présumée) de robustesse. On ne peut, en effet, s'assurer de l'absence d'une fonctionnalité ou d'un bug interne à Python et à ses logiciels associés (Boa Constructor, etc...).

La solution VB.NET a, elle, un côté plus sûre, mais la difficulté de programmation s'en trouve augmentée puisqu'il est nécessaire d'inclure toute la logique .NET dans le développement logiciel. Un dernier point à soulever est le fait que VB.NET possède un RAD bien plus évolué (mais payant) que celui disponible sous Python (Boa Constructor).

13.2 Annexe 2 : Utilisation de Pyste

13.2.1 Avant-propos

Pour une meilleure compréhension de cette annexe (qui est assez technique), il est recommandé de lire le tutorial de Pyste. Ce tutorial peut se trouver à l'adresse suivante : http://www.Boost.org/libs/Python/pyste/

13.2.2 Création du projet

Pour exporter des classes C++ d'un Projet Visual C++ (se nommant `MonProjet`), il faut dans un premier temps créer un nouveau projet Visual C++(dont le nom sera `PY_MonProjet`).

Ensuite il faut associer à ce projet deux fichiers : `PY_MonProjet.cpp` et `PY_MonProjet.h`.

Fichier `PY_MonProjet.h` :

```
#ifndef PY_MONPROJET_H_DEJA_INCLU
#define PY_MONPROJET_H_DEJA_INCLU

#endif  // PY_MONPROJET_H_DEJA_INCLU
```

Fichier `PY_MonProjet.cpp` :

```
// Includes ====================================================================
#include <boost/python.hpp>

// Using =======================================================================
using namespace boost::python;

// Declarations ================================================================

// Module ======================================================================
BOOST_PYTHON_MODULE(PY_MonProjet)
{

}
```

Il faut ensuite créer un fichier `exec_PY_MonProjet.cmd` vide.

Remarques:
- Il faut vérifier que `boost_python_debug.lib` fait bien partie des dépendances du projet Visual C++.

- En référence aux normes de programmation, l'emplacement des fichiers doit suivre l'exemple suivant :
`MonProjet.vcprj` dans `/MonProjet/prjVisual`.
`PY_MonProjet.vcprj` dans `/MonProjet/prjVisual`.
`PY_MonProjet.h` dans `/MonProjet/include`.
Les fichiers `.cpp` dans `/MonProjet/source`.
Les fichiers `.pyste` dans `/MonProjet/source`.
Les fichiers `.cmd` dans `/MonProjet/source`.

13.2.3 Génération automatique des fichiers Boost Python

Soit la classe C++ suivante à exporter :

```
template <typename TTDonnee, typename TTTraits>
class SRSerieSimple : public SRSerie<TTDonnee, TTTraits>
{
…
   const   TTRegion&     reqRegion          () const {return champ.reqRegion();};

…
};
```

Pour générer le fichier C++ Boost Python correspondant à l'exportation de cette classe en Python, il faut écrire le fichier Pyste suivant :

Fichier `PY_SRSerieSimpleSRChampAnalytiqueDoubleEFTraits.pyste` :

```
SRSerieSimple = Template("SRSerieSimple", "PY_MonProjet.h")
set_policy(SRSerieSimple.reqRegion, return_value_policy<copy_const_reference>())
SRSerieSimple("SRChampAnalytique<double> EFTraits",
↳                                        "SRSerieSimpleSRChampAnalytiqueDoubleEFTraits")
```

La première ligne du code Pyste indique à Pyste que la classe que l'on souhaite exporter en Boost Python est une classe Template.

La seconde ligne indique à Pyste que la méthode `reqRegion` de la classe `SRSerieSimple` retourne une référence constante C++.

La dernière ligne indique à Pyste les paramètres C++ de la classe Template et le nom de cette classe en Python. Ici la `TTDonnee` est un champ analytique de double : `SRChampAnalytique<double>`.

Remarque :
Le fichier `PY_MonProjet.h` doit contenir tous les entêtes nécessaires à la génération du fichier C++ Boost Python.

```
#ifndef PY_MONPROJET_H_DEJA_INCLU
#define PY_MONPROJET_H_DEJA_INCLU
#include "SRSerieSimple.h"
#include "SRChampAnalytique.h"
#include "SRChamp.h"
#include "EFTraits.h"
#endif  // PY_MONPROJET_H_DEJA_INCLU
```

Ensuite, il faut créer un fichier de commande Windows qui permette de générer le fichier C++ Boost Python à l'aide de Pyste :

Fichier `exec_PY_SRSerieSimpleSRChampAnalytiqueDoubleEFTraits.cmd` :

```
set INCLUDE=
set PYTHONPATH=C:\Program Files\boost-1.30.2\libs\python\pyste\src
set PATH=c:\PROGRA~1\GCC_XML;%PATH%

set PYSTE_MODULE=PY_SRSerieSimpleSRChampAnalytiqueDoubleEFTraits
set PYSTE_INCLUDE=
set PYSTE_INCLUDE=%PYSTE_INCLUDE% -I "E:\dev\ChampsSeries\include"
set PYSTE_INCLUDE=%PYSTE_INCLUDE% -I "E:\dev\ElementsFinis\include"
set PYSTE_INCLUDE=%PYSTE_INCLUDE% -I "E:\dev\StructuresAlgebriques\include"
set PYSTE_INCLUDE=%PYSTE_INCLUDE% -I "e:\dev\Toolkit\ToolKit_Geometrie\include"
set PYSTE_INCLUDE=%PYSTE_INCLUDE% -I "e:\dev\Toolkit\ToolKit_Collection\include"
set PYSTE_INCLUDE=%PYSTE_INCLUDE% -I "e:\dev\Toolkit\ToolKit_Systop\include"
set PYSTE_INCLUDE=%PYSTE_INCLUDE% -I "e:\dev\Toolkit\Toolkit_Exception\include"
set PYSTE_INCLUDE=%PYSTE_INCLUDE% -I "e:\dev\Toolkit\Toolkit_Config\include"
set PYSTE_INCLUDE=%PYSTE_INCLUDE% -I "F:\dev\TerraLib\terralib20\src\terralib\kernel"
set PYSTE_INCLUDE=%PYSTE_INCLUDE% -I "F:\dev\TerraLib\terralib20\src\terralib\function"
set PYSTE_INCLUDE=%PYSTE_INCLUDE% -I "E:\dev\boost_sup\include"
set PYSTE_INCLUDE=%PYSTE_INCLUDE% -I "C:\Program Files\boost-1.30.2"
set PYSTE_INCLUDE=%PYSTE_INCLUDE% -I "C:\Program Files\Python-2.2\include"

python "C:\Program Files\boost-1.30.2\libs\python\pyste\src\pyste.py" %PYSTE_INCLUDE%
↳                                                    --module=%PYSTE_MODULE% %PYSTE_MODULE%.pyste
```

Remarque:
Il faut ajouter l'appel de ce fichier commande dans le fichier `exec_PY_MonProjet.cmd` :

```
call exec_PY_SRSerieSimpleSRChampAnalytiqueDoubleEFTraits.cmd
```

PYSTE_MODULE représente le nom du module Boost Python que l'on souhaite créer.

13.2.4 Modifications sur les fichiers générés

- Après appel de la commande Windows, il faut remplacer (dans le fichier généré) :

```
BOOST_PYTHON_MODULE(PY_SRSerieSimpleSRChampAnalytiqueDoubleEFTraits)
```

par :

```
#define BOOST_PYTHON_FUNCTION(n) void declare__ ## n ()
BOOST_PYTHON_FUNCTION(PY_SRSerieSimpleSRChampAnalytiqueDoubleEFTraits)
```

Cette modification a pour effet de créer une fonction qui se nomme :
`declare__PY_ChampsSeries_SRSerieSimpleSRChampAnalytiqueDoubleEFTraits ()`.

- Il faut également corriger les *includes* qui sont entre crochets :

`<exemple.h>` devient donc `#include "exemple.h"`.

- Il faut vérifier (et modifier le cas échéant) les méthodes qui renvoient des objets par référence :

```
BOOST_PYTHON_FUNCTION(PY_SRSerieSimpleSRChampAnalytiqueDoubleEFTraits)
{
    class_< SRSerieSimple<SRChampAnalytique<double>,EFTraits>,
            SRSerieSimple_SRChampAnalytique_double_EFTraits_Wrapper >
          ("SRSerieSimpleSRChampAnalytiqueDoubleEFTraits", init<  >())
...
        .def("reqRegion", &SRSerieSimple<SRChampAnalytique<double>,EFTraits>::reqRegion,
↳                                  return_value_policy<copy_const_reference>()) // modifié
...
}
```

- Il faut mettre en commentaires les méthodes que l'on ne veut pas exporter en Python :

```
/*
        .def("executeAlgorithme",
↳           (void (SRSerieSimple<SRChampAnalytique<double>,EFTraits>::*)(int &) )
↳       &SRSerieSimple<SRChampAnalytique<double>,EFTraits>::executeAlgorithme,
↳           (void (SRSerieSimple_SRChampAnalytique_double_EFTraits_Wrapper::*)(int &))
↳       SRSerieSimple_SRChampAnalytique_double_EFTraits_Wrapper::default_executeAlgorithme)
*/
```

- Enfin, il faut appeler la nouvelle fonction créée (`declare__PY_SRSerieSimpleSRChampAnalytiqueDoubleEFTraits`) dans le fichier `PY_MonProjet.cpp` créé au début (paragraphe 13.2.2):

```
// Includes ====================================================================
#include <boost/python.hpp>

// Using =======================================================================
using namespace boost::python;

// Declarations ================================================================
void declare__PY_SRSerieSimpleSRChampAnalytiqueDoubleEFTraits();

// Module ======================================================================
BOOST_PYTHON_MODULE(PY_MonProjet)
{
    declare__PY_SRSerieSimpleSRChampAnalytiqueDoubleEFTraits();
}
```

Il ne reste plus qu'à construire la DLL PY_MonProjet.dll qui pourra être appelée en Python. Cependant cette dll, ne pourra être importée en Python que si les classes dont elle dépend ont déjà été importées en Python.

13.2.5 Remarques

13.2.5.1.1 Dépassement de capacité du Compilateur Visual C++

On utilise un seul fichier Pyste par classe C++ pour éviter une erreur du compilateur Visual C++. En effet, lors de la compilation d'un fichier C++ Boost Python trop complexe, le compilateur Visual C++ renvoie une erreur de dépassement de capacité. C'est pourquoi l'on utilise une classe Boost Python par fichier; cette classe Boost Python étant déclarée à l'aide de fonctions globales (`declare__PY_XXX`).

13.2.5.1.2 Classes parentes

Si on a besoin en Python de l'héritage existant en C++, il faut inclure dans le fichier Pyste la classe parente.

Fichier `PY_ChampsSeries_SRChampEF.pyste` :

```
SRChamp = Template("SRChamp", "PY_ChampsSeries.h")
set_policy(SRChamp.reqRegion, return_internal_reference())
SRChamp("double GOCoordXYZ<double>", "SRChampDoubleXYZ")

SRChampEF = Template("SRChampEF", "PY_ChampsSeries.h")
set_policy(SRChampEF.reqRegion, return_internal_reference())
SRChampEF("double EFTraits", "SRChampEFDoubleEFTraits")
```

Dans cet exemple, `SRChampEF` a pour parent `SRChamp`.

Dans l'exemple précédent, on a ajouté la classe `SRChampDoubleXYZ` au fichier Pyste; ce qui a pour conséquence d'ajouter du code C++ Boost Python relatif à cette classe dans le fichier généré par Pyste. Il faudra donc mettre en commentaires ce bloc de code pour n'avoir qu'une seule classe Boost Python par fichier généré.

Puisque la classe `SRChampEFDoubleEFTraits` a pour parent la classe `SRChampDoubleXYZ`, il faut également définir cette classe dans `PY_MonProjet`. Mais pour les raisons expliquées en 0, on le fera dans un nouveau fichier Pyste.

Remarque :
Il n'existe pas de classes abstraites en Python, les classes abstraites C++ ne peuvent donc être exportées en Python que si elles définissent les méthodes virtuelles pures.

13.3 Annexe 3 : Exemple d'interface graphique pour le module de scripts

L'activité « scripts » est insérée directement dans le menu principal de Modeleur :

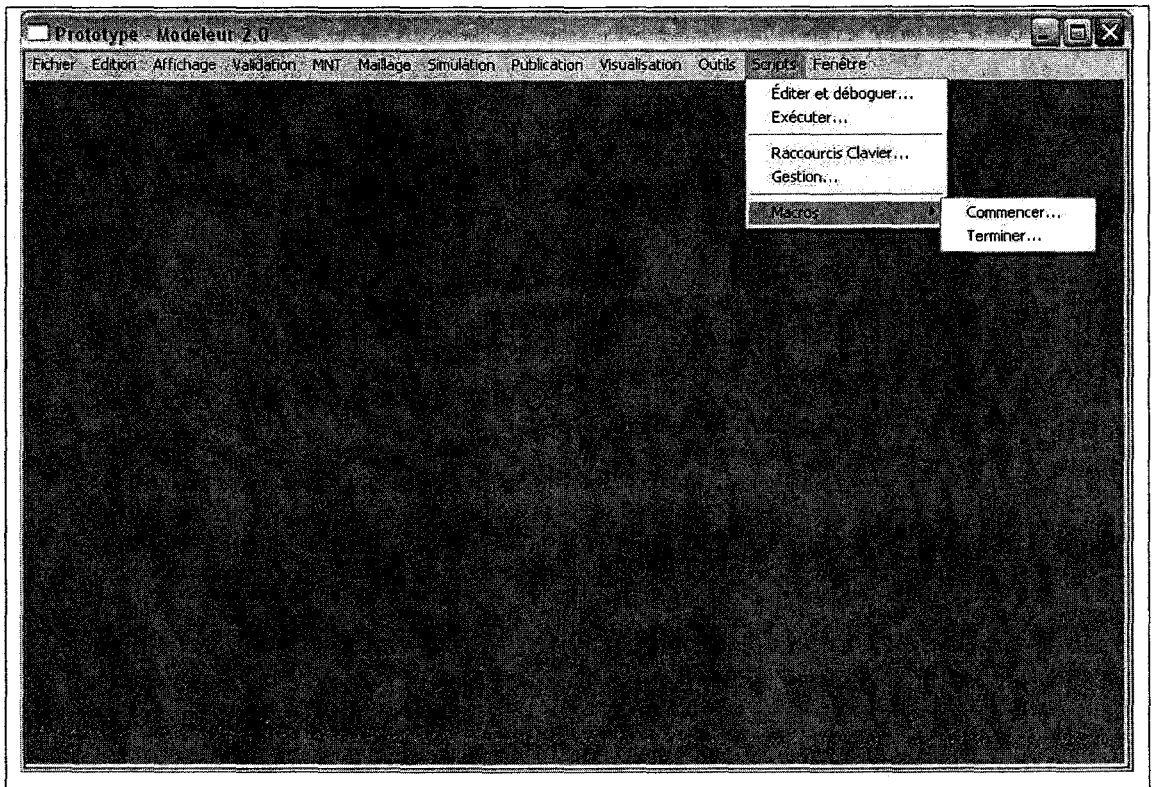


Fig-1 : Exemple du menu relatif au système de scripts

En cliquant dans le menu Scripts / Gestion, l'usager obtiendra la boîte de dialogue ci dessous :

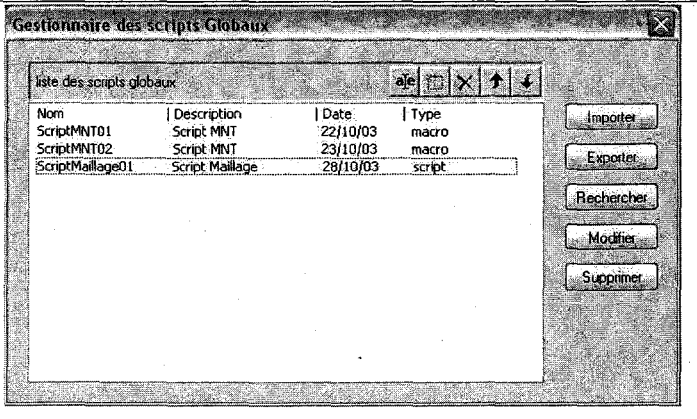


Fig-2 : Exemple de la boîte de dialogue de la gestion des scripts globaux

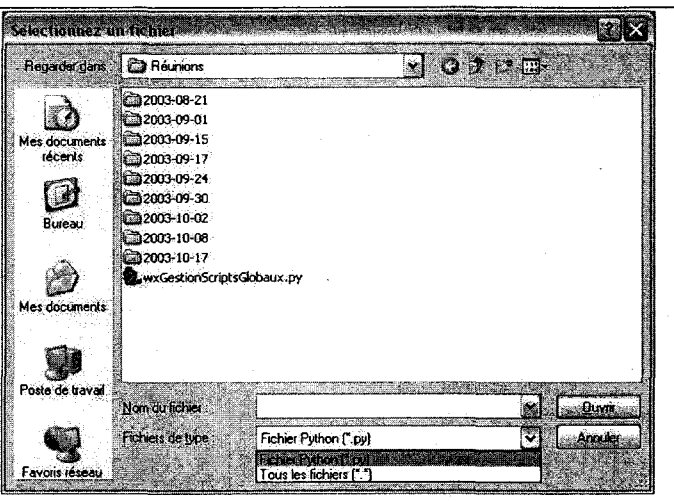


Fig-3 : Exemple de la boîte de dialogue de sélection d'un fichier local

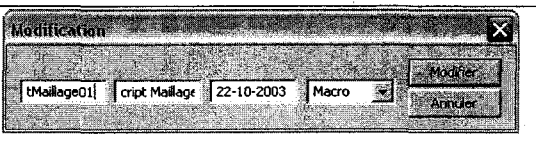
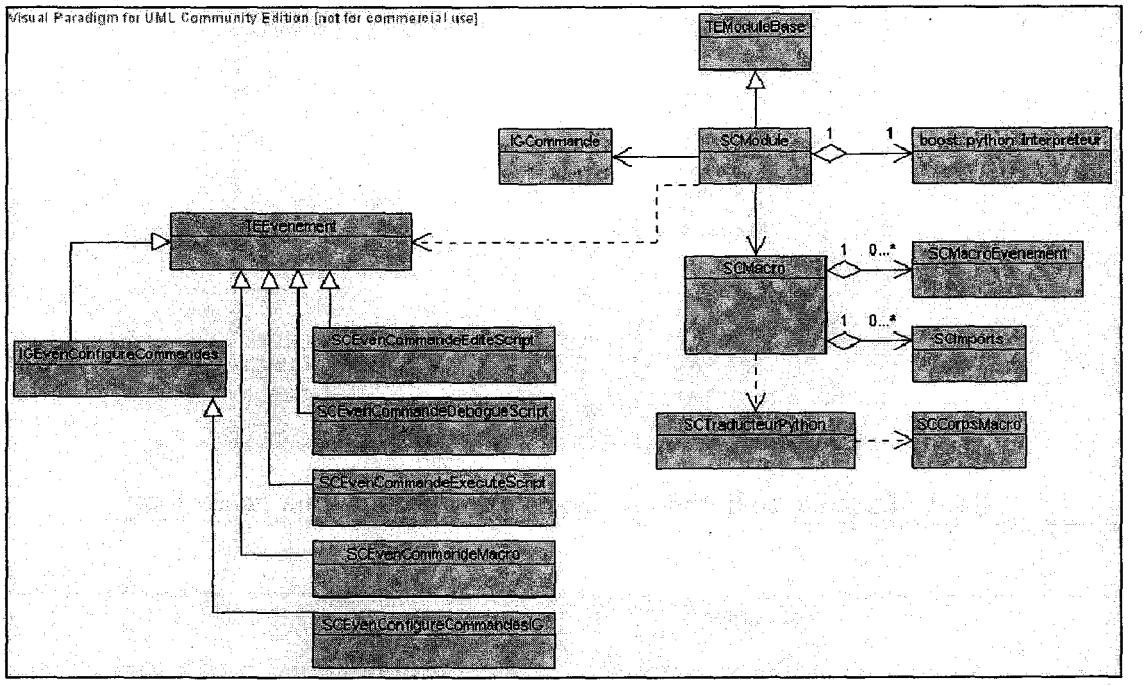
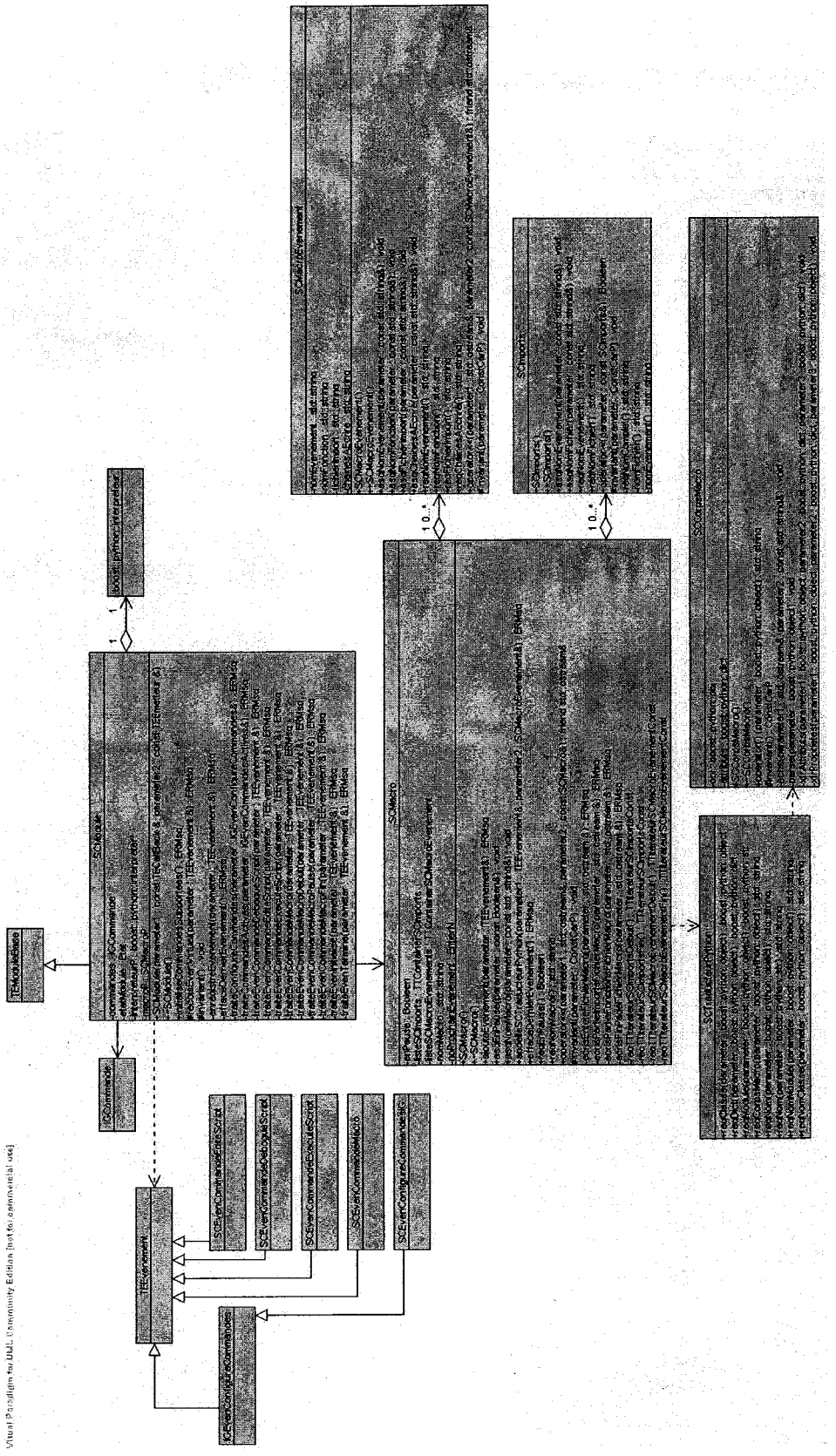


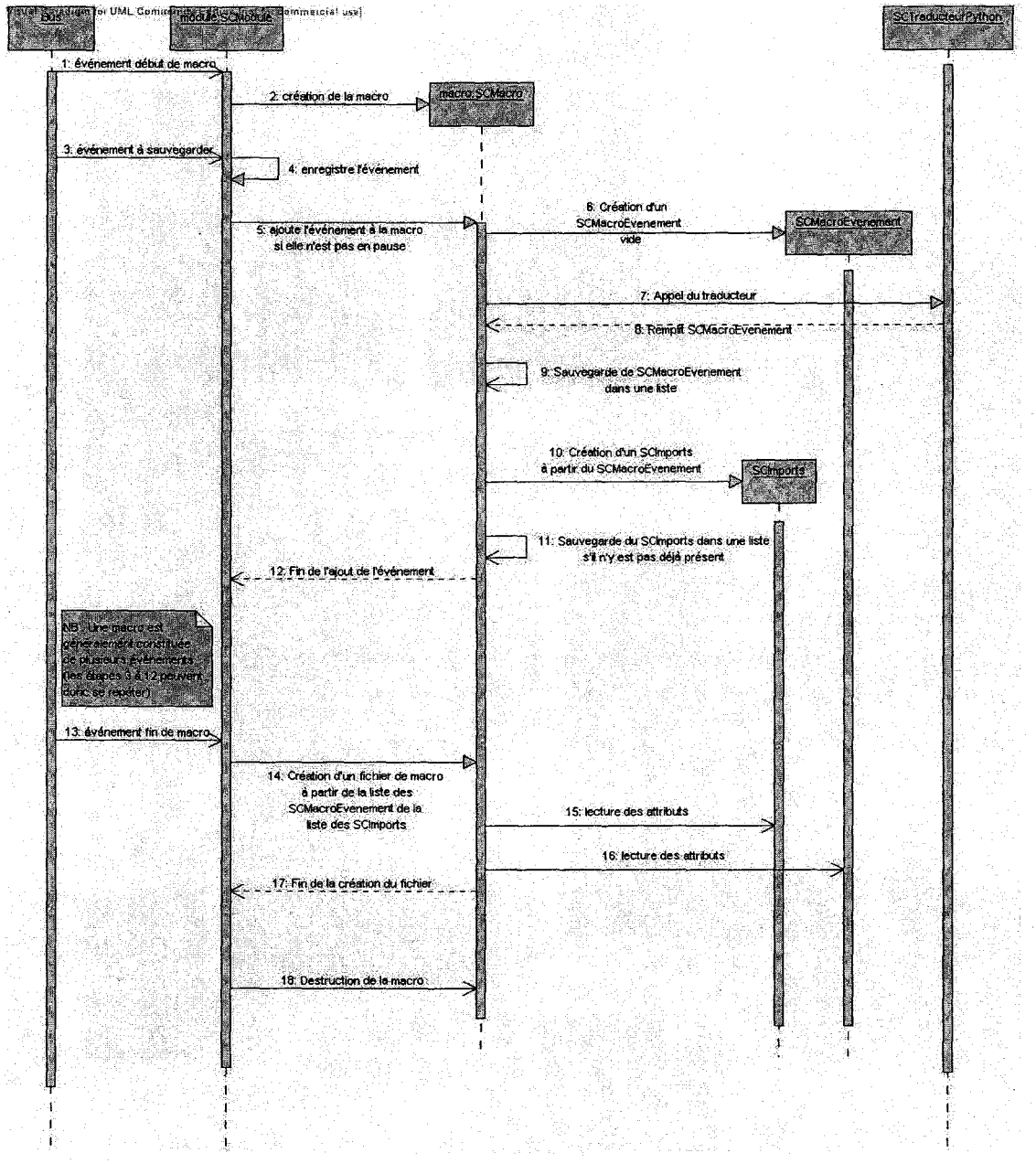
Fig-4 : Exemple de la boîte de dialogue de modification d'une entrée dans la BD (globale)

13.4 Annexe 4 : Diagramme des classes (UML)





13.5 Annexe 5 : Diagramme de séquence (UML)



Rapport de développement logiciel

Champs-Séries

Présenté par :

Olivier Kaczor

19-12-2003

1. Introduction

1.1 Objectifs

Le but de ce rapport est de présenter les différentes composantes de la partie Champs-Séries de Modeleur 2 et de discuter du développement de ceux-ci. Le rapport résume le travail d'analyse, de design et d'implantation effectué jusqu'ici et tente de déterminer les prochaines étapes de développement.

1.2 Contexte

Modeleur est un système d'information géographique (S.I.G.) dédié à l'hydraulique fluvial. Il permet en autre de construire un modèle numérique de terrain, d'effectuer des simulations d'hydraulique fluvial et de transport de contaminants et d'analyser les résultats obtenus. Les champs et séries contiennent des données spatiales qui proviennent de mesures ou de calculs.

Dans la première version de Modeleur, seules les données supportées par une structure éléments finis étaient possibles. La version 2.0 de Modeleur devra être capable de supporter d'autres types de support de données.

De plus, seules les données statiques étaient supportées dans Modeleur 1. Modeleur 2 doit permettre de manipuler des séries, que se soient des séries en débit ou des séries dans le temps.

Un champ représente une donnée spatiale stationnaire, comme la topographie de fond, la vitesse du courant ou les risques d'inondation. À partir des champs, on peut tracer des isolignes et des isosurfaces.

Une série permet de relier des champs suivant une logique, un axe, comme le temps pour une série temporelle. Les séries sont les principales structures de données avec lesquelles les utilisateurs de Modeleur 2 vont avoir à interagir.

1.3 Structure

Plusieurs concepts sont abordés dans ce rapport. Chacun des concepts est brièvement présenté et l'implantation dans Modeleur est expliquée. Les concepts discutés sont :

- *les structures algébriques;*
- *les éléments finis;*
- *les régions;*
- *les champs;*
- *les séries;*
- *les itérateurs.*

2 Design et développement

2.1 Structures algébriques

2.1.1 Concept mathématique

Une **structure algébrique** est un concept mathématique défini autour d'un nombre d'**ensembles** et d'un nombre de lois qui portent sur les éléments de ces ensembles.

Une **règle de composition binaire interne** assigne une valeur c appartenant à un ensemble S pour toutes paires ordonnées d'éléments (*a*, *b*) dans S x S. Soient *a*,*b*,*c* des éléments d'un ensemble S. Le symbole □ représente une règle de composition binaire interne dans la composition $a \square b = c$. Les structures algébriques les plus simples consistent en un ensemble et une de ces règles.

Les structures algébriques définies sur un ensemble sont caractérisées par quatre propriétés de leurs règles de composition binaires internes : **associativité**, **commutativité**, **identité** et **inverse**.

Le champ est la structure algébrique la plus puissante sur un ensemble. Un champ est muni d'une règle de composition binaire interne additive et d'une règle de composition binaire interne multiplicative. Ces règles doivent posséder les propriétés d'associativité, d'identité et d'inverse. Par exemple, les nombre réels avec l'addition et la multiplication forment un champ.

Un **groupe abélien** est une structure algébrique munie seulement d'une loi additive respectant ces trois propriétés (associativité, identité et inverse) tandis qu'un **groupe** est une structure algébrique munie seulement d'une loi multiplicative respectant ces trois propriétés.

Lorsqu'une structure algébrique est définie autour de deux ensembles ou plus, il est également possible de définir des **règles de composition externes**. Soient *a* un élément d'un ensemble V et *b*, *c* des éléments d'un ensemble S. Une règle de composition externe assigne une valeur *c* appartenant à S pour toutes paires ordonnées d'éléments (*a*, *b*) dans V x S. Le symbole □ représente une règle de composition externe dans la composition $a \square b = c$.

Une **algèbre** est une structure algébrique définie autour de deux ensembles respectant les propriétés des structures algébriques de type champ et munie d'une règle de composition externe multiplicative.

Pour plus de détail, voir en référence le livre Scientific and Engineering C++ de John Barton et Lee Nackman.

2.1.2 Implantation

Dans Modeleur 2, certaines structures comme les champs et les séries doivent contenir des données répondant aux critères d'une structure algébrique. Par exemple, les valeurs portées par les nœuds d'un maillage d'éléments finis doivent pouvoir être additionnées entre-elles et multipliées par des réels pour qu'il soit possible de les interpoler. Les nœuds doivent donc porter des valeurs respectant les propriétés de la structure algébrique de type algèbre.

Les classes de structures algébriques (SA) permettent de donner à un ensemble (type de données) les propriétés d'une structure algébrique. Le diagramme de classes des structures algébriques se trouvent en annexe. (Annexe 6.1)

Il est possible de créer un type de données pouvant être utilisé comme valeur nodale en utilisant la classe SAAlgèbre. La classe SRChamp représente une collection de valeurs nodales pouvant elle-même être utilisée comme valeur nodale d'une SRSerie (voir section 2.4). L'opération d'héritage suivante donne à la classe SRChamp les méthodes d'addition, de multiplication, d'identité et de multiplication externe avec des DReel de la classe SAAlgèbre :

```
class SRChamp : public SAAlgebre<SRChamp, DReel>
```

La classe SRChamp peut alors représenter des valeurs nodales d'un maillage éléments finis.

Toutes les valeurs de tous les types de données héritant de SAAlgebre<V, S> pourront donc utiliser les méthodes suivantes :

- opMul (const S&, const V&)
- opMul (const V&, const S&)
- opDiv (const V&, const S&)
- opMul (const S&, const S&)
- opDiv (const S&, const S&)
- opInv (const S&)
- identiteMul ( )
- opAdd (const S&, const S&)
- opSub (const S&, const S&)
- opNeg (const S&)
- identiteAdd ( )

Tous les ensembles ou types de données héritant d'une structure algébrique doivent cependant fournir des méthodes qui seront utilisées par les méthodes des structures algébriques. Par exemple, l'opérateur opMul(...) définit dans SAGroupe<T> requiert un operator*(const T&).

Un type héritant de SAGroupeAbelien<T> doit fournir :

- un opérateur+(const T&) ;
- un opérateur-(const T&) ;
- une méthode zero( ) retournant l'identité additive.

Un type héritant de SAGroupe<T> doit quant à lui fournir :

- un opérateur*(const T&) ;
- un opérateur/(const T&) ;
- une méthode uniteMul( ) retournant l'identité multiplicative;

Comme un SAChamp hérite de ces deux structures algébriques, un type héritant de SAChamp<T> doit fournir toutes ces méthodes.

Un type héritant de SAAlgebre<T, S> doit fournir toutes ces méthodes mais aussi :

- un opérateur*(const S&) ;
- un opérateur/(const S&) .

Il est standard de retrouver les opérateurs +=, -=, *= et /= avec les opérateurs +, -, * et /. Il est donc recommandé de les fournir également.

Notre classe SRChamp devrait donc ressembler à ceci :

```
class SRChamp : public SAAlgebre<SRChamp, DReel>
{
public:

   //--- Méthodes requises
   SRChamp uniteMul();
   SRChamp operator*(const SRChamp &);
   SRChamp operator/(const SRChamp &);
   SRChamp operator+(const SRChamp &);
   SRChamp operator-(const SRChamp &);
   SRChamp operator*(const DReel &);
   SRChamp operator/(const DReel &);
   SRChamp zero();
   //---

   //--- Méthodes standards
   SRChamp& operator*=(const SRChamp &);
   SRChamp& operator/=(const SRChamp &);
   SRChamp& operator+=(const SRChamp &);
   SRChamp& operator-=(const SRChamp &);
   SRChamp& operator*=(const DReel &);
   SRChamp& operator/=(const DReel &);
   //---

};
```

2.2 *Éléments finis*

2.2.1 Introduction

La méthode des éléments finis est une technique d'interpolation qui consiste à discrétiser en sous-ensembles un domaine (comme une rivière) dont on veut étudier des propriétés. Ces sous-ensembles sont appelés éléments et sont connectés entre eux par des nœuds. La discrétisation du domaine se nomme le maillage.

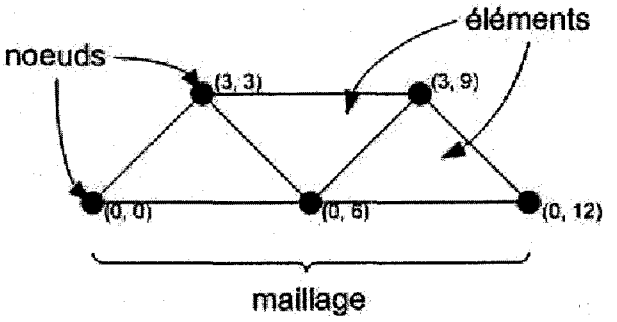


Figure 1 : Exemple d'un maillage éléments finis

Un ensemble de classes pour les éléments finis (EF) a été développé dans Modeleur 1. Ces classes fonctionnent avec des coordonnées de type GOCoordonneesXYZ et sont génériques pour le type de valeurs nodales. Dans Modeleur 2, il doit être possible de supporter n'importe quel type de coordonnées et de valeurs nodales.

2.2.2 Modification

Les templates ont été utilisées pour rendre les classes d'éléments finis génériques. Les nœuds sont définis avec des coordonnées. Le maillage et les éléments sont donc positionnés dans un espace grâce à leurs nœuds. Le maillage d'un domaine défini en trois dimensions doit obligatoirement avoir des nœuds définis en trois dimensions.

La classe de nœuds aura comme paramètre template un type de coordonnées (ex : GOCoordXYZ<DReel>) tandis que les classes de maillage et d'éléments auront comme paramètre template une structure (Trait) contenant l'information d'un maillage d'éléments finis.

Voir l'annexe 6.2 pour le diagramme de classes.

Le trait utilisé doit définir les types suivants :

- TTValeur : type des valeurs portées par une coordonnée
- TTCoord : type de coordonnées des nœuds
- TTNoeud : type des nœuds

- TTElement : type faisant référence aux éléments
- TTMaillage : type du maillage
- TTElementXX : types des éléments XX pour un maillage
- TTAlgo : type des algorithmes sur un élément
- TTAlgoConst : type des algorithmes constants sur un éléments

Différents traits peuvent être définis pour discrétiser des domaines différents. Par exemple, le trait à utiliser pour interpoler sur des domaines spatiaux en trois dimensions devrait ressembler à ceci :

```
struct EFTraits
{
   typedef DReel                                          TTValeur;
   typedef GOCoordXYZ<TTValeur>                           TTCoord;
   typedef MGNoeud<TTCoord>                               TTNoeud;
   typedef MGElement<EFTraits>                            TTElement;
   typedef MGMaillage<EFTraits>                           TTMaillage;
   typedef MGElementP1<EFTraits>                          TTElementP1;
   typedef MGElementL2<EFTraits>                          TTElementL2;
   typedef MGElementL3<EFTraits>                          TTElementL3;
   typedef MGElementL3L<EFTraits>                         TTElementL3L;
   typedef MGElementT3<EFTraits>                          TTElementT3;
   typedef MGElementT6<EFTraits>                          TTElementT6;
   typedef MGElementT6L<EFTraits>                         TTElementT6L;
   typedef MGElementQ4<EFTraits>                          TTElementQ4;
   typedef MGElementAlgorithme<EFTraits>                  TTAlgo;
   typedef MGConstElementAlgorithme<EFTraits>             TTAlgoConst;
};
```

Les valeurs des différentes dimensions d'une coordonnée doivent être accessible par les classes d'éléments finis. Les types de coordonnées utilisés dans un Trait doivent fournir un operator[] pour accéder à leurs valeurs. Par exemple, il est possible de faire :

```
EFTraits::TTCoord maCoord(5, 2, 7);
EFTraits::TTValeur composanteX = maCoord[1]; //composanteX = 5
EFTraits::TTValeur composanteY = maCoord[2]; //composanteY = 2
EFTraits::TTValeur composanteZ = maCoord[3]; //composanteZ = 7
```

2.3 *Régions spatiales*

2.3.1 Définition

Plusieurs structures de données de Modeleur, comme les champs, sont associées à des régions. Les régions spatiales sont des parties d'un espace. Elles peuvent être composées de plusieurs polygones ou contenir des trous. Les régions spatiales sont définies avec TerraLib, une librairie externe (voir section 3).

2.3.2 Adaptateur pour la librairie géométrique

Afin d'interagir avec la librairie externe, des classes ont été développées. Ceci permet d'utiliser les types de données internes à Modeleur pour définir des régions (ex : GOCoordXYZ<DReel>).

Les classes GORegion et GOBoite servent donc d'interface pour la librairie externe. Une région est créée avec des TTCoord et un travail interne créé la région avec la librairie. Le type de coordonnées TTCoord doit cependant contenir des valeurs pouvant être converties en double pour être compatible avec la librairie externe.

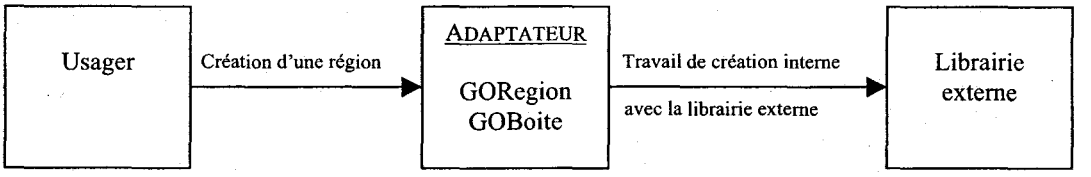


Figure 2 : Schéma résumant la création d'une région par un usager.

Voir le diagramme de classes en annexe (Annexe 6.3) pour connaître les méthodes contenues dans l'interface.

2.4 Champs

2.4.1 Concept de champs

Un champ dans Modeleur 2 est une structure de données qui possède, en plus des propriétés de la structure algébrique de type algèbre, des propriétés spatiales. Les données du champ répondent eux aussi aux structures algébriques de type algèbre et sont associées à une position dans l'espace. En effet, un champ est associé à une région spatiale. Un champ représente donc une grandeur répartie spatialement. Différents supports spatiaux pour les données du champ devront être supportés. En particulier :

- les éléments finis;
- les différences finies;
- les domaines d'équations analytiques;
- les rasters;
- les vectors.

Tous les types de champ possèdent des points en commun. On peut regrouper les méthodes des champs en catégories :

- des méthodes de gestion (ex. reqTypeChamp);
- des méthodes d'opérations sur les champs (ex. opDerive, opIntegre);

- des méthodes supportant sa structure spatiale (ex. pointInterieur);
- des méthodes retrouvant ses valeurs (ex. reqValeur)
- des méthodes du champ en tant que structure algébrique (ex. +, -, *, /).

La classe abstraite SRChamp constitue l'interface des différents types de champs. Voir le diagramme de classes en Annexe 6.4. Deux types ont été davantage travaillés : les champs éléments finis et les champs analytiques.

2.4.2 Champs éléments finis

Le champ éléments finis est la structure de données rassemblant les valeurs nodales sur un maillage éléments finis. Une valeur en tout point de la région spatiale du champ est obtenue par interpolation d'éléments finis. Pour réaliser l'interpolation, le champ doit être associé à un maillage et défini avec un Trait.

2.4.3 Champs analytiques

Un champ analytique est un champ défini par une équation mathématique. La région du champ peut alors être considérée comme le domaine de la fonction. L'équation du champ provient de code Python.

Voici un exemple de code Python pour définir un champ analytique 2D :

```
def Equation(x,y):
    return 2*x + y
```

Le champ analytique est alors défini par cette fonction équivalente en mathématique à f(x,y) = 2x+y. Cette définition est introduite dans l'interpréteur Python.

Lors de l'assignation d'une équation au champ, le nom de la fonction doit être donné. Lorsque l'usager fera une requête sur le champ, cette fonction sera évaluée dans l'interpréteur Python avec les paramètres de l'usager. L'équation et le nom de la fonction sont données au champ avec la méthode :

```
asgEquation(const std::string& equation, const std::string& nomFonction)
```

Comme pour tous les types de champs, les opérateurs +, -, *, /, etc. doivent être définis pour respecter les propriétés des structures algébriques de type algèbre. Un nouveau champ est alors créé avec les deux équations. Une troisième équation faisant l'opération est aussi créée à l'interne.

Par exemple, soient

ChampAna1 définit par l'équation :

```
def f1(x,y):
    return 2*x + y
```

ChampAna2 définit par l'équation :

```
def f2(x,y):
    return x - y
```

ChampAna3 = ChampAna1 + ChampAna2

ChampAna3 sera alors défini par le code Python suivant:

```
def f1(x,y):
    return 2*x + y

def f2(x,y):
    return x - y

def f1Plusf2(x,y)
    return f1(x,y) + f2(x,y)
```

Un problème survient lorsque les deux champs contiennent une fonction du même nom (ex. si f2 s'appellerait f1). Pour régler ce problème, chaque champ doit être évalué dans un namespace différent. Il est toutefois important de ne pas limiter les usagers à l'utilisation des fichiers pour créer les namespaces. Il existe 3 façons de créer des namespaces en Python : les classes, les modules et les fonctions. Les méthodes des classes et des modules ont été analysées.

Namespaces avec des classes :

Il a été tenté en premier lieu de simuler les namespaces avec des classes Python contenant les équations. Cette méthode comporte malheureusement des problèmes. En effet, les méthodes d'une classe Python doivent absolument avoir une instance de la classe comme premier paramètre. L'équation du champ donnée par l'usager devrait alors être modifiée pour devenir une méthode de la classe. Ce problème peut par contre être résolu en déclarant les méthodes static.

```
class ChampAna:
    def champAna1(x,y):
        return x+y
    champAna01 = staticmethod(champAna1)
```

Un autre problème existe toutefois avec les classes. Les méthodes doivent être appelées avec une instance. Il faudrait alors créer une instance de la classe (ins) et l'utiliser dans l'équation entrée par l'utilisateur :

```
def f1(x,y):
    return 2*x + y

def f2(x,y):
    return x - y

def f3(x,y)
    return f1(x,y) + f2(x,y)
```

```
def f1(x,y):
    return 2*x + y

def f2(x,y):
    return x - y

def f3(x,y)
    return ins.f1(x,y) + ins.f2(x,y)
```

Namespaces avec des modules sans fichier :

La méthode retenue pour simuler les namespaces est celle des modules. Il est possible de créer des modules en Python sans avoir recours à des fichiers. Chaque champ possède donc un module dans lequel est évaluée son équation. Lors d'une opération (+, -, *, /, etc.), un nouveau champ est créé avec un nouveau module dans lequel sont insérés les deux modules des deux champs utilisés dans l'opération.

Chaque module doit cependant avoir un nom unique. Afin de s'assurer de l'unicité des noms de modules, chaque module a comme nom ChampAnaXXXXXYYZZBBBBBB où XYZ est le nombre de jours, d'heures et de minutes écoulés depuis le 1er janvier 1970 et B un compteur de champs analytiques créés dans la session courante.

2.5 Séries

2.5.1 Concept de séries

La série est la structure de donnée avec laquelle les utilisateurs de Modeleur vont interagir.

Comme un champ, une série est une structure de données respectant les propriétés de la structure algébrique algèbre. La différence entre le champ et la série se situe au niveau de la région à laquelle la structure est associée. Les régions des séries ne sont pas des régions spatiales.

Les données d'une série doivent être en relation. Par exemple, une série temporelle doit contenir des données qui diffèrent selon le temps. Il est possible de créer une série temporelle où les valeurs nodales de la série sont des champs de débits d'une rivière à différents moments de l'année. Un champ de débit peut alors être interpolé pour connaître le débit d'un mois en particulier. Un champ de débit maximum pourrait aussi être calculé. La région de cette série serait le domaine du temps. Les données des séries peuvent dépendre de plusieurs domaines. Une série de température pourrait être définie en fonction du temps et de la force du vent.

Il existe deux types de séries concrètes : les séries simples et les séries composées. La classe abstraite SRSerie constitue l'interface des séries. Voir le diagramme de classe en annexe 6.4.

2.5.2 Séries simples

Une série simple est associée à un maillage. Ce maillage de série, comme son semblable pour les champs, possède des éléments et des nœuds. Un élément série est également associé à un ou plusieurs nœuds de série. Mis à part le type de coordonnées, la structure des séries est exactement la même que celle des champs éléments finis. Il a donc été convenu qu'une série soit associée à un maillage d'éléments finis avec un Trait différent.

Une série simple agit exactement comme un champ éléments finis. La série simple a donc un attribut de type champ éléments finis sur lequel sont effectuées les opérations de la série. Une addition de deux séries simples consiste donc à additionner les attributs champs des deux séries simples. Ceci évite d'écrire deux fois les mêmes fonctions.

Le type d'éléments (P1), représentant un élément composé d'un seul nœud a été créé afin de pouvoir manipuler l'équivalent du champ dans Modeleur 1 par le biais d'une série.

2.5.3 Séries composées

Les séries peuvent être combinées entre elles de manière à créer de nouvelles séries. Il ne faut toutefois pas générer une nouvelle série à chaque fois qu'un utilisateur veut faire ce genre d'opération. Pour permettre de manipuler ces compositions comme s'il s'agissait d'une série simple, le pattern Composite a été utilisé. Il permet de manipuler une collection d'objets appartenant à une même hiérarchie comme s'il s'agissait d'un objet unique de la classe mère. Les opérations de série peuvent donc être effectuées sur une combinaison de séries.

Pour être combinées, deux séries doivent être compatibles. Des séries sont compatibles si et seulement si elles possèdent un axe commun. Par exemple, une série de débits dans le temps est compatible avec une série de températures dans le temps pour créer une série de débits en fonction de la température.

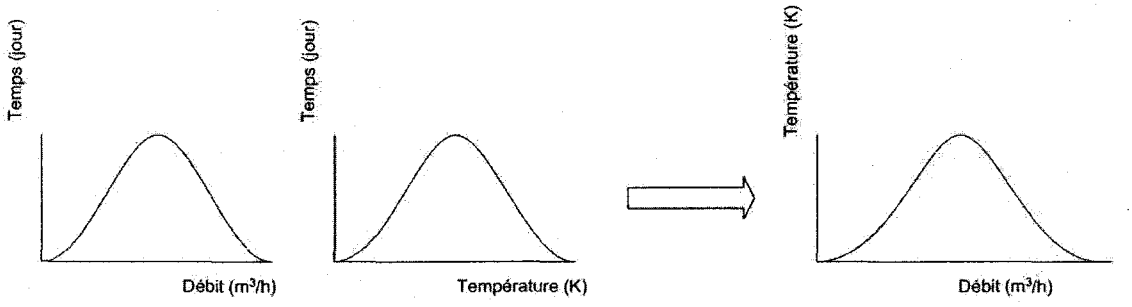


Figure 3 : Exemple de combinaison de séries.

2.6 Itérateurs

2.6.1 Introduction

Modeleur utilise des structures de données spécifiques comme les champs et les séries. Afin de parcourir les valeurs contenues dans ces structures, quatre types d'itérateurs ont été développés : des itérateurs spatiaux, des itérateurs de régions, des itérateurs de champs et des itérateurs de série.

Les itérateur spatiaux sont les itérateurs permettant de se déplacer dans une espace et sont utilisés par les itérateurs de région. Les itérateurs de champ et de série sont des itérateurs de région qui retourne une valeur lorsque **déréférencés**.

2.6.2 Itérateurs spatiaux

Les itérateurs spatiaux permettent de se déplacer dans un espace. Un itérateur spatial se déplace selon un pas d'itération sur un espace défini avec une coordonnée maximale et une coordonnée minimale. Le type de coordonnées utilisé doit fournir une fonction estDedans(...) retournant vrai si une coordonnée est entre deux autres coordonnées et faux sinon.

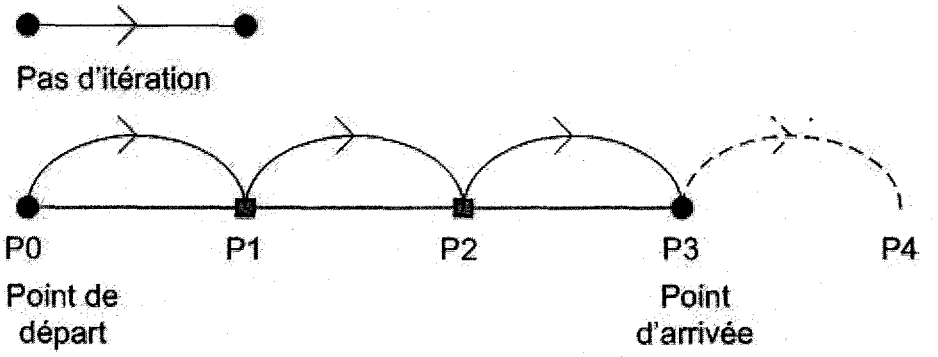


Figure 4 : Exemple d'itération sur un espace. La méthode P4.estDedans(P0, P3) retourne faux.

Les itérateurs spatiaux peuvent se promener sur une structure (op++ et op--) et être comparés entre eux (op== et op!=). Ce type d'itérateurs est appelé itérateur bidirectionnel. Les itérateurs spatiaux sont utilisés par les itérateurs de région.

2.6.3 Itérateurs de régions

Une région est un exemple de structure spatiale. Un itérateur de région est donc composé de plusieurs itérateurs spatiaux.

Les itérateurs de régions permettent de scanner une région. L'utilisateur de ce genre d'itérateurs doit fournir un certain nombre de paramètres. Il doit spécifier les directions dans lesquelles il veut itérer ainsi que les pas à prendre dans chacune des directions. En mathématique, un **vecteur** est composé d'un sens, d'une direction et d'une longueur. Le constructeur d'iterateurs de région doit donc recevoir autant de vecteurs en paramètres que le nombre de dimension de la région. Un itérateur de région construit sans paramètre pointe à l'extérieur d'une région. Il peut être utilisé comme marqueur de fin d'une région.

Les itérateurs de régions actuels ne supportent que les régions en deux dimensions. Pour faciliter l'itération, l'itérateur de région se déplace sur le plus petit rectangle contenant la région (« bounding box »). À chaque itération, l'itérateur vérifie s'il se trouve dans la région sinon il doit continuer à itérer.

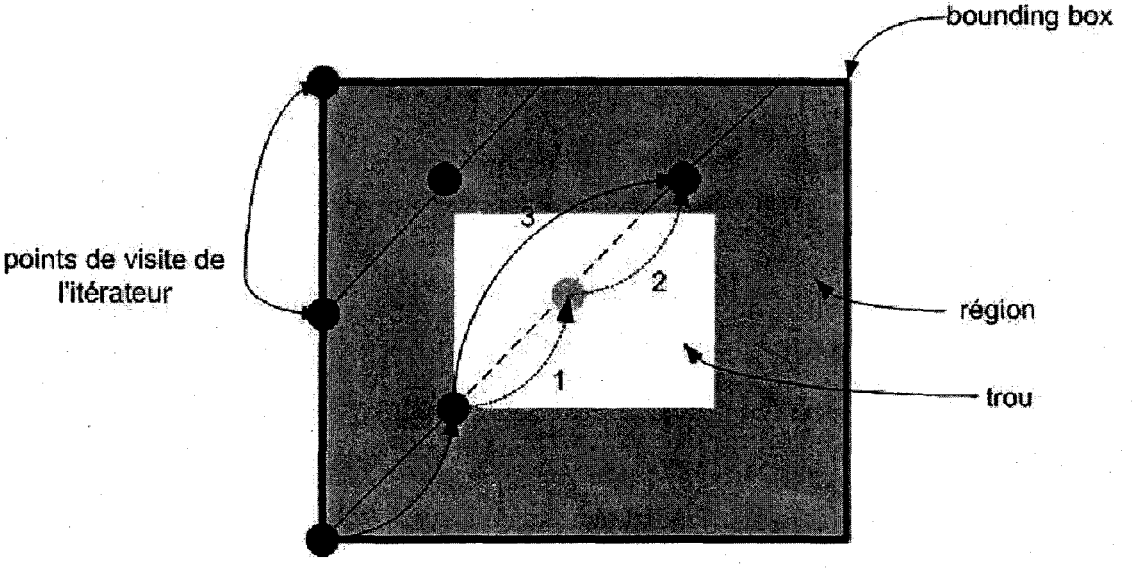


Figure 5 : Exemple d'itération sur une région dans un bounding box. L'itérateur vérifie s'il se trouve dans la région sinon il doit continuer à itérer. Le déplacement 1 et 2 sont remplacés par le déplacement 3 puisque le point d'arrivée de 1 n'est pas dans la région.

Afin de scanner une région au complet, l'itérateur de régions 2D utilise deux itérateurs spatiaux. Le premier (iterA) se déplace sur le plus petit rectangle selon son pas d'itération tandis que le deuxième (iterB) se déplace sur les arêtes du rectangle. Le pas de cet itérateur est obtenu en projetant le deuxième vecteur sur l'axe des X ou Y.

Dans les cas où le premier itérateur itère en 2D sur l'axe des X et l'axe des Y, le deuxième itérateur devra se repositionner (départ B2) pour scanner la portion de région non visitée. Le deuxième itérateur change aussi son pas pour itérer sur l'axe perpendiculaire à son ancien pas.

La position du premier itérateur est celle de l'itérateur de région. L'itération se termine lorsque les deux itérateurs indiquent la fin.

Le sommet du bounding box utilisé comme point de départ de l'itérateur est déterminé en fonction des vecteurs d'entrés. Un tableau donnant les sommets de départ selon les vecteurs d'entrés se trouve en Annexe 6.3.1.

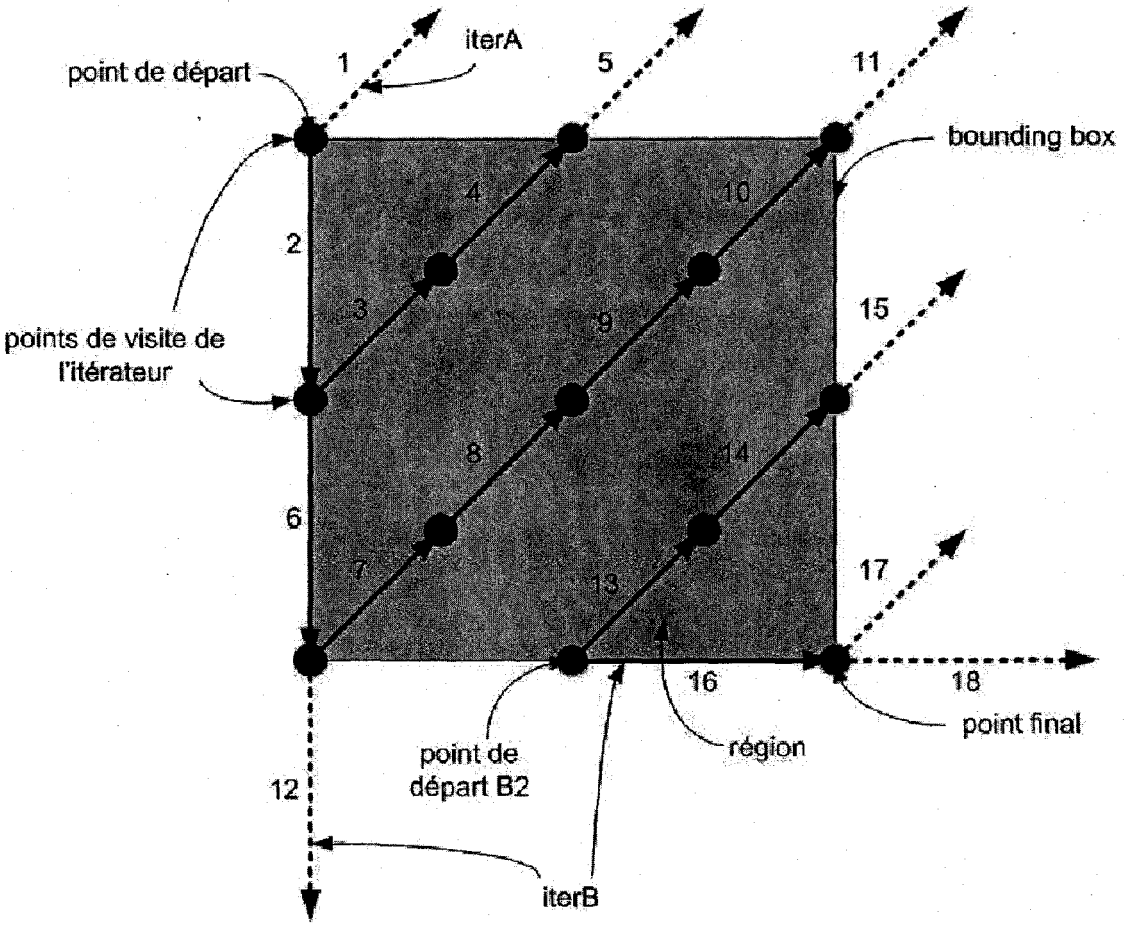


Figure 6 : Schéma résumant l'itération sur une région. L'itération se termine lorsque les deux itérateurs sont hors de la région (17, 18). Le sommet de départ a été déterminé avec le tableau de l'annexe 6.5.1. Le premier vecteur a un angle plus petit que 90^0 avec l'axe des X et le deuxième a un angle plus petit que 360^0 avec ce même axe. Le sommet est donc le point supérieur gauche du bounding box.

Certaines régions, comme les espaces curvilignes (régions 3D suivant des abscisses curvilignes 2D), posent un problème. Rien ne prouve que les itérateurs atteindront la région en itérant sur le bounding box. Ces régions peuvent toutefois être transformées en régions équivalentes 2D sur lesquelles il est possible d'itérer convenablement.

2.6.4 Itérateurs de champs

Un champ est un ensemble de valeurs associé à une région. Un itérateur de champ est donc un itérateur de région qui, lorsque déréférencé, retourne la valeur du champ à la coordonnée indiquée par l'itérateur. Pour ce faire, il doit conserver un lien avec le champ sur lequel il itère. La classe SRIterateurChamp représente les itérateurs de champs. Le constructeur d'un SRIterateurChamp reçoit donc en paramètre un pointeur sur le champ en plus des vecteurs d'itération.

2.6.5 Itérateurs de séries

Comme les séries agissent comme des champs éléments finis, les itérateurs de séries sont des itérateurs de champs.

3 Recherche d'une librairie pour les régions

3.1.1 Analyse des besoins

Modeleur utilise le concept de régions spatiales. Plusieurs requêtes peuvent être faites sur ces régions et il existe des librairies où ces opérations sont déjà implantées. Des librairies géométriques ont été analysées en fonction des besoins de Modeleur.

Les régions de Modeleur ont certaines particularités qui devront être supportées par la librairie choisie. Tout d'abord, les itérateurs de région sont implantés en utilisant des fonctions retournant le bounding box d'une région et déterminant si un point quelconque se trouve à l'intérieur d'une région. Ce genre de requête devra être possible avec la librairie choisie.

Une région peut être trouée ou être composée de plusieurs polygones. Il est donc primordial que la librairie géométrique supporte les multi-polygones et les multi-lignes.

Voici une liste de fonctionnalités que la librairie doit fournir :

- fonction retournant le bounding box d'une forme géométrique;
- fonction pouvant déterminer si un point se trouve à l'intérieur d'une forme géométrique;
- représentation d'une forme géométrique avec plusieurs polygones;
- représentation d'une forme géométrique avec plusieurs points;
- représentation d'une forme géométrique avec plusieurs lignes.

3.1.2 Analyse des différentes librairies

Cinq librairies géométriques ont été analysées selon nos besoins : CGAL, LEDA, VXL, GEOS et TerraLib. Un tableau résumant les fonctionnalités de ces librairies se trouve en annexe (Annexe 6.6.1).

Parmi ces librairies, GEOS et TerraLib semblent particulièrement intéressantes. Elles répondent aux besoins de base énoncés ci-dessus et offrent des fonctionnalités GIS. Deux tableaux résumant les fonctionnalités (Annexe 6.6.2) et la documentation (Annexe 6.6.3) de ces librairies se trouvent en annexe.

TerraLib est une librairie utilisée pour développer des applications GIS utilisant des bases de données spatiales. Elle offre davantage de fonctionnalités que GEOS telles que des classes représentant les rasters, les réseaux et les projections. Elle fournit également des classes pour interagir avec des bases de données spatiales comme Oracle, PostGreSQL, MySQL et ADO.

Les classes de raster pourraient être utilisées pour représenter les champs rasters. Les rasters de TerraLib sont considérés comme des formes géométriques. Il est donc possible de faire les opérations de la structure spatiale des champs sur eux.

Il est également possible de créer des opérateurs +, -, *, / etc. sur eux. En effet, les valeurs sont conservées dans une matrice. Des méthodes de requêtes sont implantées et pourraient être utilisées pour retourner la valeur du champ à un point donné. Des méthodes coord2index et index2coord sont également fournies pour interagir avec la matrice.

Les rasters de TerraLib possèdent des itérateurs mais nos itérateurs spatiaux pourraient aussi être utilisés.

Les valeurs contenues dans un raster de TerraLib doivent par contre obligatoirement être des doubles.

Les classes de bases de données fournissent des méthodes pour sauvegarder les rasters dans des tables. Toutes les fonctionnalités des champs rasters sont donc possibles avec la librairie TerraLib.

3.1.3 Choix de la librairie géométrique

Parmi les librairies analysées, GEOS et TerraLib sont les plus intéressantes avec des fonctionnalités GIS. Nous avons finalement opté pour TerraLib puisqu'elle offre davantage de possibilités et pourrait être utile lors de l'implantation d'autres concepts comme les champs rasters.

4 Conclusion

4.1 Résumé

La première étape de l'implantation des champs et séries était de modifier le travail existant sur les éléments finis afin de pouvoir l'utiliser dans le nouveau design. Les classes sont devenues génériques et peuvent maintenant être utilisées avec différents types de coordonnées.

Les concepts de champs et séries ont ensuite été implantés. Parmi les types de champs que Modeleur 2 devra supporter, seuls les champs éléments finis et analytiques ont été travaillés. La structure pour ajouter les autres types de champs est par contre solide. Des itérateurs ont aussi été développés pour se déplacer sur les champs et séries.

Les champs et les séries sont associés à des régions et une analyse a été faite pour trouver une librairie géométrique pouvant représenter les régions spatiales dans Modeleur. TerraLib a été choisie car elle offre des fonctionnalités intéressantes tout en répondant au besoins essentiels de Modeleur. Elle est également gratuite, bien documentée et constamment en évolution.

Le développement des champs et séries devra être poursuivi afin de répondre aux spécifications de Modeleur 2 mais le travail effectué constitue une bonne base pour la suite du projet.

4.2 Perspectives futures

Les régions des séries ne seront pas nécessairement des régions spatiales. Par exemple, il peut exister des séries temporelles. Pour l'instant, il n'existe aucune structure supportant d'autres types de régions.

Le design actuel ne permet pas non plus aux itérateurs de région d'itérer sur plus de 2 dimensions. Il ne faudrait toutefois pas poser de limites. Afin de permettre aux itérateurs d'itérer en n dimensions, il serait intéressant de créer un itérateur 1D pouvant être appelé récursivement pour naviguer en n dimensions.

Finalement, plusieurs méthodes des champs et des séries comme les dérivés n'ont pas encore été implantées.

5 Glossaire

Associativité : Propriété d'une règle de composition interne. Une loi f est dite associative si pour tous x,y et z, on a: $f(x,f(y,z)) = f(f(x,y),z)$

Commutativité : Propriété d'une règle de composition interne. Une loi f est dite commutative si pour tous x et y, on a: $f(x,y) = f(y,x)$

Champ : Structure algébrique sur un ensemble. Un ensemble est considéré comme un champ s'il permet l'associativité, l'identité et l'inverse sur une règle de composition binaire interne additive et une règle de composition binaire interne multiplicative.

Un champ dans Modeleur 2 est aussi une structure de données qui possède, en plus des propriétés de la structure algébrique, des propriétés spatiales.

Déréférencer : Terme informatique signifiant accéder à la valeur pointée par une variable contenant une adresse mémoire.

Ensemble : Collection d'objets considérée dans sa totalité. {1,2,3} est un exemple d'ensemble de chiffres.

Groupe : Structure algébrique sur un ensemble. Un ensemble est considéré comme un groupe s'il permet l'associativité, l'identité et l'inverse sur une règle de composition multiplicative.

Groupe abélien : Structure algébrique sur un ensemble. Un ensemble est considéré comme un groupe abélien s'il permet l'associativité, l'identité et l'inverse sur une règle de composition additive.

Identité : Élément neutre d'une règle de composition interne. Un élément e est l'élément neutre d'une règle de composition interne $\square$ si on a : $a \square e = a$ ou $e \square a = a$. 0 est l'élément neutre de l'addition sur des entiers.

Inverse : Soient i,e et x trois éléments d'un ensemble S. i est l'élément inverse de x pour une règle de composition interne $\square$ si on a : $e \square x = i$ où e est l'identité de $\square$.

On dit qu'un élément est inversible dans un ensemble pour une loi de composition interne si son inverse existe dans le même ensemble. 2 est inversible pour l'addition dans les entiers (inverse = -2) mais n'est pas inversible pour l'addition dans les entiers naturels (-2 n'est pas défini dans les entiers naturels).

Itérateur : Structure permettant de se déplacer sur une autre structure de données.

Itérateur spatial : Itérateur permettant de se déplacer dans une dimension d'un espace entre un point de départ et un point d'arrivée selon un pas.

Région : Parties d'un espace.

Règle de composition externe : Soient *a* un élément d'un ensemble V et *b*, *c* des éléments d'un ensemble S. Une règle de composition externe assigne une valeur *c* appartenant à S pour toutes paires ordonnées d'éléments (*a*, *b*) dans V x S. Le symbole □ représente une règle de composition externe dans la composition $a \square b = c$.

Règle de composition binaire interne : Soient a;b,c des éléments d'un ensemble S. Une règle de composition binaire interne assigne une valeur c appartenant à S pour toutes paires ordonnées d'éléments (a,b) dans S x S. Le symbole □ représente une règle de composition binaire interne dans la composition a □ b = c.

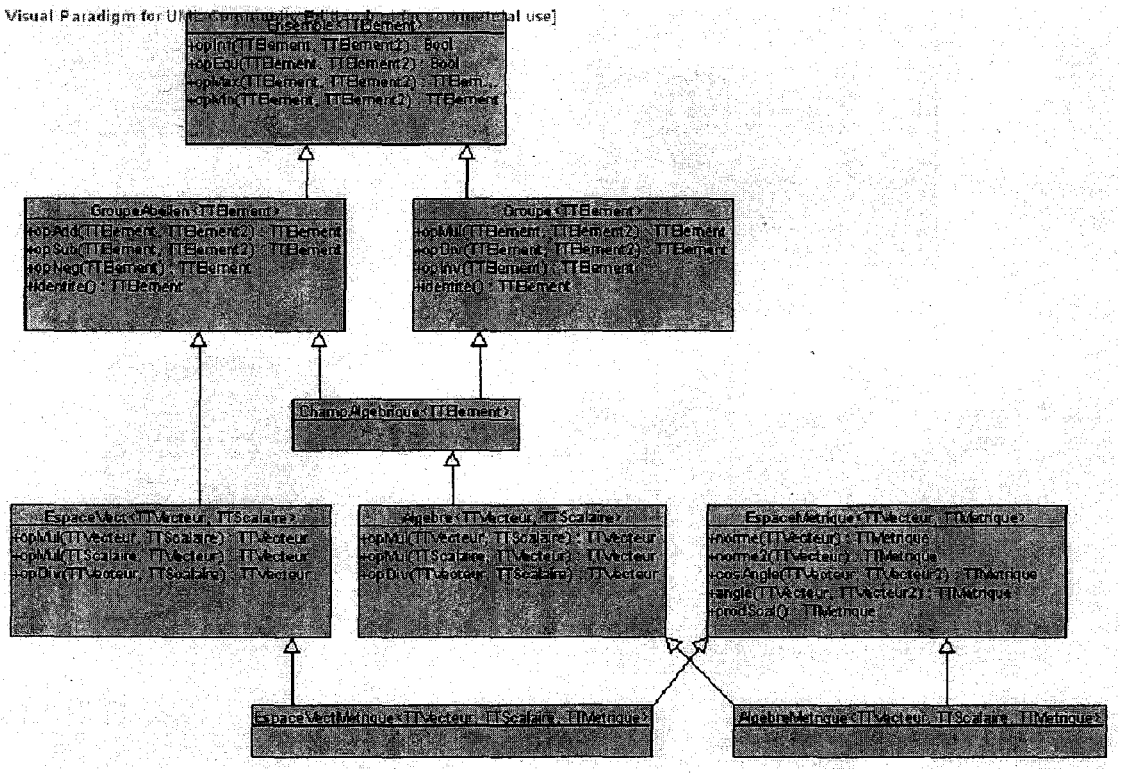
Série : Une série est une structure de données qui possède, en plus des propriétés de la structure algébrique champ, des propriétés spatiales.

Structure algébrique : Concept mathématique défini selon un nombre d'ensembles et un nombre de lois qui portent sur les éléments de ces ensembles.

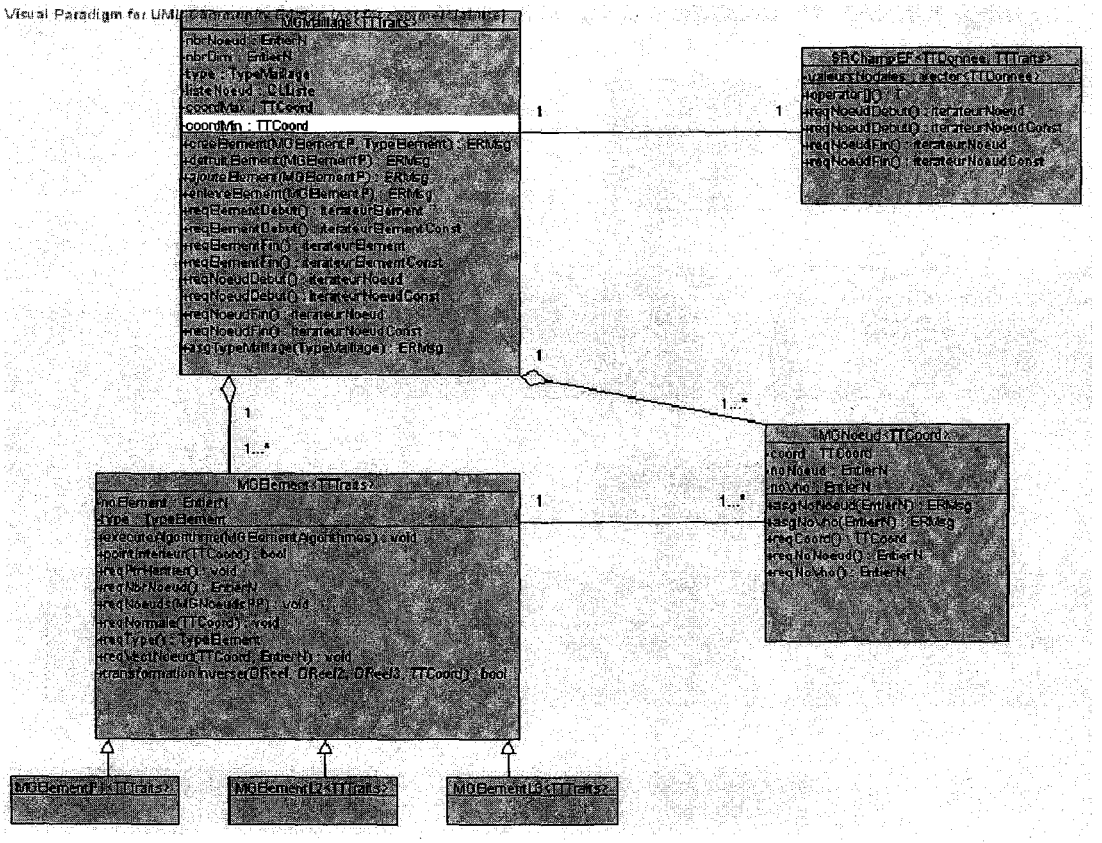
Vecteur : Structure mathématique composée d'un sens, d'une direction et d'une longueur.

6 Annexes

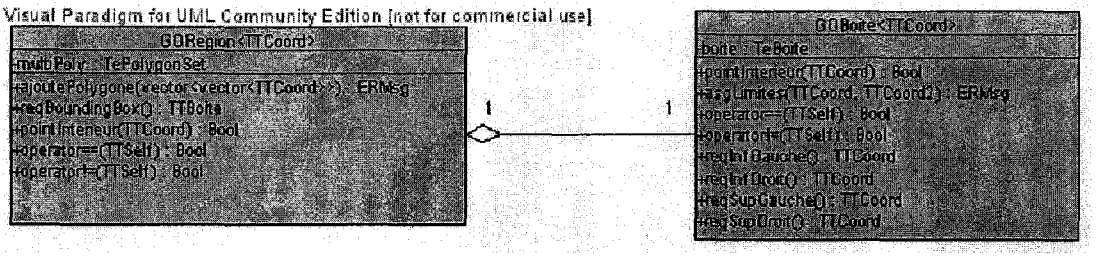
6.1 Diagramme de classes des structures algébriques



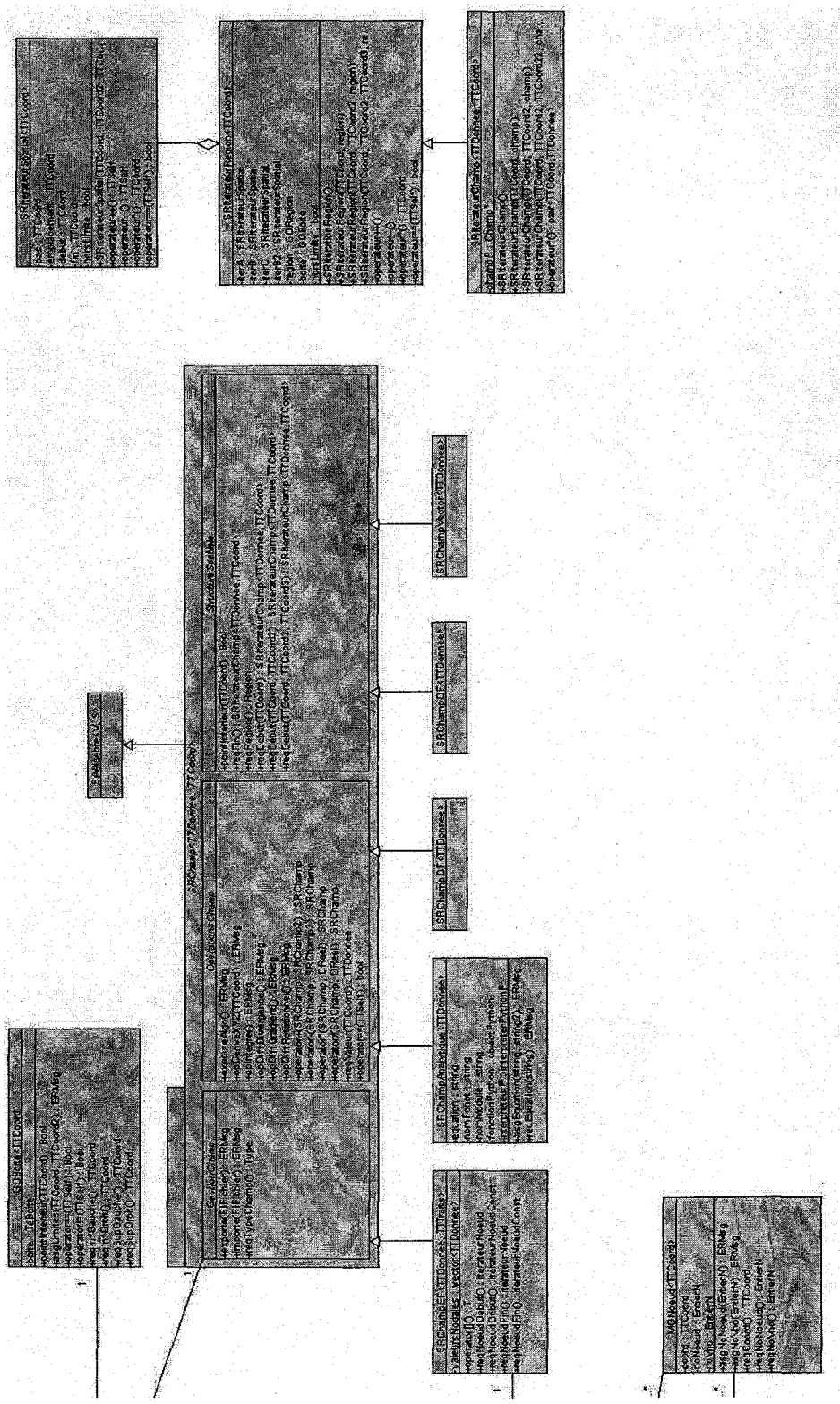
6.2 Diagramme de classes des éléments finis

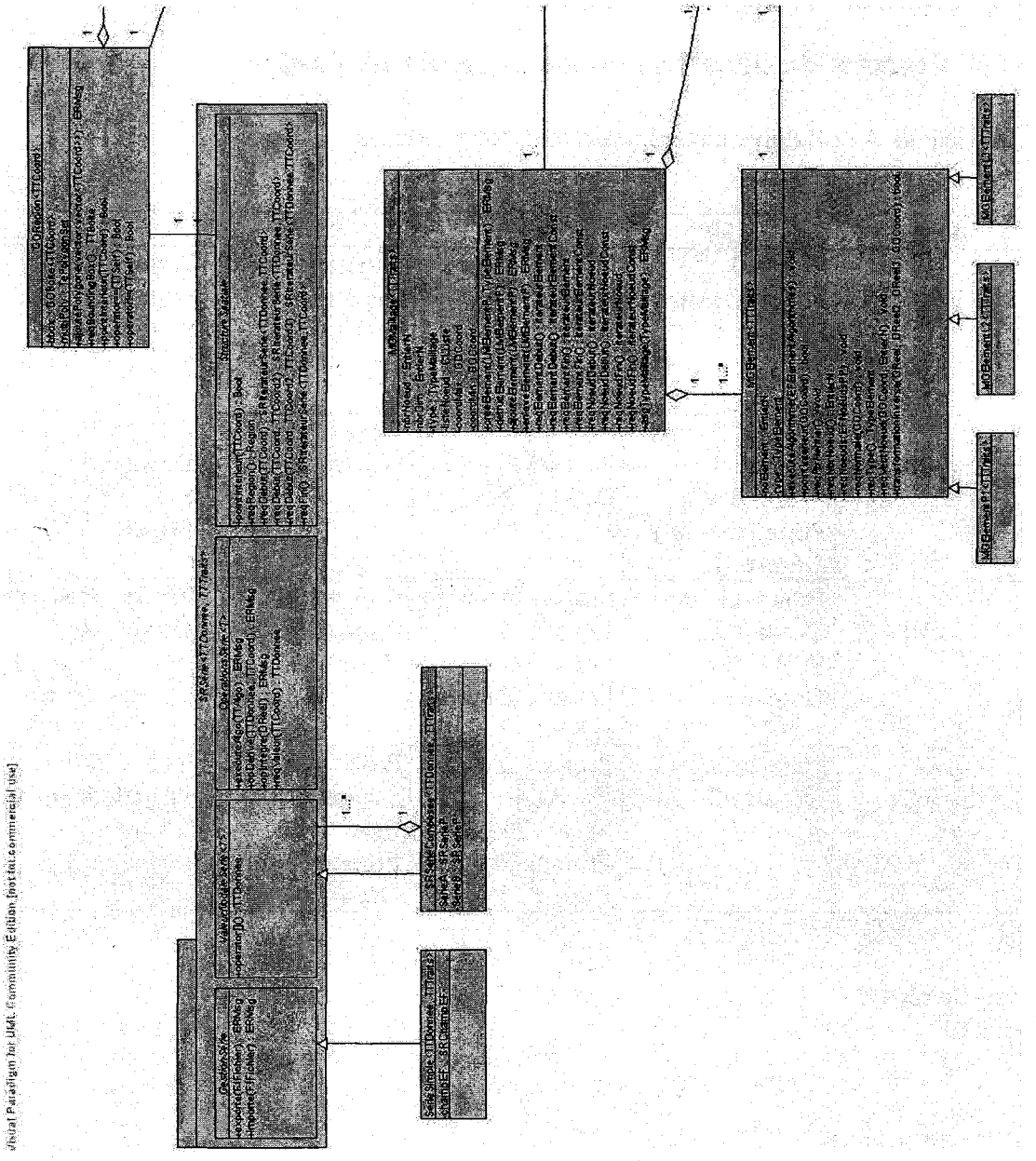


6.3 Diagramme de classes de l'adaptateur de régions



6.4 Diagramme de classes complet des Champs/Séries





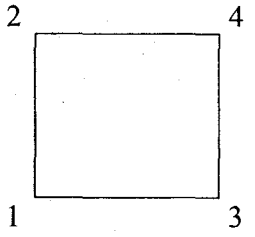
6.5 Sommet de départ selon les vecteurs d'itération

6.5.1 Sommet de départ selon les vecteurs d'itération

Les vecteurs A et B doivent être linéairement indépendants

B \ A	< 90	< 180	< 270	< 360
< 90	Si A = 0 Projette B sur y Départ : 1	Projette B sur x Départ : 1	Projette B sur y Départ : 3 Si B = 0 Départ : 2	Projette B sur y Départ : 1 Si A = 270 Projette B sur x Départ : 2
< 180	Projette B sur x Départ : 3 Si A = 0 Projette B sur y Départ : 1	Si A = 90 Projette B sur x Départ : 3	Projette B sur y Départ : 3	Projette B sur x Départ : 4 Si B = 90 Départ : 1
< 270	Projette B sur y Départ : 2 Si B = 180 Départ : 3	Projette B sur y Départ : 4 Si A = 90 Projette B sur x Départ : 3	Si A = 180 Projette B sur y Départ : 4	Projette B sur x Départ : 4
< 360	Projette B sur y Départ : 2	Projette B sur y Départ 4 Si A = 90 Projette B sur x Départ : 1	Projette B sur x Départ : 2 Si A = 180 Projette B sur y Départ : 4	Si A = 270 Projette B sur x Départ : 2

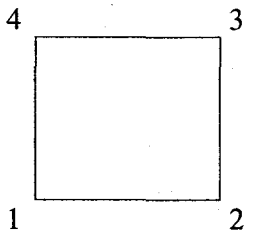
Avec la région :



6.5.2 Sommet de départ selon un vecteur d'itération et la projection d'un deuxième sur la normale du premier

proj B \ A	< 90	< 180	< 270	< 360
< 90	---	Départ : 1	---	Départ : 1 Si A = 270 Départ : 4
< 180	Départ : 2 Si A = 0 Départ : 1	---	Départ : 2	---
< 270	---	Départ : 3 Si A = 90 Départ : 2	---	Départ : 3
< 360	Départ : 4	---	Départ : 4 Si A = 180 Départ : 3	---

Avec la région :



6.6 Comparaison des librairies pour les régions

6.6.1 Comparaison de cinq librairies pour les régions

	CGAL	LEDA	VXL	GEOS	TerraLib
Géométrie 2D	√	√	√	√	√
Bounding box	√	√	√	√	√
PointInterieur	√	√	?	√	√
MultiPolygon			√	√	√
MultiLine				√	√
Géométrie 3D	√	√	√	Point3D	
Point	√	√	√	√	
Ligne, plan...	√	√	√		
Polygon	√		Boite3D		
Géométrie dD	√				
Licence	Open Source	5000-13000	Open Source	Open Source	Open Source
Notes	Utilise LEDA				

6.6.2 Tableau des fonctionnalités de GEOS et TerraLib

	Formes géométriques	**BoundingBox**	**Comparaisons entre formes géométriques**
GEOS 1.0	MultiLineString MultiPoint MultiPolygon LineString Point Coord2D Polygon Curve …	GetEnveloppe GetInternalEnveloppe	Equals test Disjoint test Touch test Intersects test Crosses test Within test Contains test Overlaps test Relate test …
TerraLib 2.0	Arc Cell CoutourLine Coord2D Line2D LinearRing Node Point Polygon … peuvent être en multi avec le pattern composite	Box	Equals test Disjoint test Touch test Intersects test Crosses test Within test Contains test Overlaps test Relate test Covers test Covered by test …

6.6.3 Documentation, support et exemples

	Documentation	Exemples	Support
GEOS 1.0	• Définitions de classes non commentées extraites avec Doxygen	• Possible de trouver des exemples dans la mailing-list	• Mailing-list de développement
TerraLib 2.0	• Définitions de classes commentées extraites avec Doxygen • Guide de référence pour programmeurs bientôt disponible	• Une dizaine de fichiers cpp d'exemples d'utilisation de la librairie	• Par email : terralib@dpi.inpe.br

EAU, TERRE ET ENVIRONNEMENT

Renaud le Boulleur de Courlon **du 14/02/03 au 14/06/03**

3ème année Hydraulique opt. : Eau et Environnement

DEA : STE

Maître de stage : Michel Leclerc

Application du projet SEGRI : étude des risques d'inondation de la rivière Chaudière

INP ENSEEIHT

Université du Québec
Institut national de la recherche scientifique

I. Index des figures et tableaux

II. Remerciements

Je tenait à remercier toutes les personnes qui m'ont aidé dans tout au long de ce stage, en me consacrant de leur temps. Tout d'abord mon maître de stage Michel Leclerc qui m'a accordé toute sa confiance pour mes travaux. Ensuite, parmi le personnel de l'I.N.R.S.Eau Terre et Environnement, Paul Boudreau et Yves Secretan qui m'ont permis de comprendre l'utilisation de Modeleur en répondant à mes nombreuses questions. Je tenais aussi particulièrement à remercier Simon Dubé pour son temps et sa patience quand il s'est agit de me fournir les données hydrologiques et de répondre à mes questions les concernant. Enfin tous les autres employés et stagiaires de l'I.N.R.S. qui par leur présence et leur bonne humeur ont permis de joindre l'utile à l'agréable.

III. Présentation de l'organisme d'accueil

1 Présentation de l'INRS

Voué à la recherche et à la formation de haut niveau, l'Institut national de la recherche scientifique (INRS) est présent là où les enjeux de société l'exigent. L'INRS a dernièrement repensé son organisation pour faire face aux défis du XXIe, en regroupant ses équipes scientifiques œuvrant dans des domaines complémentaires. En passant de huit à quatre centres de recherche, l'Institut optimise ses activités de recherche et de formation, en plus d'accroître sa capacité d'intervention dans des secteurs stratégiques pour l'avenir du Québec :

- Eau, Terre et Environnement
- Énergie, Matériaux et Télécommunications
- Santé humaine, animale et environnementale
- Urbanisation, Culture et Société

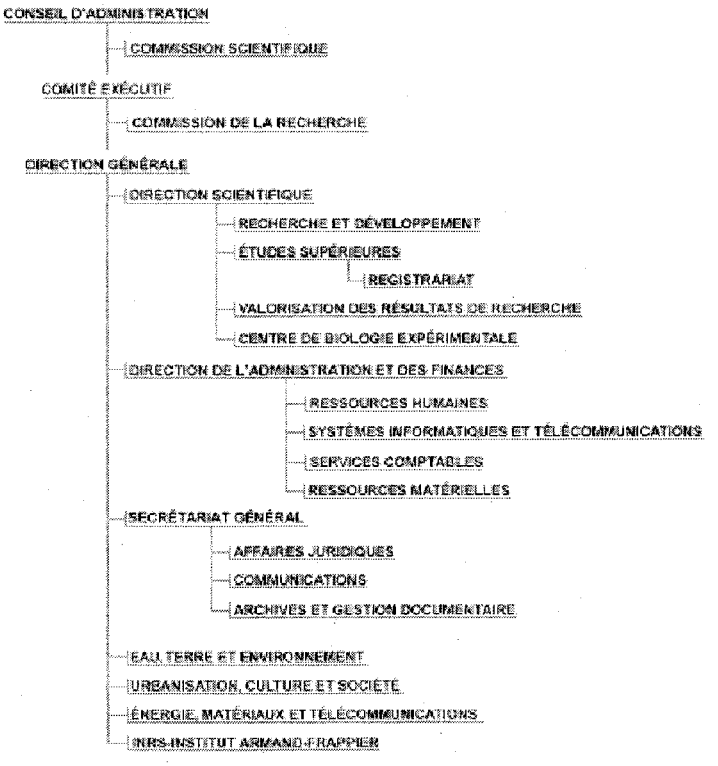


Figure 1 : Organigramme de l'INRS

2 L'INRS en quelques chiffres

L'Institut national de la recherche scientifique compte :

- 141 professeurs-chercheurs
- 147 professeurs et chercheurs invités
- 48 professeurs associés
- 572 étudiants dont :
  - 281 étudiants à la maîtrise
  - 194 étudiants au doctorat
  - 43 stagiaires post-doctoraux
  - 54 étudiants stagiaires
- 21 programmes d'études de cycles supérieures.
- 4 chaires de recherche et d'enseignement
- 7 chaires de recherche du Canada
- Plusieurs unités de recherche (laboratoires, observatoires, réseaux, groupes)
- Un budget de 72 millions de dollars
- Des fonds de recherche de plus de 31 millions de dollars

3 INRS-Eau, Terre & Environnement (dirigé par J.P. Villeneuve)

La science au service de l'eau et de l'environnement

a) La recherche

L'INRS-Eau, Terre & Environnement, par le biais de ses recherches en hydrologie(le centre abrite la seule chaire en hydrologie statistique au Canada :CRSNG/Hydro-Québec), en bio-géochimie ainsi qu'en assainissement et en développement de technologies environnementales, vise à assurer une utilisation rationnelle et durable de l'eau.

C'est dans une perspective multidisciplinaire que les professeurs-chercheurs du Centre entreprennent leurs recherches. Dans le domaine de l'hydrologie et de l'hydraulique, ils développent des méthodes d'analyse statistique et numérique, de modélisation, de télédétection et de géomatique appliquées aux écoulements. Ils s'intéressent aussi au comportement géochimique et à l'éco-toxicologie des polluants ainsi qu'à la dynamique des éléments nutritifs dans les écosystèmes. En matière d'assainissement, ils mettent au point de nouveaux procédés, entre autres, pour la valorisation des boues municipales et le traitement des rejets miniers, de façon à apporter des solutions concrètes aux problèmes de pollution.

b) Groupes de recherche

Chaire en hydrologie statistique

Groupe de recherche en aménagement et gestion intégrée des bassins versants (GIBSI)

Groupe de recherche en bio-géochimie

Groupe de recherche en éco-hydraulique

Groupe de recherche en hydrologie urbaine

Groupe de recherche en infrastructures urbaines

c) Les partenariats et collaborations

Afin d'apporter des solutions aux problèmes qui touchent la protection de l'environnement, l'INRS-Eau, Terre & Environnement participe activement à des projets de recherche réalisés de pair avec des ministères, des organismes publics et des entreprises. À titre d'exemple, il assure la direction d'un réseau de recherche canadien sur les métaux dans l'environnement, soutenu par le Conseil de recherches en sciences naturelles et en génie du Canada, qui réunit tant des scientifiques que des représentants gouvernementaux et industriels.

En plus de coopérer avec plusieurs universités canadiennes et étrangères, il collabore régulièrement avec les ministères québécois de l'Environnement, de l'Agriculture, des Pêcheries et de l'Alimentation du Québec, des Affaires municipales et de la Métropole, Hydro-Québec, la Société d'Énergie de la baie James, le Centre Saint-Laurent, Environnement Canada, la Communauté urbaine de Québec, la Ville de Québec, la Ville de Montréal, ainsi qu'avec des entreprises comme les Breuvages Nora, la Société d'électrolyse et de chimie d'Alcan, l'Industrielle de l'environnement, BPR Groupe-conseil, Roche ltée Groupe-conseil.

d) Les champs d'expertise scientifique

Par l'étude des processus et leur modélisation, les professeurs-chercheurs de l'INRS-Eau, Terre & Environnement ont mis au point de nouvelles méthodes, très performantes, pour l'analyse de données appliquées aux phénomènes hydrauliques, hydrologiques, limnologiques et hydro-géologiques. Forts de quelques décennies de recherche dans le domaine de l'eau et de l'environnement, ils détiennent une expertise appréciable en ces matières :

- Biodégradation des polluants environnementaux d'origine industrielle, agricole et municipale;
- bio géochimie et éco-toxicologie des métaux;
- Développement de logiciels de gestion des eaux;
- Études statistiques des événements hydrologiques;
- Gestion des aquifères et modélisation de l'évolution des contaminants;
- Modélisation hydrodynamique et modélisation hydrologique des écoulements;
- Télédétection appliquée au suivi du couvert nival;
- Traitement des rejets et valorisation des boues d'épuration.

e) Les services à la recherche et à la collectivité

Le centre INRS-Eau, Terre & Environnement possède des laboratoires spécialisés de microbiologie, de radio-isotopes, de microscopie, de spectrophotométrie, d'essais biologiques, d'analyses chimiques. Son expertise en chimie analytique environnementale est reconnue et recherchée tant par le milieu de la recherche gouvernementale que par l'entreprise privée. De même, le Centre est doté d'une solide expertise en matière de développement de logiciels de gestion des eaux, qui est fortement sollicitée ici et ailleurs dans le monde.

IV. Introduction

1 Présentation du projet SEGRI

Développement d'une méthodologie de réduction des risques d'inondations.

Le projet SEGRI vise à développer une méthodologie intégrée qui cherche d'abord :

- À établir le bilan des risques actuels d'inondations sur un cours d'eau;
- À élaborer une stratégie intégrée de réduction des risques qui se justifie par sa rentabilité économique, tout en respectant les critères de développement durable : respect du milieu naturel et équité entre les divers intervenants à risque du bassin.
- À gérer le risque résiduel par une concertation des intervenants et une mise à niveau continue des plans de mesure d'urgence.

Les données favorisant la prise de décision proviendront du logiciel modeleur dont le développement fait partie du projet.

Le projet a été découpé en 2 principaux volets :

- Recherche et développement
- Validation et transférabilité

La présente étude s'inscrit dans le second volet et sert à démontrer la faisabilité et la pertinence de l'approche par une application à un tronçon problématique de la rivière Chaudière : La municipalité de Sainte-Marie de Beauce.

- Les partenaires :
- La municipalité de Sainte–Marie de Beauce
- Certaines MRC (Municipalité Régionale de Conté) de la région Chaudière Appalaches,
- Le Cobaric (Comité de Bassin de la Rivière Chaudière)
- Trois firmes privées : Les consultants BPR Groupe conseil, GPR (utilisation du LASER aéroporté pour les problèmes d'inondation) et Hydrosoft Énergie (développement du marché latino-américain);
- Les ministères de l'environnement, de l'industrie et du commerce
- Les ministères de la Sécurité Civile du Québec et Protection Civile Canada

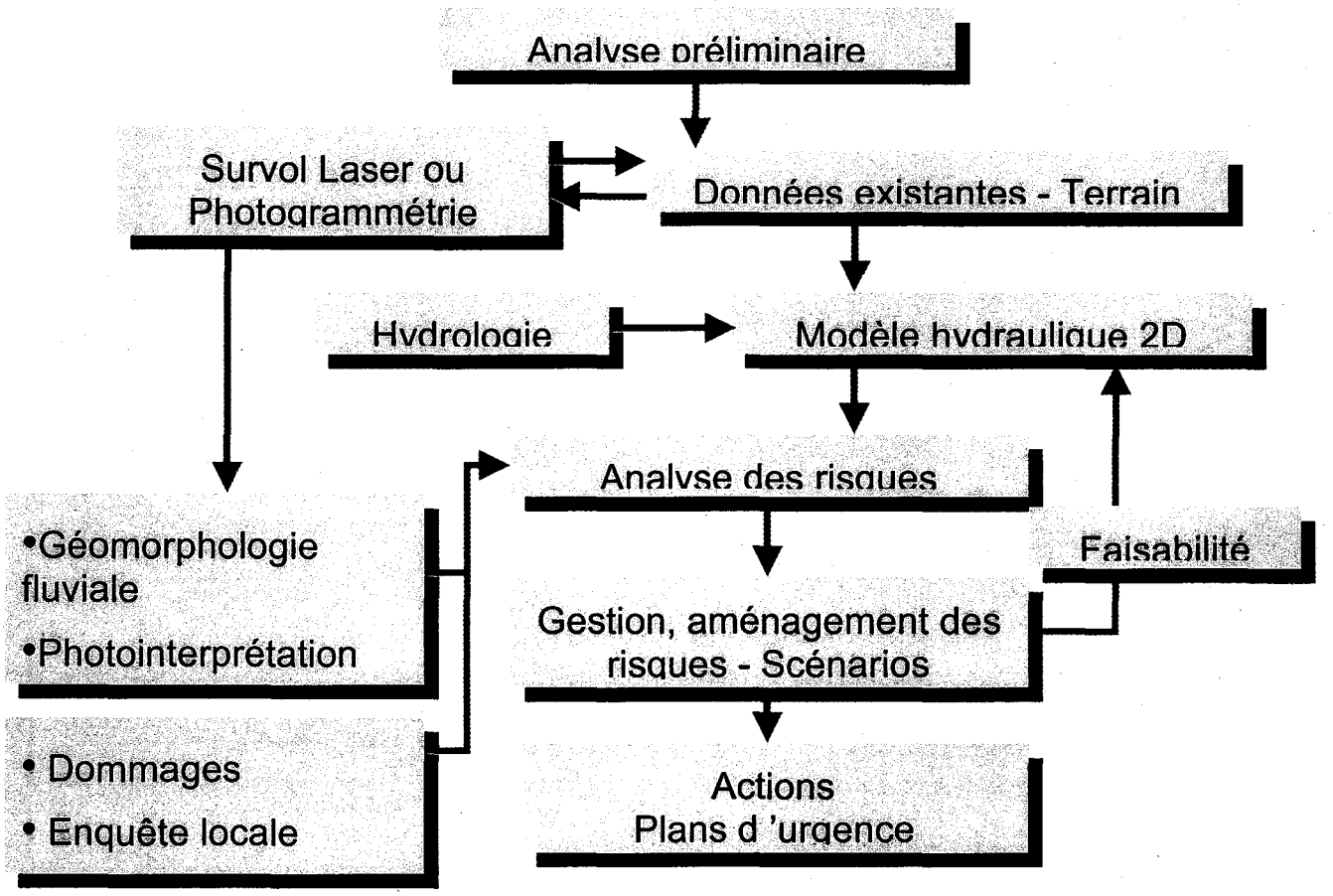


Figure 2: Méthodologie de la réduction des risques d'inondation utilisée dans le projet SEGRI

2 Le travail demandé

Cette étude commencera par une collecte de données provenant des différents partenaires. Ensuite viendra la conception d'un modèle numérique de terrain de la zone d'étude à partir des données collectées avec le logiciel Modeleur v1.0a07. Ensuite si l'avancement du projet le permet, la calibration et la validation du modèle par des simulation hydrodynamique avec le module hydrosim à partir des données hydrométriques collectées.

3 Présentation du logiciel Modeleur

Le logiciel Modeleur a été développé à l'INRS par Y.Secrétan & coll. [Y.Roy, Y.Secretan &coll. (2000)]. Il s'agit d'un logiciel permettant de construire des modèles numériques de terrain géoréférencés. Il peut prendre en compte en plus de l'altitude, le substrat, le frottement de Manning, les forces du vent, ainsi que les frottements associés au couvert de glace. Chacune de ces informations constitue une couche de donnée indépendante liée à un maillage propre construit par triangulation (algorithme de Delaunay) où importé. Les données sont alors interpolées linéairement sur les cotés des triangles. Le traitement d'une couche d'information se fait par le biais d'une partition qui délimite le domaine sur lequel on dispose de données et qui permet de diviser ce

domaine en sous-domaines associés à des jeux de données indépendants. Ce logiciel comporte également un mailleur pour construire le maillage hydrodynamique sur lequel sont projetées les différentes couches de données et d'ajouter les conditions aux limites.

Ensuite il permet de réaliser des simulations hydrodynamiques grâce au module de résolution numérique Hydrosim [M.Heniche, Y.Secrétan, M. Leclerc (2000)] qui discrétise les équations de St venant 2D stationnaire et eau peu profonde en utilisant la méthode des éléments finis sur des mailles triangulaires avec le modèle couvrant-découvrant utilisant des profondeurs positives et négatives [Y.Secretan & col 2000]. La résolution se fait avec l'algorithme GMRES (Generalized Minimum RESidual [Y.Secretan GMRES introduction]) et combiné à un calcul de la matrice de pré-conditionnement de type ILU (n) (Incomplete lower upper factorization level n).

4 Présentation du domaine d'étude

a) La rivière Chaudière et son bassin versant

Longue de 209 Km, la rivière Chaudière (dont la localisation est présentée sur la figure 2 et le profil en long sur la figure 3) prendre prend sa source au niveau du lac Mégantic et se jette dans le fleuve Saint-Laurent à Lévis en face de Québec. La superficie de son bassin versant est de 6 682 Km^2 à Lévis et de 5 570 Km^2 à Sainte-Marie, son débit moyen annuel est de 115 m^3/s. Celui-ci dépasse régulièrement les 1000 m^3/s lors des crues de printemps (le record est estimé à 3500m^3/s) et peut descendre sous les 10 m^3/s lors des fortes périodes d'étiages. Le temps de concentration de ce bassin versant est de l'ordre de la trentaine d'heure. Cette rivière est réputée depuis longtemps pour ses inondations qui touchent principalement la section moyenne, appelée section des eaux mortes. Cette zone située entre Saint-Georges et Scott-Jonction, présente une très faible pente de 0.1m/Km et un lit mineur peu encaissé provocant le ralentissement et l'étalement des écoulements de crues.

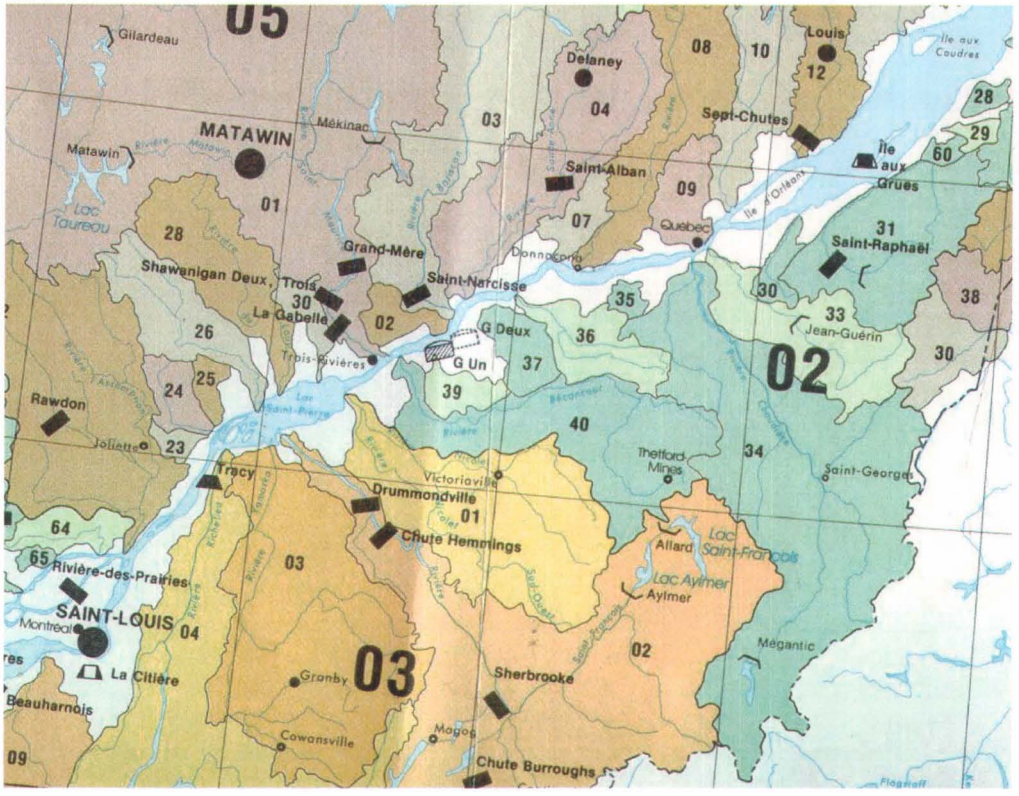


Figure 1 : La rivière Chaudière et son bassin versant

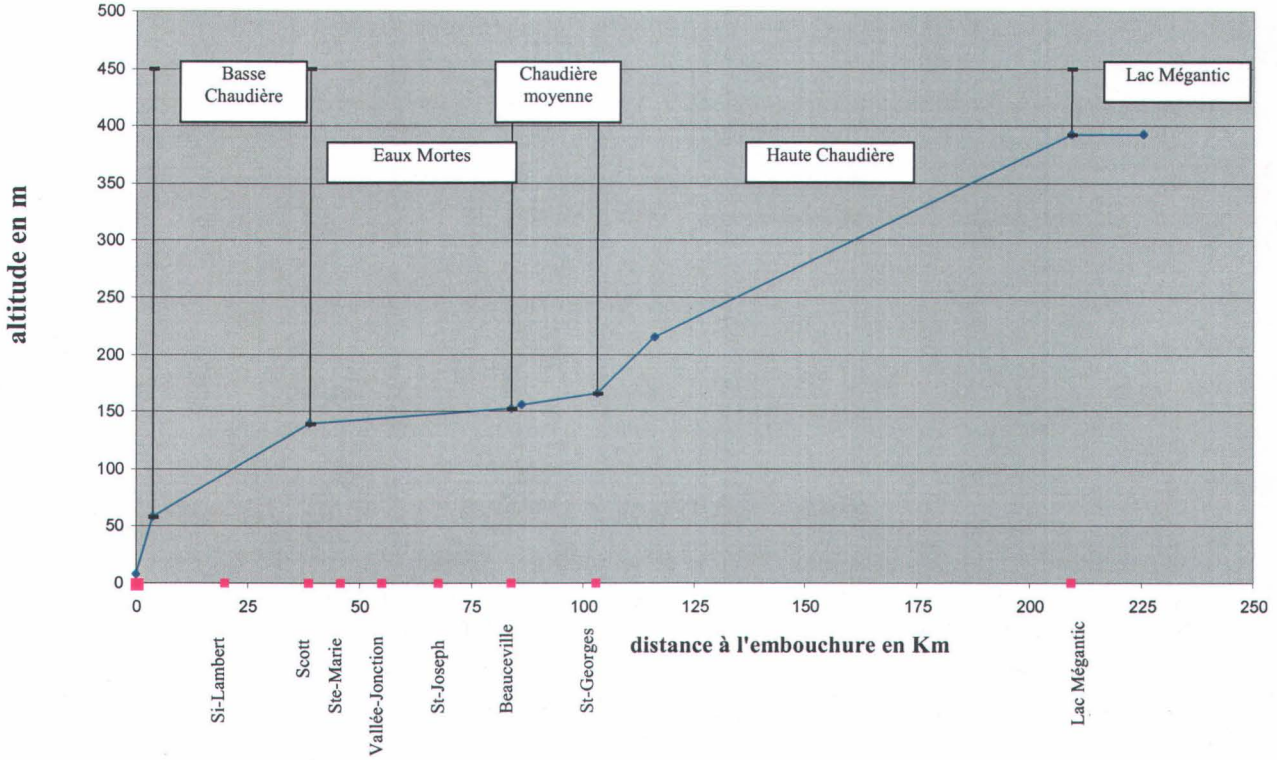


Figure 2 : Profil de la rivière chaudière

b) La municipalité de Sainte-Marie

Ste Marie de Beauce est située à une quarantaine de kilomètres au sud de Québec(voir figure 2) dans la région Chaudière Appalaches. Elle se trouve dans la zone des eaux mortes et est soumise par là même à de fréquentes crues. Comme de nombreuses municipalités, le développement de l'agglomération urbaine c'est fait au mépris des plaines inondables -d'après les cartes d'inondation du Ministère de l'environnement de1979, un tiers de la ville était construit entre les limites 20 ans et 100 ans- jusqu'à l'établissement d'une législation stricte [TECSULT(1993)],[J.Y.Goupil (1998)]. La municipalité souffre donc fréquemment des crues de la Chaudière. Le montant annuel des dégâts pour la municipalité de Sainte-Marie a été estimé par les groupes-conseils TECSULT et ROCHE à près de 1.8 millions de dollars Canadien en 1993 contre 2.5 million pour le tronçon des eaux-mortes. Les dégâts de la municipalité de Sainte-Marie représentent donc près de 70 % des dommages totaux .C'est ainsi qu'elle s'est trouvée être intéressée par le projet et fait l'objet de la partie application.

V. Conception d'un modèle numérique de terrain

Le modèle numérique de terrain est nécessaire à la mise en œuvre du modèle hydrodynamique lequel permet de lier le niveau de l'eau, et donc les hauteurs de submersion, au régime hydrologique. Il comprend différentes variables distribuées qui, pour les fins de la présente étude se limitent à l'élévation et aux facteurs de résistance à l'écoulement. Accessoirement, les principaux bâtiments pouvant faire l'objet de dommages importants dans la zone d'inondation seront représentés. Les données à partir desquelles a été construit le modèle numérique de terrain proviennent de plusieurs sources. Les prochaines sections serviront à en décrire la provenance, la structure et les traitements requis pour les insérer de façon homogène dans un MNT à deux dimensions.

1 Les sections fournies par le ministère de l'environnement du Québec

La bathymétrie (lit mineur recouvert par le débit plein-bord) a été construite en interpolant les sections fournies par le Ministère de l'environnement relevées en 2002 sous la forme d'une géométrie HEC-RAS géoréférencée établie pour une application à une dimension. Dans ce cas, seule la distance entre les sections est utile et la sinuosité du cours d'eau de même que la position relative de ses berges sont perdues. La conversion en deux dimensions est requise pour l'application d'un modèle 2D tel qu'Hydrosim. Pour réaliser l'interpolation des sections et placer celles-ci de manière à décrire correctement le lit mineur, il a fallu écrire un programme en FORTRAN (le code est inséré en annexe). Celui-ci nécessitait comme données d'entrée un fichier contenant les sections connues ainsi que la position géoréférencée des points délimitants le lit mineur de la rivière. Les coordonnées de ces points ont été obtenues en traçant les sections le long du lit mineur sous Modeleur puis en plaçant des points sur les intersections entre les rives gauches et droites (obtenues par numérisation d'une carte d'inondation du ministère voir chap. III.3.) et chacune des sections, et enfin en exportant le fichier correspondant. À partir de quoi, le code crée un fichier de données brutes de topographie (i.e. : semis de points contenant les altitudes des points), un fichier de maillage structuré qui permettait de forcer Modeleur à interpoler entre les différentes sections créées de la bonne façon (c'est à dire d'éviter de relier un point appartenant au lit mineur avec un point de la plaine inondable), ainsi qu'un fichier de champs scalaire de topographie pour attribuer une altitude à chacun des nœuds du maillage.

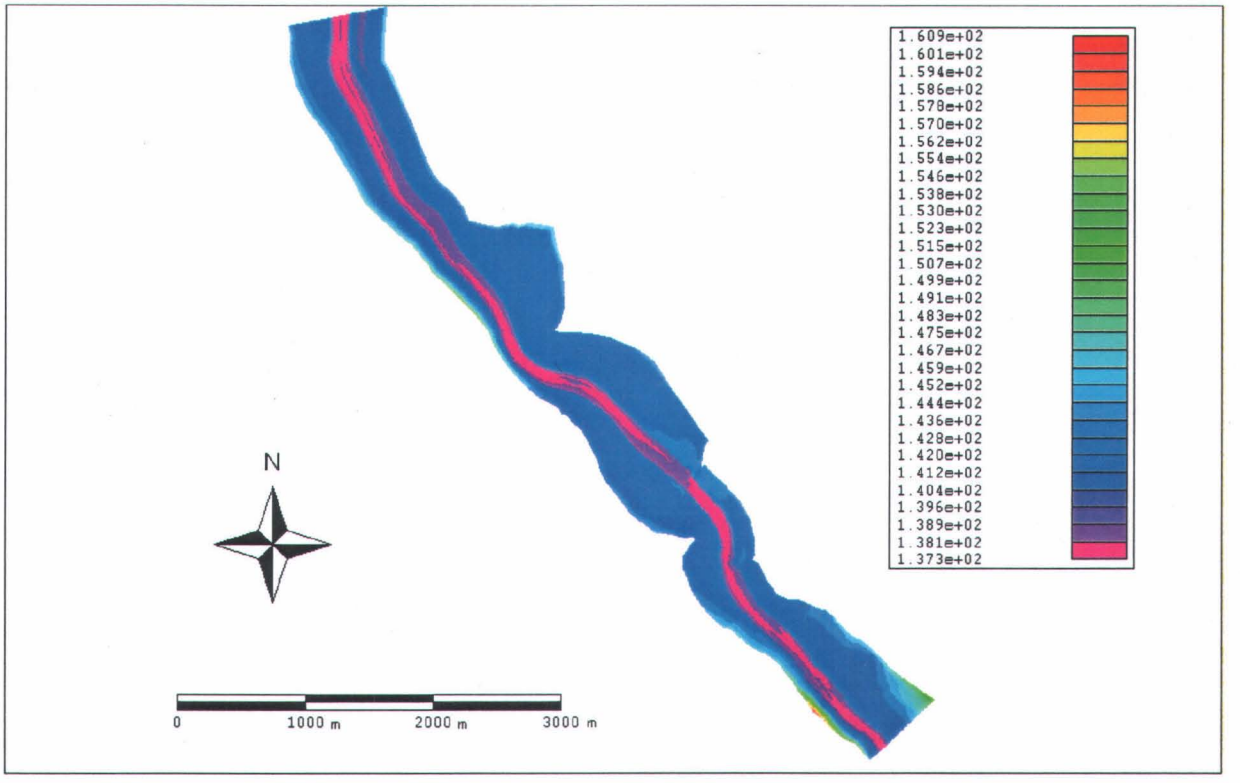


Figure 5 : Topographie obtenue à partir de l'interpolation des sections

2 La plaine inondable

Les données nous ont été fournies par une société locale d'arpenteur-géomètres (Ecce Terra Inc.), consultant des Services techniques de la Municipalité de Sainte-Marie sous format AUTOCAD™ (*.DXF). Le fichier comprenait :

- semis de points de mesure (50 000 points)
- lignes de rupture de pente (contenant plusieurs points de même altitude reliés)

Les données provenaient d'une campagne de terrain de photographies aériennes (ortho-photographies au 1/20.000, puis traitement par photogrammétrie) commandée par la ville de Sainte-Marie ainsi que d'autres campagnes d'arpentage pour la compléter. D'après l'arpenteur de cette société, la précision horizontale serait de 15 cm et la précision verticale de 20 cm. Ces données ont été importées dans Modeleur sous format de données brutes de topographie (i.e. : x y z erreur) et donc interpolées linéairement avec la méthode de Delaunay.

La société nous a également fourni un exemplaire de l'ortho-photographies nous permettant entre autres de distinguer et définir les différentes zones d'utilisation du sol (zones forestière, champêtres ou urbaine) éventuellement converties en rugosité effective du terrain, ainsi que de vérifier que les interpolations de Modeleur n'introduisent pas d'incohérence. Cette photographies a notamment permis de localiser un endroit où l'algorithme de triangulation avait relié artificiellement les deux berges d'un petit cours d'eau (la rivière Chassée) situé juste en aval de Sainte-Marie. Ce cours d'eau était soupçonné de constituer un accès pour l'eau vers le centre ville en cas de crue moyenne à forte.

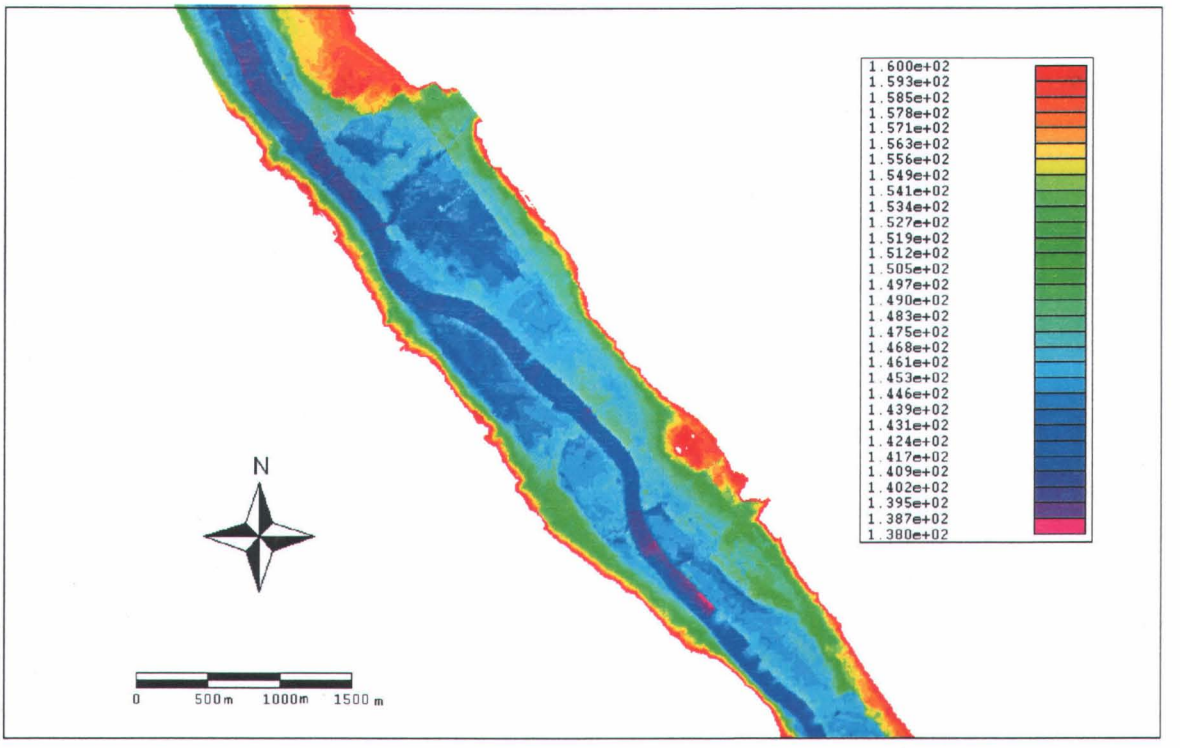


Figure 6 : Topographie obtenue à partir du fichier AUTOCAD™ (*.dxf) de Ecce Terra Inc.

3 Données complémentaires

Pour délimiter le lit mineur de la rivière, les berges ont été numérisées sur une carte d'inondation géoréférencée du ministère datant de 1979 au 1/20.000, sur laquelle étaient également disponibles les limites d'inondation pour des crues de périodes de retour de 20 et 100 ans. À partir de cette carte, les plus grands bâtiments de la municipalité ont également été localisés. Les numérisations de la carte ont été réalisées grâce à une table à numériser Altec AC30 modèle standard et au logiciel Atlas Draw.

Le ministère de l'environnement du Québec nous a également fourni les notes prises par les techniciens qui ont réalisé les sections en 2002 sur lesquelles figurent des informations sur les substrats au niveau de chaque section soit une répartition des substrats traduit en pourcentage de tel substrat (sable, gravier, caillou etc.) Ces données ont ensuite été interpolées linéairement le long du lit mineur sous Modeleur.

De plus à partir du fichier AUTOCAD™ fourni par l'entreprise Ecce Terra, les lignes de rupture de pente ont été importées séparément pour permettre une meilleure visualisation du site et des zones inondées.(Ces lignes apparaissent sur les représentations des résultats)

Ces données seront probablement complétées (et améliorées) par une campagne de balayage au laser aéroporté dans les prochains mois (il s'agit d'une commande du ministère de l'environnement du Québec) ce qui permettra de corriger les défauts du modèle et de tester sa sensibilité à la précision de la topographie.

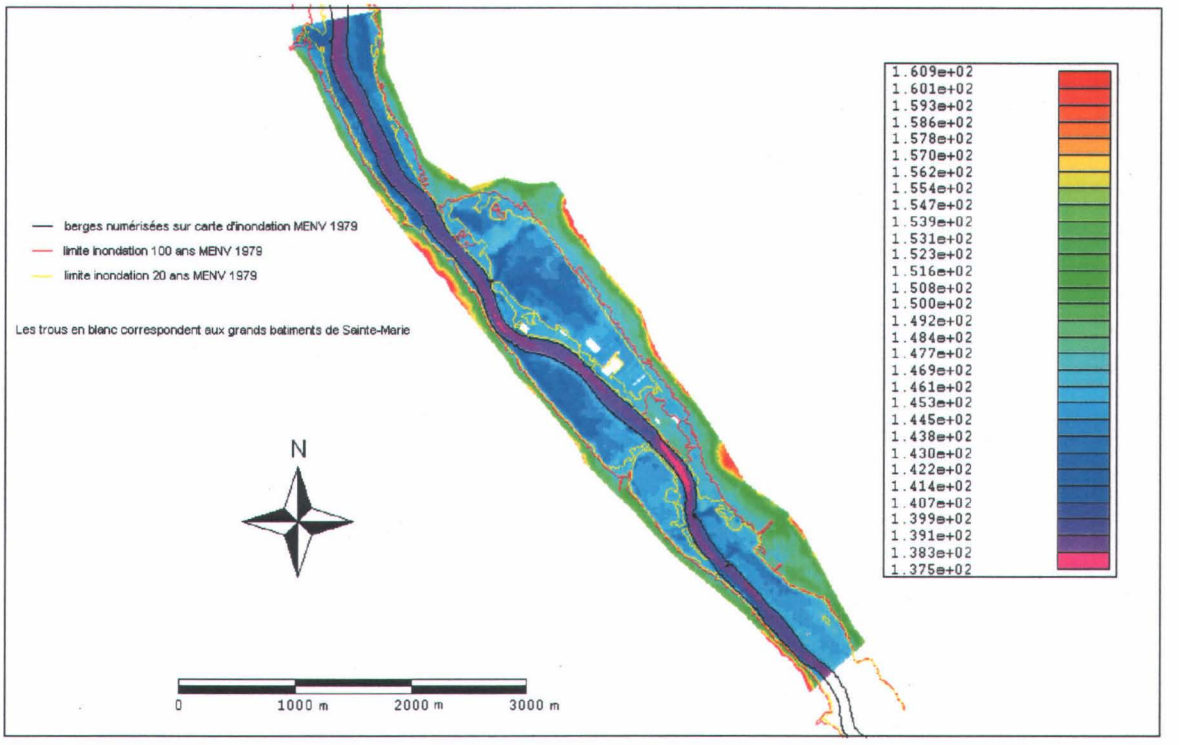


Figure 7 : Topographie de la plaine inondable (relevé Ecce Terra Inc.), section et lignes numérisées (MENV 1979)

VI. Hydrologie

1 Hydrologie historique de la rivière Chaudière à Sainte-Marie

Les données hydrométriques sont d'abord constituées des débits moyens journaliers à Saint-Lambert (entre les années 1925 et 2003; N=78) une section située à 28 km en aval de Sainte-Marie. Pour ses calculs statistiques, le Centre d'Expertise Hydrique (CEH) du ministère a utilisé une loi log normale à trois paramètres. Les valeurs obtenues sont listées dans le tableau ci-dessous :

Récurrence (ans)	2	3	5	10	20	50	100	200	1000	2000	10000
Débit (m^3/s)	1208	1370	1534	1722	1887	2085	2223	2356	2644	2763	3030

Tableau 1: Débits de crue pour les récurrences caractéristiques

La liste des crues de la rivière Chaudière de puis 1917 a été portée en annexe.

2 Relation niveau-débit pour les fins de la modélisation

Les débits de crues utilisés sont ceux calculés par le Ministère de l'Environnement en 2002. Pour ses calculs statistiques, le Centre d'Expertise hydrique du Ministère a utilisé une loi log normale à trois paramètres.

Les données hydrométriques sont constituées des mesures journalières de débit à Saint-Lambert (entre les années 1925 et 2003) et des mesures journalières de hauteur d'eau à la station pont/route de Sainte-Marie; elles ont été fournies par le centre d'expertise hydrique et ont permis l'établissement d'une relation niveau-débit à Sainte-Marie. Pour ce faire, les débits mesurés à Saint-Lambert, ont été rapportés à Sainte-Marie grâce à l'hypothèse de linéarité entre les débit et la superficie des bassins versants à différent points de la rivière.

Parmi toutes les données récupérées, seules les données postérieures au 9/09/1999 sont utilisées pour estimer la relation niveau-débit car la constante géodésique à ajouter aux valeurs de niveau d'eau a changé plusieurs fois au cours des années antérieures à cette date ce qui nous créait un problème d'homogénéité. De plus, la rivière n'a pas le même comportement pour les crues de printemps et pour les crues éclairs d'été pour lesquelles la transposition des débits de Saint-Lambert à Sainte-Marie peut s'avérer moins fiable (a t-il plu sur tout le bassin versant?). C'est pourquoi parmi les données postérieures au 9/9/1999, seules les données correspondant à des crues de printemps ont été conservées. Une relation niveau-débit a été estimée par une régression polynomiale sous Microsoft Excel la relation obtenue est :

$$Q=38.51.H^2-10758.H+751305$$

Elle n'a pu être validée qu'avec les deux événements pour lesquels les débits ont été jaugés à Sainte-Marie en même temps que les niveaux ont été mesurés soit les événements des 31/03/2003 et 2/05/2003.

Cette relation a servi à calculer le débit pour l'événement du 2 juillet 2002 qui n'avait pas été jaugé, ceci afin de disposer d'un évènement à un plus fort débit ($1291 m^3/s$) pouvant servir de seconde validation. Ensuite, elle pourra aussi servir pour calculer les niveaux correspondant aux débits de crues, calculés par le ministère de l'environnement du Québec, pour les périodes de retour caractéristiques soit 2 5 10 20 et 100 ans.

VII. Simulation hydraulique : étude de la situation actuelle

1 Le maillage hydrodynamique

Le maillage hydrodynamique de type éléments finis sur lequel sont projetées les différentes couches de données (topographie, champs de substrat et de frottement *n* de Manning, conditions aux limites et état d'initialisation) est composé de 59 876 nœuds et de 29 592 éléments, il est présenté sur la figure 7. Afin d'optimiser le nombre d'éléments et le temps de calcul, le maillage est partitionné (partition Modeleur) en plusieurs sous-domaines plus ou moins raffinés en densité de maille: la taille typique des mailles varie de 15 m pour le lit mineur à 45 m pour les limites extérieures du domaine et les étendues champêtres de la plaine inondable où la connaissance des vitesses est superflue vis à vis du niveau d'eau lequel n'a pas besoin de mailles très fines pour converger vers une solution.

Sur ce maillage, les bâtiments les plus importants surtout ceux de nature industrielle ont été inclus (trous dans le maillage: l'usine de Pâtisseries Vachon, la confiserie, l'Aréna, la M.R.C., Bonneville ainsi qu'un bâtiment repéré sur l'ortho-photographie situé le long de la rue Saint-Jean et pouvant influencer significativement l'écoulement. La formulation du couvrant/découvrant dans Modeleur impose de ne pas mailler les zones que l'on considère comme sèches tout au long des simulations et présentant de fortes pentes (un mur ou une digue par exemple) si l'on veut un réel contournement de la zone par l'écoulement. En effet, si la zone est maillée, l'écoulement va rencontrer un accroissement brusque du frottement de Manning jusqu'à la valeur choisie par l'utilisateur, appelée « pénalisation du coefficient de Manning » et généralement comprise entre 6 et 10. Il y subsiste donc un écoulement qui, sans être significatif, demeure réel. De plus, ne pas mailler les zones occupées par les grands bâtiments de Sainte-Marie permettait un meilleur repérage lors de la visualisation du domaine .

Ensuite, les limites des sous-domaines entourant le lit mineur se sont avérées primordiales pour représenter correctement la topographie. En effet, l'interpolation linéaire sur les éléments du maillage peut avoir tendance à lisser le relief si les nœuds n'épousent pas correctement les ruptures de pentes, notamment au sommet des berges et au pied des talus puisque les éléments dans cette zone sont de l'ordre de la quinzaine de mètres. Ainsi, si les sommets des éléments sont trop éloignés, il peut en découler une variation de superficie de l'ordre de la quinzaine de mètres carrés ou plus dans la section verticale d'écoulement. L'importance de ce phénomène s'est fait sentir au moment de la calibration du modèle : en repositionnant certains nœuds du maillage de façon minimiser cette erreur, le niveau d'eau s'est abaissé d'une dizaine de centimètres. Il a fallu également faire attention aux limites des sous-domaines de la partition de topographie (délimitant les zones renseignées par différents jeux de données) pour arriver à modéliser la topographie du site le plus précisément possible avec le moins d'éléments possibles.

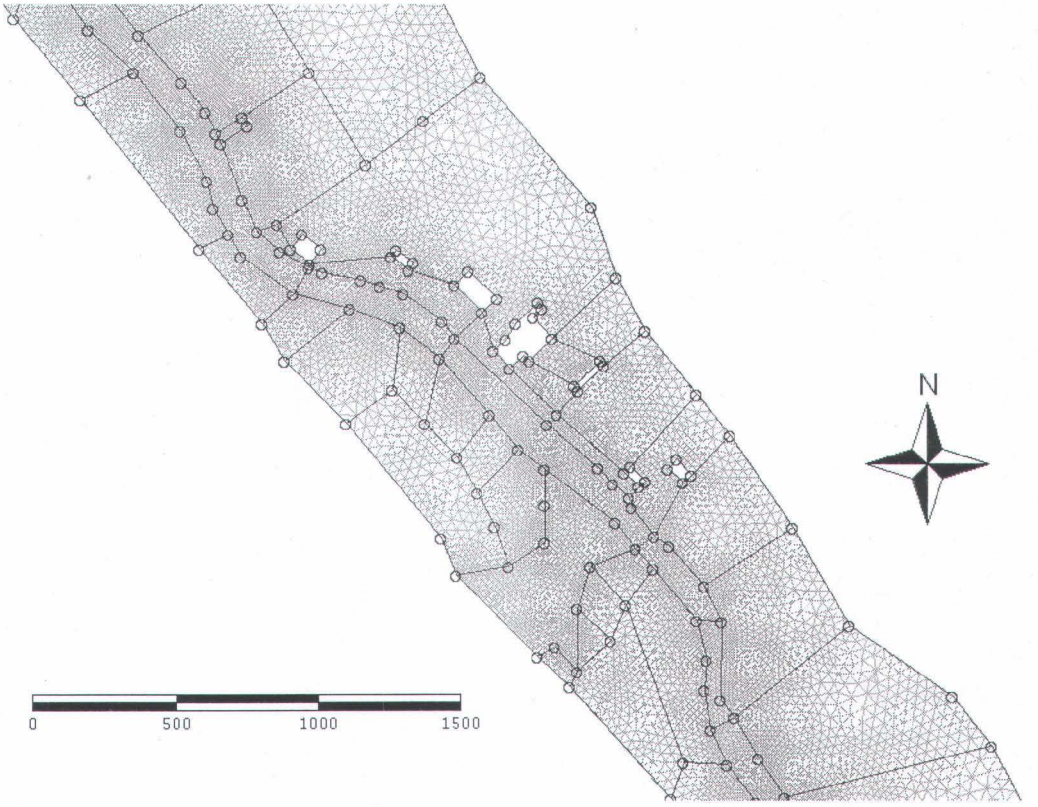


Figure 8 : Aperçu du maillage hydrodynamique et de la partition de maillage ayant servi à le créer

2 Calibration /validation

a) Événements de référence

La détermination des conditions aux limites et d'initialisation pour le calibrage et pour la validation du modèle a été faite à partir des évènements du 2 mai 2003 et du 31 mars 2003 pour lesquels la hauteur d'eau a été mesurée sur plusieurs sites, entre Saint-Lambert et Saint-Georges,. Les

débits correspondants, respectivement de 437.9 et 762 m^3/s, ont été jaugés au pont-route de Sainte-Marie (ce dernier débit s'est avéré incorrect et a été corrigé pour prendre la nouvelle valeur de 706 m^3/s).

L'événement de crue du 2 juillet 2002 a également été utilisé. Le débit n'ayant pas fait l'objet de jaugeage à Sainte-Marie, sa valeur a dû être estimée. Deux méthodes ont été utilisées : la relation niveau-débit déjà établie au pont-route, et une transposition de la valeur mesurée à Saint-Lambert en tenant compte du décalage lié au temps de parcours de l'onde et de la taille du bassin versant. Dans le deuxième cas, le débit est donc dérivé indirectement des mesures de débit prises aux 15 minutes à la station de Saint-Lambert qui sont bien sûr décalées dans le temps. Pour estimer ce décalage, la célérité de l'onde de crue a été estimée par la célérité d'une onde de gravité : $c=\sqrt{gH}$ (avec H = valeur moyenne de la profondeur du lit mineur $\approx$ 6m). Ensuite, connaissant la distance de Sainte-Marie à Saint-Lambert, on obtient le temps de décalage entre la prise de mesure de la hauteur d'eau et celle du débit et remonter dans la série pour retrouver la valeur recherchée. Enfin, le débit trouvé pour la station de Saint-Lambert est réduit à Sainte-Marie par l'application d'une règle de trois grâce au rapport de taille des bassins versants. Nous avons ainsi obtenu 1262 m^3/s par cette dernière méthode contre 1291 m^3/s par la relation niveau-débit établie, soit une différence relative de 2.2% considérée comme une preuve suffisante de concordance.

b) Patron de conditions aux limites et critère d'ajustement

Les choix possibles pour les conditions aux limites sous Hydrosim sont : imposer le débit à l'amont et la hauteur en aval, ou bien imposer la hauteur à l'amont et à l'aval. Les simulations qui ont servi à calibrer et à valider le modèle ont été réalisées avec des conditions aux limites niveau-débit, la validation s'effectuant par comparaison du niveau d'eau au pont/route

Les résultats étaient jugés corrects quand les résidus (de la résolution) étaient en dessous de 10^{-3} et quand le champ de vitesse était cohérent, c'est à dire que l'algorithme couvrant-découvrant n'introduisait pas des vitesses locales énormes pour les très faibles profondeurs. Ce phénomène s'explique du fait que le débit spécifique a été choisi comme inconnue et que les vitesses sont obtenues en le divisant par la profondeur qui est très faible à la limite du couvrant découvrant.

c) Paramètres de calibration

Les paramètres de calibration sont les paramètres de résolution numérique : nombre de Pecklet (représentant dans ce modèle la viscosité numérique applicable à l'opérateur de diffusion), la pénalité de Manning (pénalité de frottement de Manning pour les zones sèches) ainsi que le champ de frottement (n de Manning) dans le lit mineur et dans la plaine inondable, bien que ce dernier ait une influence moindre sur le niveau d'eau.

Finalement le champ de frottement a été réduit de 0.005 sur l'ensemble du domaine par rapport aux valeurs initialement fixées, le Pecklet a été fixé à 1 et la pénalité de Manning à 8. De plus, le paramètre de hauteur minimum de submersion a été utilisé pour remédier aux difficultés de convergence.

d) Résultats de calibration

L'évènement du 2 mai 2003 a servi de référence pour la calibration. Le modèle s'est très bien comporté pour les évènements du 2 juillet 2002 et du 31 mars 2003. Les conditions aux limites ainsi que les résultats des simulations sont regroupés dans le tableau ci-dessous :

Tableau 2 : Récapitulatif des conditions aux limites et des résultats des simulations

Date de l'évènement	Débit imposé (m^3/s)	Niveau aval imposé (m)	Niveau au pont simulé (m)	Niveau au pont mesuré (m)	Différence
2/05/03	437.9	142.18	143.06	143.04	0.02
31/03/03	706	143.03	144.05	144.01	0.04
2/07/02	1291	144.42	145.54	145.52	0.02

3 Aperçu des résultats

a) Profils du niveau d'eau

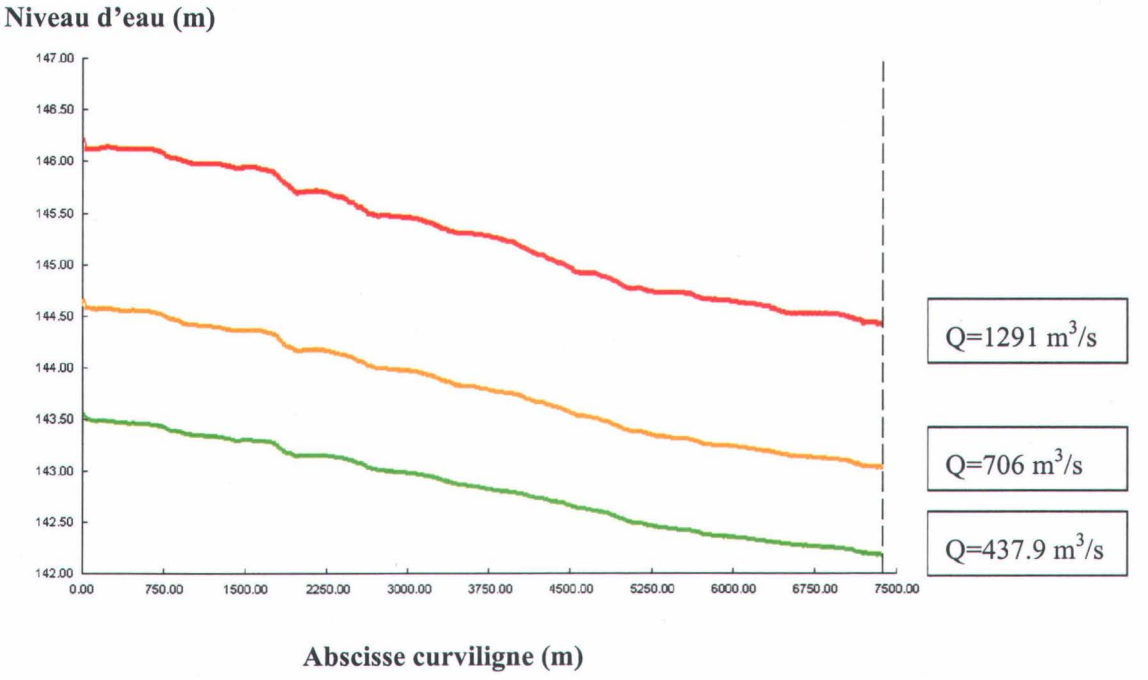


Figure 9 : Profils des niveaux d'eau simulés

b) Profondeur de submersion

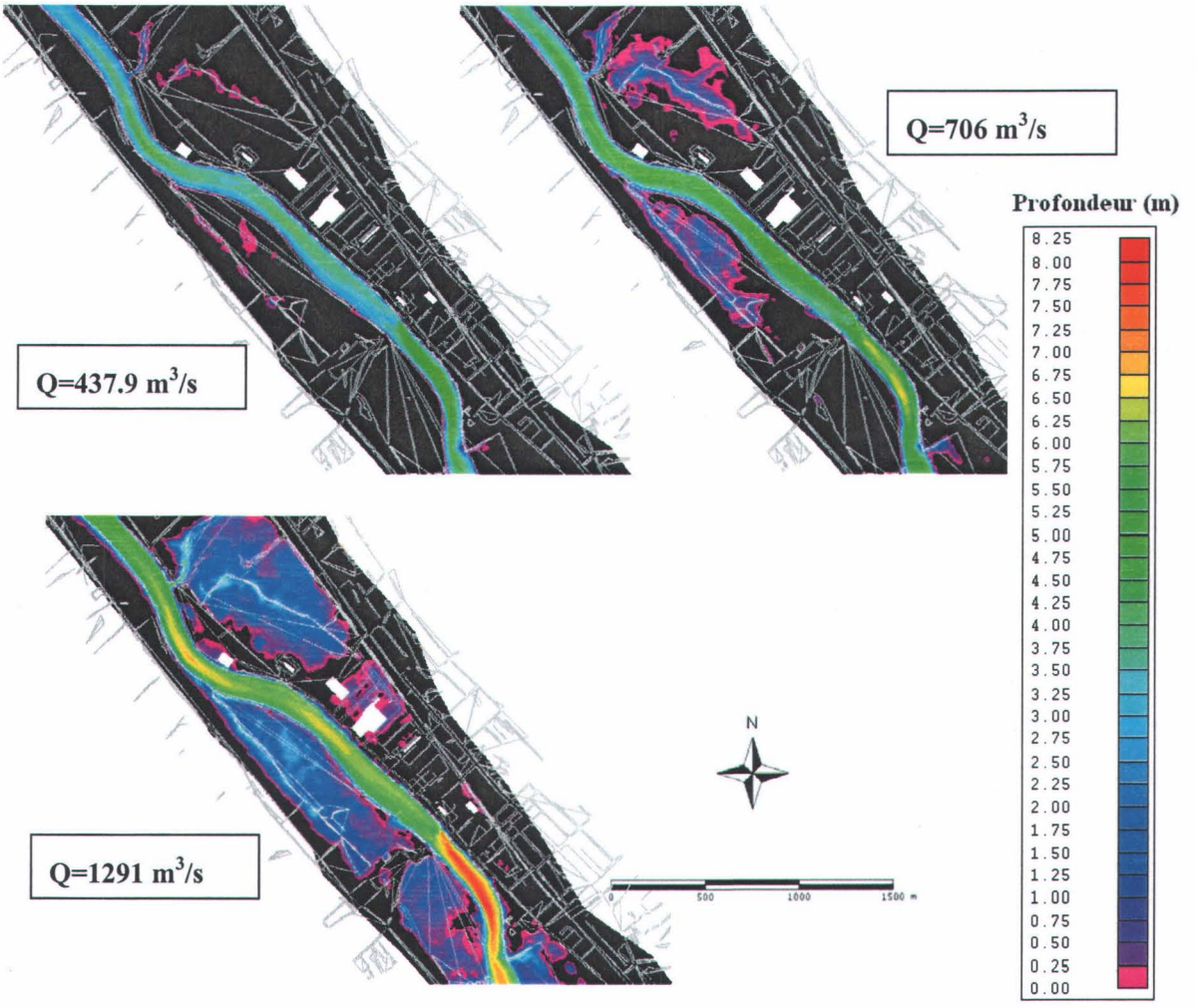


Figure 10 : Profondeurs de submersion pour les trois évènements simulés

c) Critique des résultats

Même si ces résultats semblent encourageants, il faut cependant garder à l'esprit les différentes approximations nécessaires pour combler le manque de données fiables. Premièrement, la topographie du lit mineur dans le modèle numérique de terrain n'est qu'une approximation par interpolation linéaire le long du lit mineur de seize sections transversales pour un tronçon de 7.3 km de longueur. À cela s'ajoute la même approximation en ce qui concerne le substrat. Et enfin le manque de données hydrométriques précises pour la calibration et la validation. En effet, la station de mesure au pont-route de Sainte-Marie de dispose pas d'un seuil, la relation niveau-débit n'est donc pas univoque ce qui explique également les différences de débit pour un même niveau (voir 3.a.ii). Ces résultats ne seront donc vraiment validés que lorsque les données précitées seront réactualisées (après une campagne de terrain pour la bathymétrie et la campagne de balayage par laser aéroporté pour la plaine inondable.

VIII. Les suites du projet

Bien que les éléments mentionnés ci-après ne fassent pas partie du mandat reçu pour le stage, nous avons jugé utile de lever le voile sur ces aspects de la méthodologie applicable en analyse de risques dans une perspective de réduction de ceux-ci. Ainsi, nous allons brièvement aborder la notion de risque, son estimation dans les conditions actuelles (scénario du *statu quo*), l'élaboration et l'analyse de *scenarii* alternatifs permettant de les réduire, et l'analyse de faisabilité permettant d'en évaluer la pertinence par rapport aux critères économiques, sociaux et environnementaux.

1 L'analyse de risque

L'analyse de risque doit prendre en compte deux facteurs indépendants : l'aléa représenté par les champs de submersion des événements de crues statistiques de référence et leur probabilité, et la vulnérabilité de la zone inondable comprenant tous les éléments susceptibles de subir un préjudice ou un dommage lors de tels événements Cette analyse permet d'établir et de comparer les coûts du *statu quo* ou des *scenarii* alternatifs. Le risque comprend les éléments reliées à la sécurité des personnes et des coûts matériels liés aux dommages aux biens, et aux manques à gagner. Les coûts du risque sont établis sur une base moyenne annuelle en tenant compte de leur probabilité; ils comprennent notamment :

- les coûts en regard des destructions des bien immobiliers privés et publics;
- le manque à gagner fiscal de la municipalité (taxe) dû à la dévaluation du prix des propriétés à cause de l'inondabilité des terrains.
- les coûts des possibles interventions.

2 Scenarii alternatifs

On désigne ici les solutions possibles pour limiter ou éliminer l'impact des crues, qu'il s'agisse du contrôle hydrologique ou hydraulique. Ces configurations font l'objet de simulations et de comparaisons avec la configuration actuelle. Leur efficacité est d'abord mesurée à partir des niveaux d'eau obtenus ..

3 Faisabilité des scenarii alternatifs et recommandations

Au-delà de l'efficacité technique , la faisabilité des *scenarii* alternatifs recoupe divers critères dont la rentabilité sur le plan économique, la durabilité sur le plan de la dynamique sédimentaire, l'acceptabilité des solutions proposées sur le plan social (ex : relocalisation de propriétés), le respect des composantes environnementales de la plaine d'inondation surtout si on y retrouve des milieux humides participant activement à l'écologie du milieu en période d'inondation.

4 Étude d'impact des aménagements humains sur la configuration des berges

Cette étude doit permettre de comprendre l'évolution cumulative que l'anthropisation de la rivière a occasionnée sur la morphologie de son lit et son impact sur les hauteurs d'eau des crues. Cette démarche vise à tirer des leçons de comportements non-recommandables à l'égard du cours d'eau (les empiètements, les implantations indues, notamment) et d'émettre des recommandations sur la protection des berges et du littoral de la rivière. Un tel exercice vise également à sensibiliser les populations au phénomène du risque dans leur municipalité

IX. Conclusion

Malgré le manque de données fiables et validées, les résultats de l'étude sont encourageants car le modèle numérique de terrain obtenu avec les données actuelles a pu être validé par les simulations hydrodynamiques. Les objectifs du stage ont donc été remplis.

La collecte de donnée a montré que de telles études ne peuvent encore êtres menées partout car les données de topographie sont manquantes dans de nombreux secteurs, d'où l'objectif du ministère de l'environnement du Québec de caractériser un grand nombre de zones inondables urbaines (plus de 200 tronçons) avec des séries de campagnes de balayage au laser aéroporté dont la précision sur la topographie est typiquement de ±10 cm ce qui devrait régler le problème des données sur les plaines inondables.

Pour la bathymétrie par contre, c'est la partie comparative de la suite de l'étude qui devrait permettre d'en savoir plus, c'est à dire de quantifier les erreurs dues au manque de précision. Cette étape devrait permettre d'évaluer plus correctement le gain en précision par rapport au coût des données. Est-il nécessaire de réaliser une campagne de terrain avec un échosondeur pour les relevés bathymétriques ou quelques sections bien positionnées suffisent-elles?

Dans un premier temps, cette recherche méthodologique devrait permettre une meilleure gestion des cours d'eau pour la protection des agglomérations contre les inondations et par la suite la protection des cours d'eau et de leur littoral.

En ce qui concerne le logiciel Modeleur, son apprentissage ajouté à mon expérience d'autres codes de calcul comme FLUENT, m'a permis de formuler quelques conseils sur ce qui pourrait être amélioré pour la prochaine version qui est en préparation. Le cas de Sainte-Marie a notamment mis en évidence une certaine fragilité du code dans les zones où la plaine inondable a une très faible pente. Le mode de résolution du couvrant-découvrant tel que programmé semble avoir ici trouvé une limite se traduisant par un manque de robustesse lors de la résolution

Ce stage m'a permis de compléter ma formation dans le domaine de la modélisation en y ajoutant un volet sur les modèles numériques de terrain et en parfaisant ma compréhension sur les codes de calcul en hydraulique. Il m'a également permis d'approfondir ma culture en sciences de l'environnement grâce à l'étude du système québécois et du point de vue de leurs chercheurs notamment au travers d'un colloque sur la modélisation d'habitat et sur la gestion des débits réservés auquel j'ai eu la chance de participer.

Tout ceci dans un cadre de travail très enrichissant puisqu'on m'a laissé le soin de gérer la partie du projet qui m'a été confiée à ma guise tout en restant à ma disposition pour répondre à mes questions.

Annexes

1 Programme d'interpolation en fortran

```
implicit none

real*8 x0,y0,z0,x,y,z,x1,x2,x3,x4,y1,y2,y3,y4
real*8 alpha,xlim
real*8 deltax12,deltax23,deltax34
real*8 deltay12,deltay23,deltay34
real*8 xn,yn,zn
real*8 xn1,yn1,xn2,yn2,xn3,yn3,xn4,yn4
real*8 dm,dtot,beta
real*8 long_riv_dr_ind1,long_riv_dr_ind2
real*8 long_riv_ga_ind1,long_riv_ga_ind2
real*8 long_riv_dr,long_riv_ga
real*8 ddt,dgt,ddn,dgn,gamma
integer i,j,k,l,dim_x,dim_y,dim_z
integer nbr_section,nbr_noeud,ng,nl,nd,n
integer nbr_point,nbr_element,num_elem,numero_noeud
integer sommet_11,sommet_12,sommet_13
integer sommet_21,sommet_22,sommet_23
integer choix,nbr_groupe
integer ind1,ind2,cont,ii,jj
integer nbr_nouv_sect,nb_tot_sect,somme_nouv_sect
parameter (dim_x=250)
parameter (dim_y=150)
parameter (dim_z=30)
dimension x(dim_y,dim_x),y(dim_y,dim_x),z(dim_y,dim_x)
dimension x0(dim_y,dim_x),y0(dim_y,dim_x)
dimension x1(dim_y),y1(dim_y),x2(dim_y),y2(dim_y),x3(dim_y)
dimension y3(dim_y),x4(dim_y),y4(dim_y)
dimension z0(dim_y,dim_x)
dimension nbr_noeud(dim_x)
dimension xn(dim_z,dim_y,dim_x),yn(dim_z,dim_y,dim_x)
dimension zn(dim_z,dim_y,dim_x)
dimension xn1(dim_z,dim_y),yn1(dim_z,dim_y)
dimension xn2(dim_z,dim_y),yn2(dim_z,dim_y)
dimension xn3(dim_z,dim_y),yn3(dim_z,dim_y)
dimension xn4(dim_z,dim_y),yn4(dim_z,dim_y)
dimension ind1(dim_z),ind2(dim_z),dm(dim_z,dim_y,dim_x)
dimension dtot(dim_z,dim_z)
dimension nbr_nouv_sect(dim_z)
```

c RECUPERATION DES PARAMETRES DU PROGRAMME AINSI QUE LES NOMS DE FICHIERS

```
write(*,*) 'combien de noeud pour la rive gauche?'
read(*,*) ng
write(*,*) 'combien de noeud pour le lit mineur?'
read(*,*) nl
write(*,*) 'combien de noeud pour la rive droite?'
read(*,*) nd
write(*,*) 'ATTENTION PAS D''ESPACE dans les noms'
write (*,*) ' '

n=ng+nl+nd

if (n.gt.250) then
        write(*,*) 'trop de points dans les sections'
end if
```

c OUVERTURE DU FICHIER CONTENANT LES SECTION ET LECTURES DES DONNÉES

```
open (14,FILE='sect.txt')
read(14,*) nbr_section

do i=1,nbr_section
read(14,*) nbr_noeud(i)
read(14,*) x1(i),y1(i)
read(14,*) x2(i),y2(i)
read(14,*) x3(i),y3(i)
read(14,*) x4(i),y4(i)
write(*,*)'section',i

        do j=1,nbr_noeud(i)
                read(14,*) x(i,j),y(i,j),z(i,j)
                write(*,*) j
        enddo

        do j=nbr_noeud(i)+1,n
        x(i,j)=0
                y(i,j)=0
                z(i,j)=0
        enddo
```

c CALCUL DES COORDONNÉES EQUIRÉPARTIES SUR CHACUN DES TROIS PARTIES DE SECTION

```
do j=1,ng
        x0(i,j)=(j-1)*(x2(i)-x1(i))/(ng-1)+x1(i)
        y0(i,j)=(j-1)*(y2(i)-y1(i))/(ng-1)+y1(i)
enddo

do j=ng+1,ng+nl
        x0(i,j)=(j-(ng+1)+1)*(x3(i)-x2(i))/(nl-1)+x2(i)
        y0(i,j)=(j-(ng+1)+1)*(y3(i)-y2(i))/(nl-1)+y2(i)
enddo

do j=ng+nl+1,n
        x0(i,j)=(j-(ng+nl+1)+1)*(x4(i)-x3(i))/(nd-1)+x3(i)
        y0(i,j)=(j-(ng+nl+1)+1)*(y4(i)-y3(i))/(nd-1)+y3(i)
enddo


z0(i,1)=z(i,1)
z0(i,n)=z(i,nbr_noeud(i))

do j=2,n-1
        k=1
        xlim = x0(i,j)
                do while (x(i,k) .le. xlim .and. k .lt.
nbr_noeud(i))
                k=k+1
                enddo

                if (k .lt. 1 .or. k .gt. nbr_noeud(i)) then
write(*,*) 'le poimteur est parti trop loin'
                end if

                if (k .ne. nbr_noeud(i) .and. xlim .gt. x(i,k))
then
write(*,*) 'attention x0(i,j) est plus grand que x4(i)'
                end if


                alpha=(x0(i,j)-x(i,k-1))/(x(i,k)-x(i,k-1))
                z0(i,j)=z(i,k-1)+alpha*(z(i,k)-z(i,k-1))
        enddo
enddo

close (14)

nbr_point=nbr_section*n
nbr_element=(2*n-2)*(nbr_section-1)
```

c AJOUT DE SECTION SI L'UTILISATEUR LE DÉSIRE

```
write(*,*) 'voulez-vous ajouter des sections 1=oui,2=non?'
read(*,*) choix
        if (choix .lt.2) then
open(18,FILE='position.txt')
read(18,*) nbr_groupe
```

c LECTURES DES PARAMÈTRES DE LOCALISATION DES NOUVELLES SECTIONS À INTERPOLER

```
do i=1,nbr_groupe
        read(18,*) ind1(i),ind2(i)
        read(18,*) nbr_nouv_sect(i)
        do j=1,nbr_nouv_sect(i)
                        read(18,*) xn2(i,j),yn2(i,j)
                        read(18,*) xn3(i,j),yn3(i,j)
        enddo
enddo
```

c CALCUL DES XN4,YN4,XN1,YN1 EN FONCTION DES LONGUEURS DES RIVES DES SECTIONS EXTRÊMES ET DES RAPPORTS DES DISTANCES DES POINTS D'INDICES 2 ET 3 AUX SECTIONS EXTRÊMES

```
!calcul des distances et longueurs intervenant ansi
que des rapports utiles

do i=1,nbr_groupe

        !rive droite

        long_riv_dr_ind1=sqrt((x4(ind1(i))-x3(ind1(i)))**2
```

```
     &+(y4(ind1(i))-y3(ind1(i)))**2)

          long_riv_dr_ind2=sqrt((x4(ind2(i))-x3(ind2(i)))**2
     &+(y4(ind2(i))-y3(ind2(i)))**2)

          ddt=sqrt((x3(ind2(i))-x3(ind1(i)))**2
     &+(y3(ind2(i))-y3(ind1(i)))**2)

          !rive gauche

          long_riv_ga_ind1=sqrt((x2(ind1(i))-x1(ind1(i)))**2
      &+(y2(ind1(i))-y1(ind1(i)))**2)

long_riv_ga_ind2=sqrt((x2(ind2(i))-
x1(ind2(i)))**2+(y2(ind2(i))-y1(ind2(i)))**2)

dgt=sqrt((x2(ind2(i))-x2(ind1(i)))**2+(y2(ind2(i))-
y2(ind1(i)))**2)

          do j=1,nbr_nouv_sect(i)

          !rive droite

                    ddn=sqrt((xn3(i,j)-x3(ind1(i)))**2
      &+(yn3(i,j)-y3(ind1(i)))**2)

                    long_riv_dr=long_riv_dr_ind1
      &+ddn/ddt*(long_riv_dr_ind2-long_riv_dr_ind1)

                    !rive gauche

          dgn=sqrt((xn3(i,j)-x3(ind1(i)))**2
      &+(yn3(i,j)-y3(ind1(i)))**2)
                    long_riv_ga=long_riv_ga_ind1
      &+dgn/dgt*(long_riv_ga_ind2-long_riv_ga_ind1)

                    !pente section

                    gamma=(yn3(i,j)-yn2(i,j))/(xn3(i,j)-
xn2(i,j))

                    !calcul des xn1 xn4,yn1 yn4


          xn4(i,j)=xn3(i,j)+long_riv_dr/sqrt((1+gamma**2))
                    yn4(i,j)=yn3(i,j)+gamma*(xn4(i,j)-xn3(i,j))

          xn1(i,j)=xn2(i,j)-long_riv_ga/sqrt((1+gamma**2))
                    yn1(i,j)=yn2(i,j)-gamma*(xn2(i,j)-xn1(i,j))


          enddo !fin boucle sur les sections

enddo !fin boucle sur les groupes
```

c CALCUL DES x(i,j,k),y(i,j,k)

```
do i=1,nbr_groupe

          do j=1,nbr_nouv_sect(i)

                    deltax12=(xn2(i,j)-xn1(i,j))/(ng)
                    deltay12=(yn2(i,j)-yn1(i,j))/(ng)
                    deltax23=(xn3(i,j)-xn2(i,j))/(nl)
                    deltay23=(yn3(i,j)-yn2(i,j))/(nl)
                    deltax34=(xn4(i,j)-xn3(i,j))/(nd)
          deltay34=(yn4(i,j)-yn3(i,j))/(nd)
                    xn(i,j,1)=xn1(i,j)
                    yn(i,j,1)=yn1(i,j)

                    do k=2,ng-1
                              xn(i,j,k)=xn1(i,j)+(k-1)*deltax12
                              yn(i,j,k)=yn1(i,j)+(k-1)*deltay12
                    enddo

                    xn(i,j,ng)=xn2(i,j)
                    yn(i,j,ng)=yn2(i,j)

                              do k=ng+1,ng+nl-1
                              xn(i,j,k)=xn2(i,j)+(k-
(ng+1)+1)*deltax23
          yn(i,j,k)=yn2(i,j)+(k-(ng+1)+1)*deltay23
                              enddo

          xn(i,j,ng+nl)=xn3(i,j)
          yn(i,j,ng+nl)=yn3(i,j)

                    do k=ng+nl+1,n-1
                    xn(i,j,k)=xn3(i,j)+(k-(ng+nl+1)+1)*deltax34
                    yn(i,j,k)=yn3(i,j)+(k-(ng+nl+1)+1)*deltay34
                              enddo

                    xn(i,j,n)=xn4(i,j)
                    yn(i,j,n)=yn4(i,j)

          enddo ! fin boucle section
 enddo !fin boucle groupe
```

C INTERPOLATION LINÉAIRE DES z,DÉBUT DES BOUCLES SUR LES SECTIONS ET SUR LES POINTS

```
do i=1,nbr_groupe
          do j=1,nbr_nouv_sect(i)
                    do k=1,n
```

c CALCUL DES POSITION DES NOUVELLES SECTIONS EN TENANT COMPTE DE TOUTES LES NOUVELLES SECTIONS D'UN GROUPE

```
dm(i,j,k)=sqrt((xn(i,1,k)-x0(ind1(i),k))**2+(yn(i,1,k)-
y0(ind1(i),k))**2)

dtot(i,k)=sqrt((xn(i,1,k)-x0(ind1(i),k))**2+(yn(i,1,k)-
y0(ind1(i),k))**2)

          if (j.ne.1) then
                    do l=1,j-1
dm(i,j,k)=dm(i,j,k)+sqrt((xn(i,l,k)-xn(i,l+1,k))**2+(yn(i,l,k)-
yn(i,l+1,k))**2)
                    enddo
          end if

                              do l=1,nbr_nouv_sect(i)-1
dtot(i,k)=dtot(i,k)+sqrt((xn(i,l,k)-xn(i,l+1,k))**2
      &+(yn(i,l,k)-yn(i,l+1,k))**2)
                              enddo

dtot(i,k)=dtot(i,k)+sqrt((xn(i,nbr_nouv_sect(i),k)-
      &x0(ind2(i),k))**2+(yn(i,nbr_nouv_sect(i),k)-
y0(ind2(i),k))**2)

beta=dm(i,j,k)/dtot(i,k)

zn(i,j,k)=z0(ind1(i),k)+beta*(z0(ind2(i),k)-z0(ind1(i),k))

                    enddo !fin boucle sur les points
          enddo !fin boucle sur les sections
enddo !fin boucle sur les groupes
```

c ÉCRITURE D'UN FICHIER CONTENANT TOUTES LES SECTIONS

```
open(19,FILE='fichier_temp')
i=1

do j=1,nbr_groupe

          do while (i .ne. ind1(j)+1 .and. i.lt. ind1(j)+1)
                    do k=1,n
                              write(19,*) x0(i,k),y0(i,k),z0(i,k)

                    enddo
          i=i+1
          enddo

          do l=1,nbr_nouv_sect(j)
                    do k=1,n
                              write(19,*) xn(j,l,k),yn(j,l,k),zn(j,l,k)
                    enddo
          enddo
enddo

do l=i,nbr_section
          do k=1,n
                    write(19,*) x0(i,k),y0(i,k),z0(i,k)
          enddo
enddo

close(19)

somme_nouv_sect=0

do i=1,nbr_groupe
```

```
nbr_point=nbr_section*n
nbr_element=(2*n-2)*(nbr_section-1)
write(*,*) somme_nouv_sect

open(20,FILE='fichier_temp')
cont=0
do ii=1,nb_tot_sect

        do jj=1,n
                read(20,*) x0(ii,jj),y0(ii,jj),z0(ii,jj)
                write(*,*) ii,jj
                cont=cont+1
        enddo
enddo
close(20)
write(*,*) nb_tot_sect
write(*,*) cont
```

c FICHIER DONNÉES BRUTES DE TOPOGRAPHIE

```
open (15,FILE='donnees_brutes_topo')
write(15,*) nbr_point

do i=1,nbr_section
                do j=1,n
                        write(15,*)
x0(i,j),y0(i,j),z0(i,j),0.00
                enddo
        enddo

        close (15)
```

c FICHIER MAILLAGE

```
        open (16,FILE='maillage_reg')
        write(16,*)nbr_point,nbr_element,2
        do i=1,nbr_section
                do j=1,n
        numero_noeud=(i-1)*n+j
        write(16,*)numero_noeud,numero_noeud,x0(i,j),y0(i,j)
                enddo
        enddo

        write(16,*)0,0
        write(16,*)1,0
        write(16,*)2,0
        write(16,*)3,nbr_element
        num_elem=1

do i=1,nbr_section-1
        do j=1,n-1
                sommet_11=j+i*n
                sommet_12=j+i*n+1
                sommet_13=j+(i-1)*n
                sommet_21=j+(i-1)*n+1
                sommet_22=j+(i-1)*n
                sommet_23=j+i*n+1

        write(16,*)num_elem,sommet_11,sommet_12,sommet_13
                num_elem=num_elem+1

        write(16,*)num_elem,sommet_21,sommet_22,sommet_23
                num_elem=num_elem+1
        enddo
enddo

write(16,*)4,0
write(16,*)5,0
write(16,*)6,0

close(16)
```

c FICHIER CHAMP SCALAIRE DE TOPOGRAPHIE

```
open (17,FILE='champ_scal_topo')
write(17,*) nbr_point
do i=1,nbr_section
        do j=1,n
        write(17,*) z0(i,j)
        enddo
enddo

close(17)

end if ! FIN DE LA CONDITIONNELLE SUR CHOIX DE
L'AJOUT DE SECTION
```

```
end if ! FIN DE LA CONDITIONNELLE SUR CHOIX DE
L'AJOUT DE SECTION
```

c FICHIER DONNÉES BRUTES DE TOPOGRAPHIE

```
open (15,FILE='donnees_brutes_topo')
write(15,*) nbr_point

do i=1,nbr_section
        do j=1,n
                write(15,*) x0(i,j),y0(i,j),z0(i,j),0.00
        enddo
enddo

close (15)
```

c FICHIER MAILLAGE

```
open (16,FILE='maillage_reg')
write(16,*)nbr_point,nbr_element,2
do i=1,nbr_section
        do j=1,n
        numero_noeud=(i-1)*n+j
        write(16,*)numero_noeud,numero_noeud,x0(i,j),y0(i,j)
        enddo
enddo

write(16,*)0,0
write(16,*)1,0
write(16,*)2,0
write(16,*)3,nbr_element
num_elem=1

do i=1,nbr_section-1
        do j=1,n-1
                sommet_11=j+i*n
        sommet_12=j+i*n+1
                sommet_13=j+(i-1)*n
                sommet_21=j+(i-1)*n+1
                sommet_22=j+(i-1)*n
                sommet_23=j+i*n+1
                write(16,*)num_elem,sommet_11,sommet_12,sommet_13
                num_elem=num_elem+1
                write(16,*)num_elem,sommet_21,sommet_22,sommet_23
                num_elem=num_elem+1
        enddo
enddo

write(16,*)4,0
write(16,*)5,0
write(16,*)6,0
close(16)
```

c FICHIER CHAMP SCALAIRE DE TOPOGRAPHIE

```
open (17,FILE='champ_scal_topo')
write(17,*) nbr_point
do i=1,nbr_section
        do j=1,n
        write(17,*) z0(i,j)
        enddo
enddo

close(17)

END
```

2 Données fournies par le Ministère de l'environnement et de la Faune du Québec

a) Hydrologie :

- Liste et description des stations hydrométriques en activité ou dont les données sont archivées
- Côtes de crues pour la rivière Chaudière (géoréférencées) : tableau de valeurs tous les ½ Km pour les crues de période de retour 20 et 100 ans et tous les 2 ½ Km pour les 2, 5, 10, 50 ans recalculés en 2002 pour réactualiser les cartes d'inondation, ainsi que les débits de crues utilisés (2 ans 20 ans et 100 ans)
- Données des hauteurs d'eau pour plusieurs stations hydrométriques autour de Sainte-Marie : une station au pont de Sainte-Marie (encore en service), une 5.1 Km en aval du pont fermée en 1971), une à Scott-Jonction
- Débits de crues, probabilité écart type pour les crues de récurrence : 1.0001, 1.0005, 1.001, 1.005, 1.0101, 1.0204, 1.0526, 1.1111, 1.25, 1.4286, 2, 3, 5, 10, 20, 50, 100, 200, 1000, 2000, 10000
- Niveaux de crues calculé en 1978 le long de la rivière de Saint-Lambert à Sartigan à 80 Km plus loin pour les récurrences 2, 5, 10, 20, 50, 100 ans
- Données de débits sur la station de Saint-Lambert : moyenne journalière de 1925 à 2003, débit aux 15 min pour les évènements des 2 juillet 2002, 31 mars 2003, 1 avril 2003 et 2 mai 2003
- 5 tableaux d'observation des niveaux d'eau entre Saint-Lambert et Saint-Georges le 4 avril 2002, le 2 juillet 2002, le 31 mars 2003, le 1 avril 2003 et le 2 mai 2003
- Rapport courbe de remous rivière Chaudière par José Llamas, ing, Québec 1966

b) Bathymétrie :

- Fichier de géométrie H.E.C.-R.A.S. contenant les sections relevées en 2002 par le Ministère de l'Environnement du Québec.
- Notes et remarques relevées par les techniciens au sujet des sections (utiles pour les renseignements sur le substrat au niveau de chaque section)

c) Autre :

- 2 cartes d'inondation éditées en 1979 par le MENV entourant Ste-Marie
- 1 carte de localisation des sections fournies.

3 Index des Cartes disponibles pour le projet

- Carte de sensibilité au ruissellement des unités de paysage (Rapport TECSULT)
Pas d'intérêt particulier pour le projet

- Carte de sensibilité à l'érosion des unités de paysage (R.T.)
Pas d'intérêt particulier pour le projet

- Carte de localisation et description des unités de paysage (R.T.)
Description de la couverture du bassin versant dans son ensemble (pas assez précis pour l'étude de notre tronçon

- Évolution des réseaux de transport (R.T.)
Permet d'avoir une idée du réseau routier et des voies de chemin de fer dans le bassin versant, mais pas assez précis pour notre étude

- Zones d'étude et localisation des stations hydrométriques et climatologiques utilisées pour la modélisation (R.T.)

- Les stations hydrométriques utilisées sont celles du pont-route de Sainte-Marie et celle de Chaudière-Saint-Lambert située à 26 Km en aval de Sainte-Marie

- Ouvrage de rétention des glace sur la rivière Bras Saint-Victor (R.T.)
Localisation et description des ouvrages (pieux caisson pour créer un embâcle)

- Localisation et caractéristiques des réservoirs étudiés sur les tributaires de la rivière Chaudière (R.T.)
- Solution d'excavation (tronçon Scott) Km 14 à 19 (R.T.)
    - variante 1 : écrêtement du seuil rocheux à l'aval de Scott Jonction
    - variante 2 : excavation du chenal entre Scott et Sainte-Marie
- Solution d'excavation (tronçon Scott – Sainte-Marie) I : Km 19 à 24 (R.T.)
    - variante 2 : excavation du chenal entre Scott et Sainte-Marie
    - variante 3 : écrêtement du seuil à Scott et excavation du chenal
- Solution d'excavation (tronçon Scott – Sainte-Marie) II : Km 24 à 28 (R.T.)
    - variante 2 : excavation du chenal entre Scott et Sainte-Marie
    - variante 3 : écrêtement du seuil à Scott et excavation du chenal
- Projet d'endiguement de la ville de Sainte-Marie : plan et profil (R.T.)

- Solution aux inondations par embâcle à Beauceville par profilage du fond des berges du Km 56 à 60.5 : vue en plan / profil en long et coupe **1/2** (R.T.)

- Solution aux inondations par embâcle à Beauceville par profilage du fond des berges du Km 56 à 60.5 : vue en plan / profil en long et coupe **2/2** (R.T.)

- Carte d'inondation de la vallée Jonction (Ministère Environnement 1978)

- Carte d'inondation de la ville de Sainte-Marie (M. E. 1978)

- Carte d'inondation de la ville de Sainte-Marie annotée par les Travaux Publics de la ville (M.E. + Travaux Publics de la ville de Sainte-Marie)

- Photo aérienne de la ville de Sainte-Marie au 1:4000 +lignes de niveaux (Arpenteur Géomètre consultant Ville de Sainte-Marie)

4 Listes des crues les plus importantes (q>1300m^3/s)

Les débit sont mesurés à la station hydrométrique de Saint-Lambert (entre 1915 et 2003). Les données antérieures à 1991 proviennent du rapport TECSULT

Débits (m^3/s)	Dates
3500 (estimé)	1/04/1917
2140	10/04/1991
1980	1/04/1987
1860	30/03/1998
1830	4/05/1926
1723	17/04/1994
1680	30/04/1974
1650	07/05/1947
1640	16/04/1955
1640	18/04/1969
1640	19/04/1982
1580	27/04/1942
1540	23/04/1992
1420	23/04/1958
1390	26/03/1979
1390	25/04/2001
1380	1/04/1986
1360	5/05/1972
1350	19/06/1922
1330	30/03/1989
1320	25/09/1981
1320	19/04/1970
1310	21/04/1975
1309	2/07/02

Bibliographie référée au texte ou seulement consultée

- Analyse de risques d'inondation par embâcles de la rivière Montmorency et identification de solutions techniques innovatrices. Phase I - pré faisabilité
- Développement de solutions techniques innovatrices pour le contrôle des embâcles de la rivière Montmorency. Phase I – Pré-faisabilité Rapport Université LAVAL, département génie civil, Rapport INRS-ETE R577a janvier 2001.
- Développement de solutions techniques innovatrices pour le contrôle des embâcles de la rivière Montmorency. Phase II – faisabilité. Rapport Université LAVAL, département génie civil GCT-2002-03, INRS-ETE R577b février 2002.
- Simulation hydrodynamique et analyse morpho-dynamique de la rivière Montmorency en crue dans le secteur des Ilets. Rapport de recherche INRS-Eau R522, février 1998
- Travaux d'atténuation des risques de crue à l'eau libre de la rivière Montmorency dans le secteur des Ilets. Phase II Rapport de recherche INRS-Eau R555, mars 2000.
- Protection des Rives et du Littoral et des Plaines Inondables guide des bonnes pratiques. Environnement et Faune Québec. Jean-Yves Goupil. Publication du Québec 1998
- State-of-the-art of estimating flood damages in urban areas, Neil S. Grigg and J. Helweg. *Water Resources Bulletin*, vol 11, n°2, April 1975
- Estimating flood frequency and average annual damage, Léo R. Beard, *Journal of Water Resources Planning and Management*, March/April 1997
- Perception of flood risk : a case study of the Red River flood of 1997, *Water Resources Research*, vol 35, n°11, November 1999
- Expected probability and annual damage estimators, Jerry Stedinger, *Journal of Water Resources Planning and Management* , march/April 1997
- Le risqué d'inondation et son évolution sur la rivière Châteauguay, M.-c. Bouillon, F.P. Brissette, C. Marche, *Canadian Journal for Civil Engineering*, 26: 186-196, 1999
- Historical evolution of the flooding risk on a Québec river, F.P. Brissette, R. Leconte, C. marche, J. Rousselle, 15ème congrès canadien d'hydrotechnique de la Société canadienne de genie civil, Victoria, B.C., Canada, 30 mai au 2 juin 2001
- A two-dimensional finite element drying-wetting shallow water model for rivers and estuaries, M.Heniche & col, *Int. J. Advanced in Water Resources*, 23:359-372, 2000
- GMRES-An introduction, Y.Secretan, document interne Département de Génie Civil Université Laval
- Modeleur Guide d'utilisation version 1.0a07,Y.Roy, Y.Secretan &coll., juin 2000, rapport INRS-eau R482-G3F
- Hydrosim Guide d'utilisation version 1.0a06, M.Heniche, M.Leclerc, Y.Secretan, juin 2000, rapport INRS-eau R482-G2
- Étude de modélisation du bassin versant de la rivière Chaudière rapport d'étape n°1, les effets du développement sur le comportement de la rivière, vol.1 État de la situation, TECSULT Inc. : P.R. Tremblay & coll., mars1993
- Étude de modélisation du bassin versant de la rivière Chaudière rapport d'étape n°2, Solution aux inondations en eau libre, TECSULT Inc. : P.R. Tremblay & coll., janvier 1994