Interfaçage de l'outil de modélisation PROSE : développement de l'outil GABI, Générateur Automatique de Belles Interfaces

Stéphanie EVEN[1], Ronan KERYELL[2], Pierre PAQUIN[2]
[1] : Centre de Géosciences,
École des Mines de Paris,
35 rue Saint-Honoré, 77 305 FONTAINEBLEAU cedex
Stephanie.Even@ensmp.fr
[2] : Laboratoire d'Informatique et Télécommunications,
École des Télécommunications de Bretagne,
Technopôle Brest-Iroise, 29280 PLOUZANE

Février 2006

1. Introduction

Pendant cette phase du PIREN SEINE et dans le cadre du développement d'outils opérationnels en vue de leur diffusion, l'accent a été mis sur les aspects interfaçage du logiciel PROSE (Even et al.

2002; Even et al. 2003; Even et al. 2004). Les travaux réalisés les années précédentes ont permis le développement d'un outil GABI conçu comme un éditeur générique de format des données d'entrée de logiciels scientifique basés sur l'utilisation de parsers (Bonniez 2001; Even et al. 2002; Even et al. 2003) et qui a été appliqué à ProSe.

Par ailleurs une interface générale a été développée permettant de gérer l'appel aux fonctionnalités autour du logiciel ProSe : création des données en entrée, lancement du calcul, post-traitements permettant de générer des graphiques (Even et al. 2004).

Les travaux proposés pour l'année 2005 concernent essentiellement des développements pour rendre l'outil GABI complètement opérationnel :

- la gestion des commentaires ;
- la gestion des inclusions de fichiers ;
- la lecture de fichiers texte en entrée.

2. Concepts de l'outil GABI

2.1. Généralités

Dans la majorité des cas, les interfaces sont en effet crées au cas par cas, soit par les concepteurs de logiciels, soit leur développement est sous-traité à des laboratoires ou bureaux d'étude. Par ailleurs de nombreux outils de génération automatique d'interface existent déjà (Schlee 2002). Cependant, ces outils restent souvent critiqués pour leur incapacité à produire des interfaces riches et complexes et une partie de la conception reste dans tous les cas manuelle (description de l'interface, des données). Il en résulte que le développement d'une interface pour un logiciel est une opération souvent coûteuse, pouvant conduire à des limitations en terme de développement/adaptation du logiciel de calcul lui-même.

L'idée du projet GABI est d'automatiser complètement la création d'une interface permettant de créer/modifier les fichiers de données lus en entrée de logiciels scientifiques. Pour cela l'outil doit être capable d'analyser la partie *code de lecture des données d'entrée* d'un logiciel et de générer le code d'interface automatiquement (figure 1). Un programme consiste à interpréter et structurer des données, calculer et générer des résultats. Un des principes du projet est de séparer la partie du code assurant la lecture et l'interprétation des données de la partie calculatoire du modèle.

Le projet est basé sur l'utilisation de parsers, qui permettent une représentation des données sous la forme d'une arborescence (figure 2). Cette représentation conceptuelle est facilement accessible à partir de l'analyse des fichiers définissant le lexique et la grammaire d'un logiciel. Ainsi chaque action de la grammaire correspond à un nœud de l'arbre. Les actions finales utilisent quant à elle les éléments lexicaux correspondant aux feuilles de l'arbre. Elles sont de trois types :

identifiant ou chaîne de caractère connue ;

une liste de plusieurs variables possibles ;

une valeur à définir par l'utilisateur.

L'application est développée en JAVA afin de profiter des facilités des langages objets et des nombreux composants graphiques et de gestion des données fournis en standard. C'est un langage couramment utilisé qui bénéficie de nombreux développements. De plus il présente l'avantage d'être indépendant de la machine. L'outil GABI est actuellement géré sous SVN permettant un travail de développement et d'archivage multi-utilisateurs.

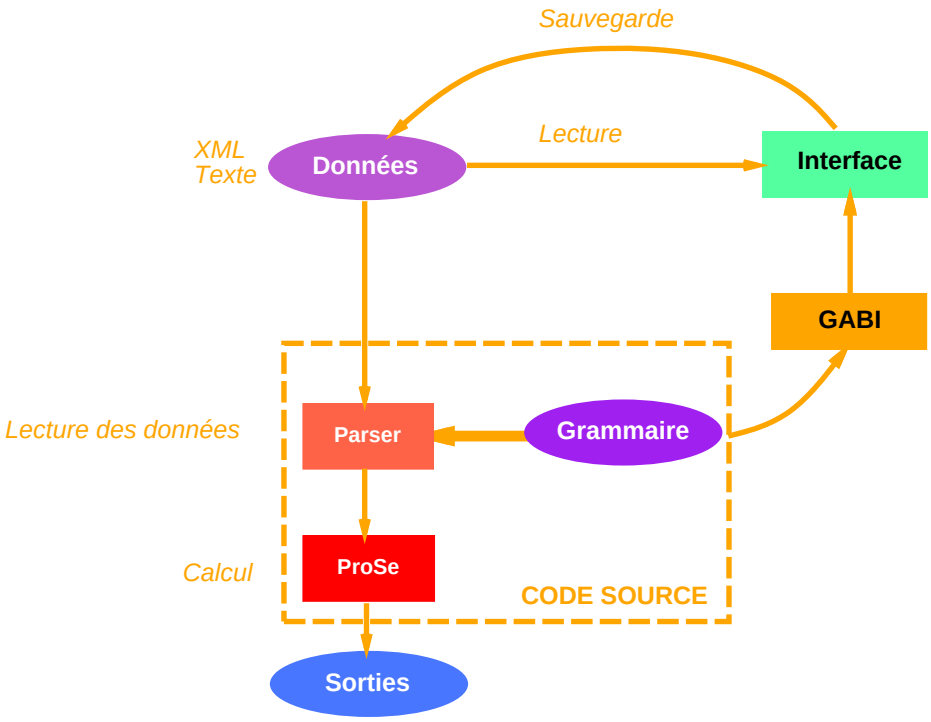

Figure 1: *Schématisation des interconnections entre les différents modules constitutifs d'un logiciel scientifique interfacé à l'aide de l'outil* GABI.

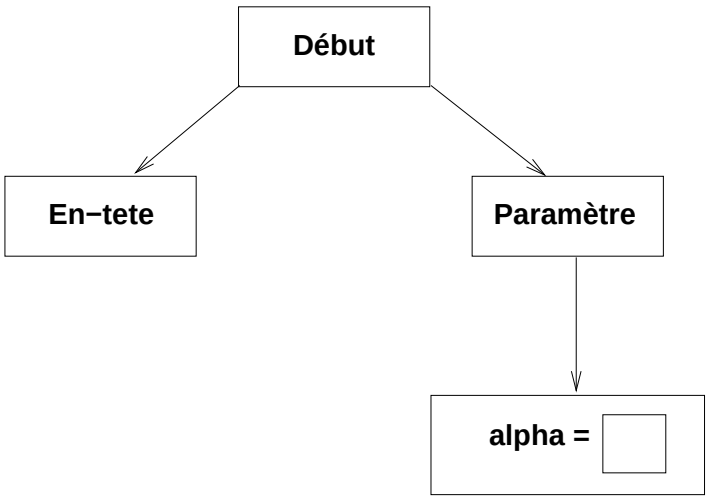

Figure 2: *Représentation succinte et schématique d'une arborescence de données.*

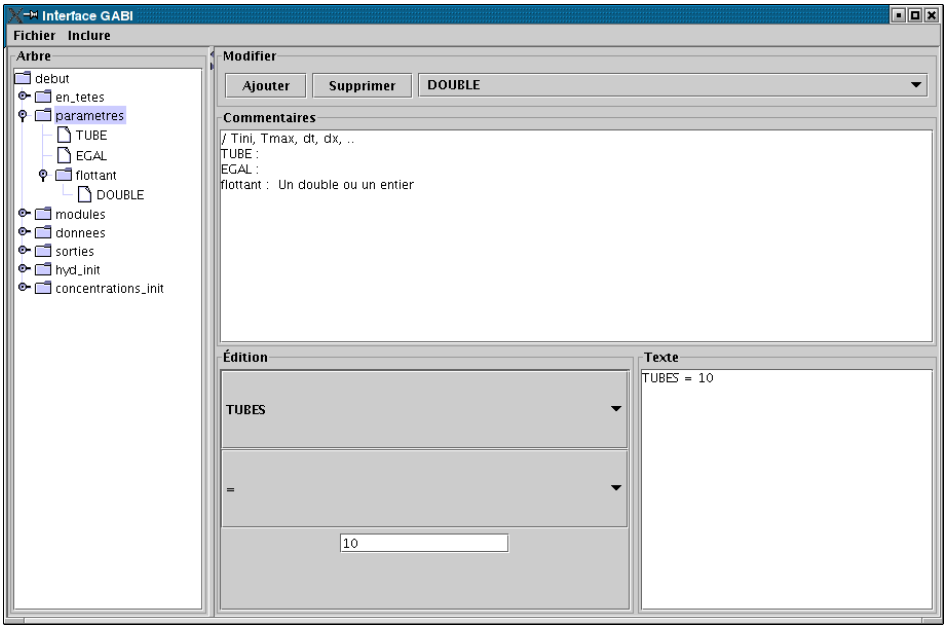

Figure 3: *Affichage et implémentation d'une arborescence de données à l'aide de l'outil* GABI. *Remplissage du champ valeur par l'utilisateur. Affichage du texte dans la partie en bas à droite.*

2.2. État d'avancement du projet

Les fonctions assurées actuellement par l'outil sont :

1. la **construction de l'arbre conceptuel des données**. Cette étape est réalisée à travers l'analyse des fichiers lexique(`.lex`), fournissant les composants de base, c. à d. les feuilles de l'arbre et grammatical (`.yacc`) permettant de décrire la structure de l'arbre.

2. la **mise en forme de l'interface** : affichage de l'arbre, fonctions de modification et d'édition. À ce niveau, le choix d'une interface dynamique a été réalisé afin de limiter les problèmes d'espace mémoire et d'augmenter la rapidité de réponse.

L'ensemble de ces fonctions ont été intégrées dans une interface disposant des fonctions de génération de l'arbre conceptuel, de sa sauvegarde, d'ouverture d'un ancien arbre, de son affichage, de sa modification (figure 3). Il est donc possible de créer un jeu de données, de les afficher au sein de l'interface et de les sauvegarder au format texte. La relecture au format texte n'est actuellement pas possible. Les données sont donc également sauvegardées dans un format d'arbre sérialisé, seul format actuellement lisible en entrée de l'interface.

Deux fonctions importantes des parsers ont également été introduites :

- la fonction commentaire ;
- la fonction inclure.

2.3. Objectif du travail réalisé en 2005

La gestion de fonctions « commentaire » et « inclusion » ont été améliorées. La mise en place d'une fonction de lecture de données au format texte a également été introduite. En effet, dans sa version

actuelle, l'outil GABI ne lit que des arboresences stockées en un format sérialisé.

3. Les commentaires

Les commentaires peuvent être envisagés sous deux formes :

1. les commentaires « linéaires » introduits par un tag et tels que le restant d'une ligne compris entre le tag et la fin de la ligne sont ignorés :

```
# Ceci est un commentaire dans \prose

// Ceci est un commentaire linéaire en C, java ...
```

2. les commentaires « paragraphes » ; sont alors considérés en commentaire tous paragraphes compris entre deux tags. Dans ce cas, deux tags complémentaires sont généralement définis : un tag d'ouverture et un tag de fermeture :

```
/* Ceci est un commentaire
un peu plus long,
tel qu'on l'écrit en C
*/
```

3.1. Détection automatique des commentaires

Nous ne nous sommes intéressés qu'au premier cas. Ce type de commentaire est géré très simplement au niveau du fichier lexical. Il suffit de déclarer que le(s) caractère(s) de tag suivis de tous les caractères, sauf une fin de ligne, sont en commentaire, à savoir que la valeur renvoyée est nulle :
`#[\^ \n]*\n` { no_ligne++;} /* Vire les commentaires. */
Lors de l'analyse du fichier lexical, nous pouvons donc simplement identifier

1. la définition de commentaires de type linéaire, en reconnaissant un référent se terminant par `[\^ \n]*\n`
2. garder en mémoire le tag d'introduction au commentaire précédent la chaîne `[\^ \n]*\n`, en l'occurence #.

Plusieurs tags de commentaires peuvent être définis et sont mémorisés dans un vecteur.

3.2. Gestion des commentaires dans l'interface

La déclaration des commentaires au niveau du fichier lexical permet de les identifier et de les traiter à n'importe quel niveau dans les fichiers de données.

Gestion des commentaires au niveau de l'arbre Le premier traitement des commentaires qui avait été réalisé consistait à ajouter systématiquement une sous action commentaire à tous les nœuds de l'arbre. Cela s'est avéré alourdir énormément l'arborescence, car chaque commentaire était vu comme un nœud, au même titre que les autres actions, et apparaissait alors dans l'arbre (figure 4). De plus, les commentaires ne pouvaient être attribués à des feuilles, car les feuilles ne peuvent pas contenir de nœuds.

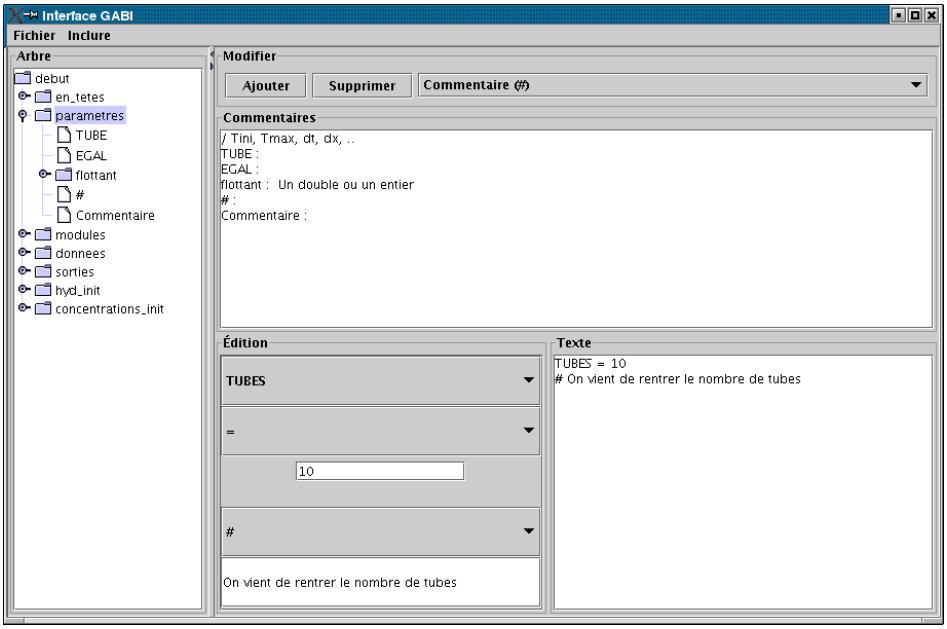

Figure 4: *Gestion des commentaires au niveau des nœuds de l'arbre. Les commentaires apparaissaient alors comme sous action possible de chaque noeud (menu de sélection en haut à droite) et étaient affiché dans l'arbre (à gauche) à chaque instanciation.*

Gestion des commentaires au niveau d'un menu Nous avons, dans un second temps, préféré gérer les commentaires au niveau d'un menu (figure 5). Des sous nœuds de type « commentaire » ont été crées, qui sont distincts des nœuds « action » de l'arbre. Les commentaires peuvent désormais être introduits à un nœud ou une feuille.

L'utilisateur sélectionne le nœud ou feuille de l'arbre qu'il veut commenter, puis il sélectionne le menu `commentaire`. Une fenêtre s'ouvre, où l'utilisateur saisit son texte puis le valide.

Le commentaire n'apparaît plus explicitement au niveau de l'arbre, ce qui en allège la présentation. L'intitulé du nœud ou de la feuille commenté sont simplement affichés avec en extension le tag du commentaire (ex : # dans notre cas).

Le commentaire est par contre affiché dans la fenêtre texte, en bas à droite (figure 5).

4. Les inclusions

L'inclusion de fichiers est également gérée au niveau du fichier lexical. Elle autorise une structure complexe de fichiers imbriqués en lecture. Les informations relatives aux fichiers ouverts en cascade sont enregistrées dans une « pile », ou tableau d'objets fichiers. A chaque nouveau fichier ouvert, la pile est incrémentée. A chaque fermeture de fichier on revient au fichier précédent, dont l'adresse, le numéro de ligne, etc ont été gardés en mémoire.

La déclaration de la fonction d'inclusion comprend i) la déclaration d'un mot clé (ex : `inclure` ou `include`) ii) la lecture, conditionnée par un état particulier, d'un nom de fichier. Ces deux actions conjuguées conditionnent l'ouverture du nouveau fichier qui incrémente la pile des fichiers et l'actualisation des coordonnées du fichier vers lequel est dirigé la lecture du parser.

Les fins de fichier sont détectées automatiquement par le parser. Une focntion yywrap() est appelée automatiquement et qui peut être reprogrammée. Dans notre cas, chaque fin de fichier provoque

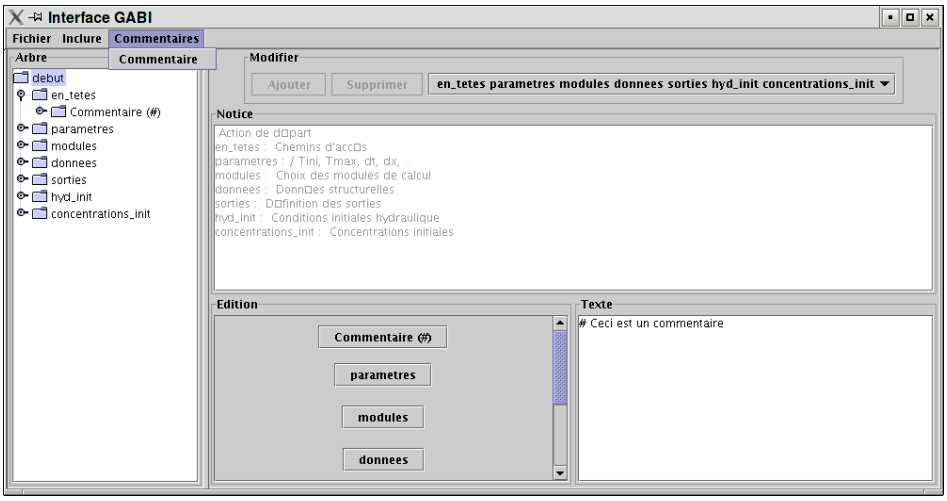

Figure 5: *Gestion des commentaires par un menu.*

la décrémentation de la pile de fichier et une réactualisation des coordonnées du fichier vers lequel est dirigé la lecture du parser.

Les fonctions inclure peuvent se trouver à tout endroit et permettent une structuration très souple des données, dont l'utilisateur final du logiciel est entièrement maître (figure 6).

4.1. Détection automatique des inclusions

Pour détecter automatiquement les inclusions nous avons dû adopter une convention (figure 7) :

- l'existence de référent(s) commençant par la chaîne « incl », indépendamment des minuscule ou majuscule ;
- ce référent conditionne le passage dans un état spécifique qui commence lui-même par « incl » pour la lecture du nom du fichier.

Une fois la présence de la fonction inclure détectée dans le fichier lexical, un menu « inclure » dans la barre de menus de l'interface générée par GABI (figure 5).

4.2. La gestion des inclusions dans l'arborescence des données

4.2.1. Gestion au niveau des nœuds de l'arbre

C'est cette option qui avait été choisie dans un premier temps, car très simple à mettre en œuvre.

Le menu **inclure** comprend trois fonctions : **Inclure un fichier**, **Désélection** et **Retour à la racine**.

Inclure un fichier Après avoir cliqué sur un nœud, la sélection du menu `Inclure/Inclure un fichier` ouvre une fenêtre dans laquelle l'utilisateur saisit le nom d'un fichier. Le nœud est alors affiché en étant suivi de l'indication (INCLURE) (figure 8).

Figure 6: *Extrait d'un fichier lu en entrée de* ProSe*, contenant des déclarations implicites de paramètres et des inclusions de fichiers. A titre d'exemple, on pourrait avoir une déclaration du type* `tini = inclure FichierX` *avec* `FichierX` *contenant* `1.0 [j]`

```
chemins : $HOME/Seine/
        : $HOME/Marne/
        : $HOME/Oise/

#Instant initial
tini = 1.0 [j]

#Instant final
tmax = 366.0 [j]

#Pas de temps de calcul
delta_t = 5 [mn]

#Pas d'espace
delta_x = 500 [m], 1.0

# Structure de la rivière
inclure seine
inclure Neuilly_conf/marne
inclure Mery_Conf/oise

# Conditions limite amont
inclure AmontChoisy1996
inclure AmontNeuilly1996
inclure AmontMery1996
inclure AffluentsChoisyPoses1996

#Apports 1996
inclure ApportsChoisyPoses1996Bis
inclure ApportsNeuillyConf1996
...
```

```
/** @file entree.l
* @brief Définition des mots du lexique
*/


%x inclure /* Définition des états */


\ldots /* Déclarations diverses */

%%

/* Liste des référents */

[Ii][Nn][Cc][Ll][Uu][Rr][Ee] {BEGIN(inclure);}

/* Lecture d'une chaîne de caractères = nom du fichier*/
<inclure>[^$ \t\n:;,\"]+
{
/* Action d'ouverture du fichier */
BEGIN(INITIAL); /* Retour à l'état initial */
}
```

Figure 7: *Déclaration d'une fonction d'inclusion de fichier dans un fichier lexical .lex : i) déclaration de la liste des états en en-tête dont l'état* `inclure`*, ii) déclaration du tag d'inclusion (inclure à lire en minuscule ou majuscule) commandant le passage dans l'état* `inclure`*, iii) lecture d'une chaîne de caractères quelconques conditionnée par l'état* `inclure` *conditionnant le passage à l'état* `INITIAL`*. En plus du retour à l'état* `INITIAL`*, la lecture du nom du fichier commande l'appel à une fonction qui incrémente la pile des fichiers et réoriente le pointeur de fichier à lire vers le nouveau fichier ouvert. Cette procédure n'est pas détaillée dans l'exemple. Les définitions des états et les tags servant à définir l'inclusion sont* a priori *entièrement libres. Pour l'analyse à l'aide de l'outil* GABI *nous avons convenu de les appeler par des chaînes commençant par « incl ».*

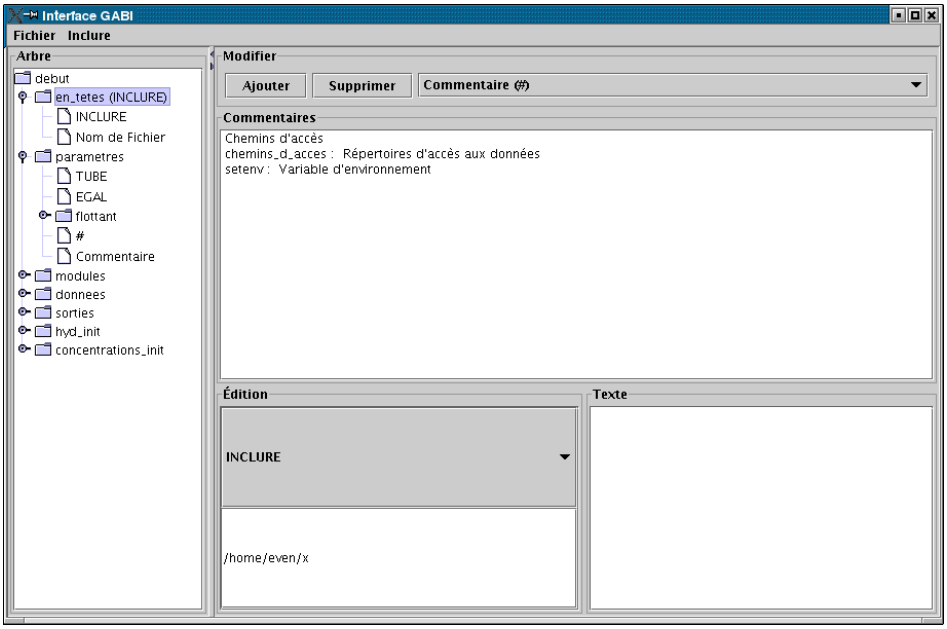

Figure 8: *Utilisation de la fonction* **inclure** *pour créer un fichier* x *où seront écrites les données de l'en-tête du fichier.*

Si l'utilisateur clique sur le nœud correspondant à l'inclusion, seule la partie de l'arbre comprise sous ce nœud est affichée. De même, la fenêtre texte (en bas à gauche) ne contiendra que le contenu strict du fichier inclus (figure 9).

Désélection Cette option permet d'annuler l'inclusion d'un fichier.

Retour à la racine Cette option permet de revenir à la racine générale de l'arbre. Le sous-arbre situé après le nœud de l'inclusion n'apparaît plus explicitement. Dans la fenêtre texte, seule l'information `inclure` suivi du nom du fichier apparaît.

Remarque Si cette solution est très simple à mettre en œuvre, elle est trop contraignante pour la structuration des données. En effet, elle suppose que toute l'information contenue dans un même sous arbre se trouve forcément dans un même fichier.

Prenons pour exemple d'une rivière, décrite à partir de singularités (barrages, point de confluence, ...) et de biefs (zones entre singularités). La rivière se présente alors elle même sous forme d'un graphe avec des nœuds (singularités) reliés entre eux par les biefs (figure 11).

Prenons le cas d'une action qui décrit la lecture des biefs. Si une fonction inclure est introduite à ce nœud, tous les biefs devront être contenus dans le même fichier. Cela ne permet pas :

- de sectionner l'information de description des biefs dans différents fichiers ;
- de mettre dans un même fichier l'information des biefs et celle des singularités, si les singularités sont vues comme une action de même niveau que celle des biefs ;

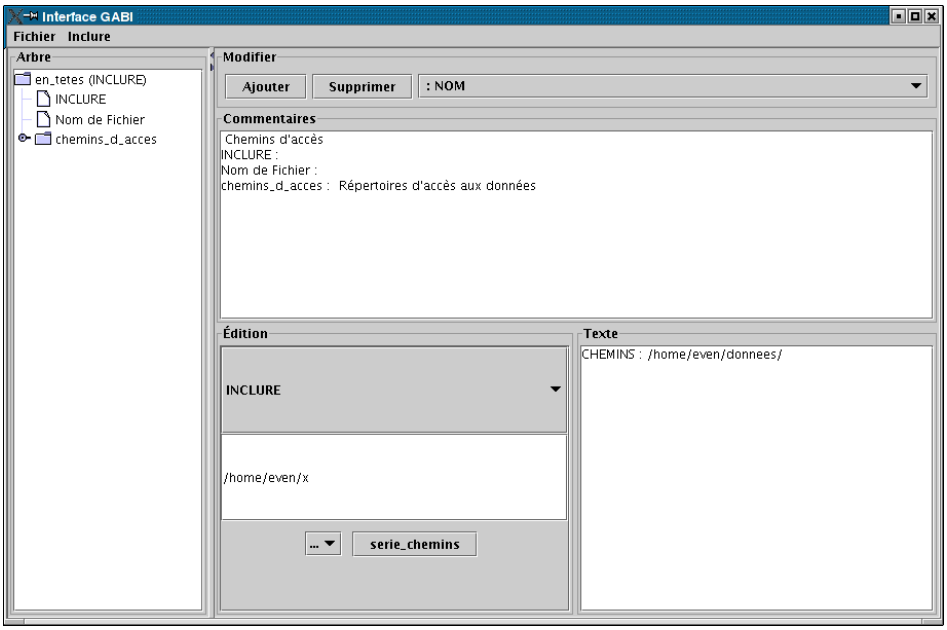

Figure 9: *Visualisation d'un sous-arbre à partir d'un nœud où se situe une inclusion.*

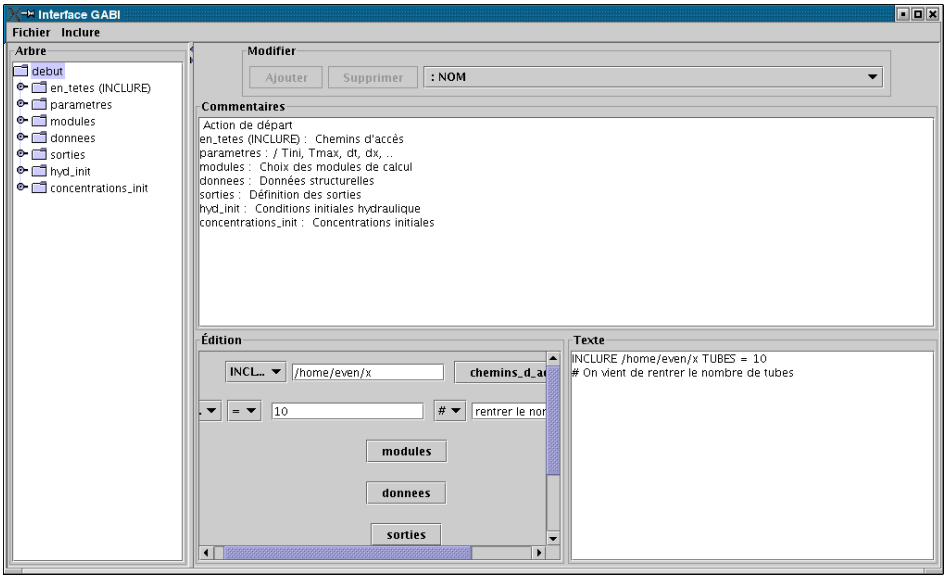

Figure 10: *Retour à la racine générale de l'arbre. Le nœud en tête n'apparaît qu'à travers la ligne* `inclure /home/even/x` *et le contenu du fichier n'est plus affiché.*

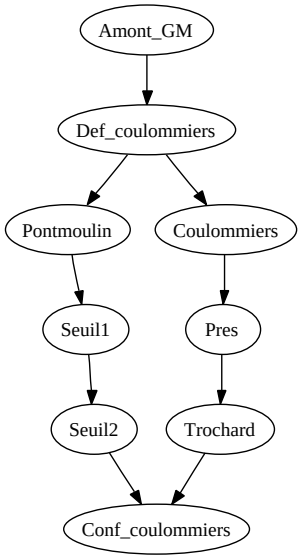

Figure 11: *Exemple de schéma de rivière tel que décrit dans* ProSe.

Or, ces pratiques sont courantes : les bases de données servant à décrire des cours d'eau sont organisées par secteurs géographiques et selon les simulations ont inclus la base de données pour une rivière (biefs et singularités dans un même fichier) puis celle d'une autre rivière (figure 11).

Pour prendre un autre exemple, supposons que l'on parse une boucle `while` en langage C :

```
 while {
    a = b ;
}
```

Avec la gestion des `inclure` au niveau des nœuds, on peut avoir

```
 while {
    inclure FichierInstruction
}
```

avec `FichierInstruction` contenant `a = b ;` mais pas

```
 while {
    inclure FichierInstruction
```

avec `FichierInstruction` contenant `a = b ;}` . Ces à dire que des informations de même niveau ne peuvent se retrouver dispatchées dans différents fichiers. Or les parsers permettent *a priori* une structuration quelconque de l'information. Il y a donc une perte de potentiel et de souplesse.

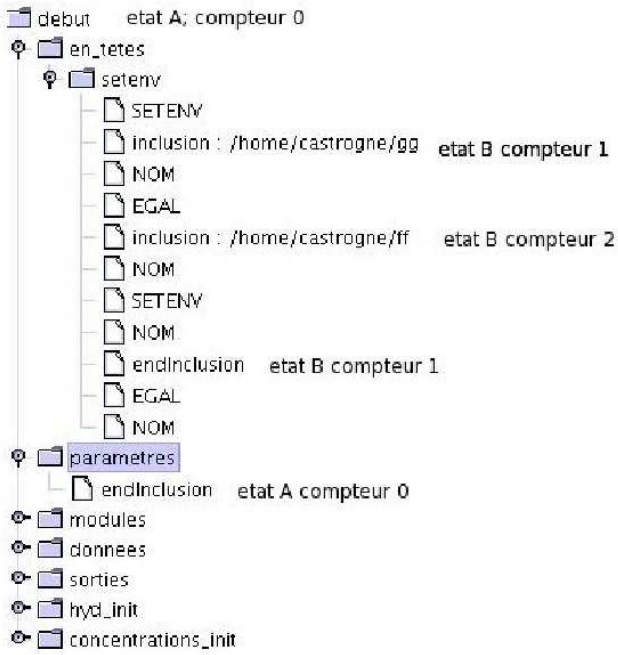

Figure 12: *Marqueurs d'inclusion dans une arborescence de données générées à partir du logiciel* P*ro*S*e*

4.2.2. Gestion à l'aide de marqueurs

L'information contenue dans un fichier est bornée par deux marqueurs : un marqueur désignant le début de l'inclusion, un marqueur pour la fin de l'inclusion (figure 12). Les marqueurs peuvent se trouver à tout niveau de l'arbre.

Un compteur est incrémenté à chaque « début » d'inclusion et décrémenté à chaque « fin » d'inclusion. Les marqueurs « début » et « fin » ayant la même valeur de compteur se correspondent. Ce compteur a également une valeur de vérification.

De même que pour les commentaires, l'inclusion n'est plus gérée comme un nœud simple, mais un type de nœud `inclure` a été crée. Les marqueurs d'inclusion peuvent alors être adressés tant à un nœud de l'arbre qu'à des feuilles. De même, ils n'apparaissent pas explicitement dans l'affichage de l'arbre. Les nœuds où commence ou fini une inclusion sont juste affichés avec des symboles particuliers (figure).

Si le nœud sélectionné dans l'arbre se trouve au dessus de l'inclusion, la ligne `inclure Fichier...` apparaît dans la fenêtre de texte ; si le nœud sélectionné est le nœud où a eu lieu l'inclusion ou si c'est un descendant de ce nœud, l'information contenue dans le fichier inclus est affichée (figure 14).

5. Lecture de données textuelles en entrée de l'interface

Les fichiers lus en entrée des logiciels sont au format texte. L'arborescence des données lue par l'interface générée par l'outil GABI est actuellement dans un format sérialisé. L'interface sauvegarde les données dans un format arbre sérialisé, directement lisible en entrée, ou dans un format texte lisible par le logiciel de calcul (figure 15).

Il est cependant souhaitable que l'interface puisse lire des fichiers texte en entrée.

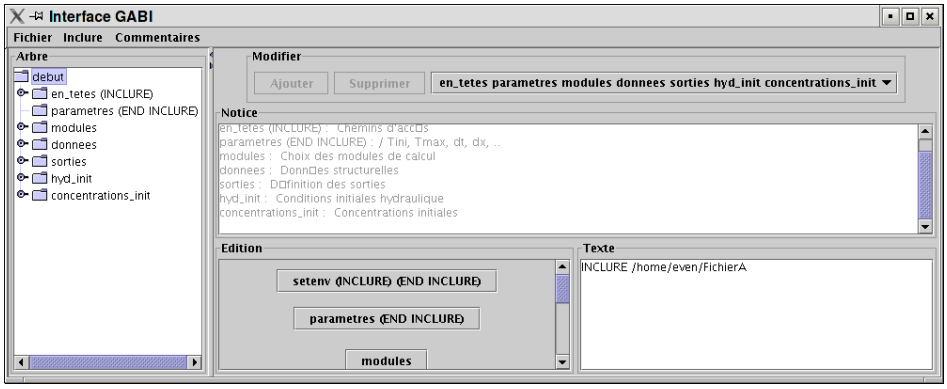

Figure 13: *Visualisation de l'arbre après l'inclusion des informations d'en-tête dans un fichier* `FichierA`*; affichage de l'information textuelle* `inclure /home/even/FichierA` *si sélection à la racine.*

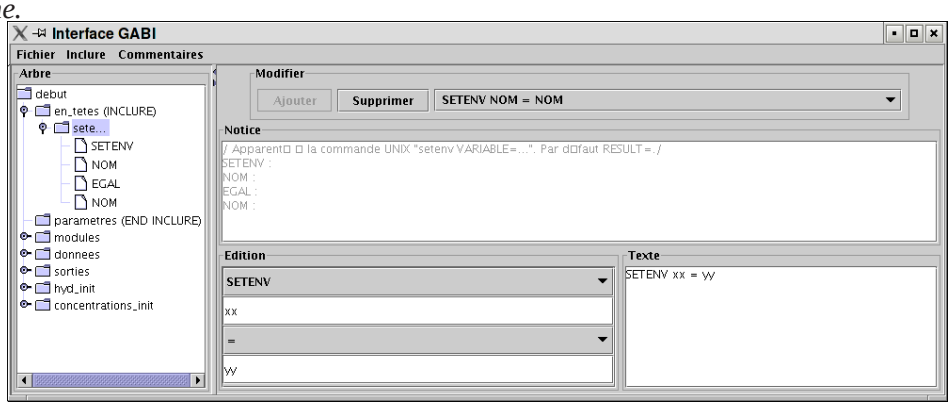

Figure 14: *Affichage de l'information contenue dans le fichier si sélection d'un nœud sous l'inclusion.*

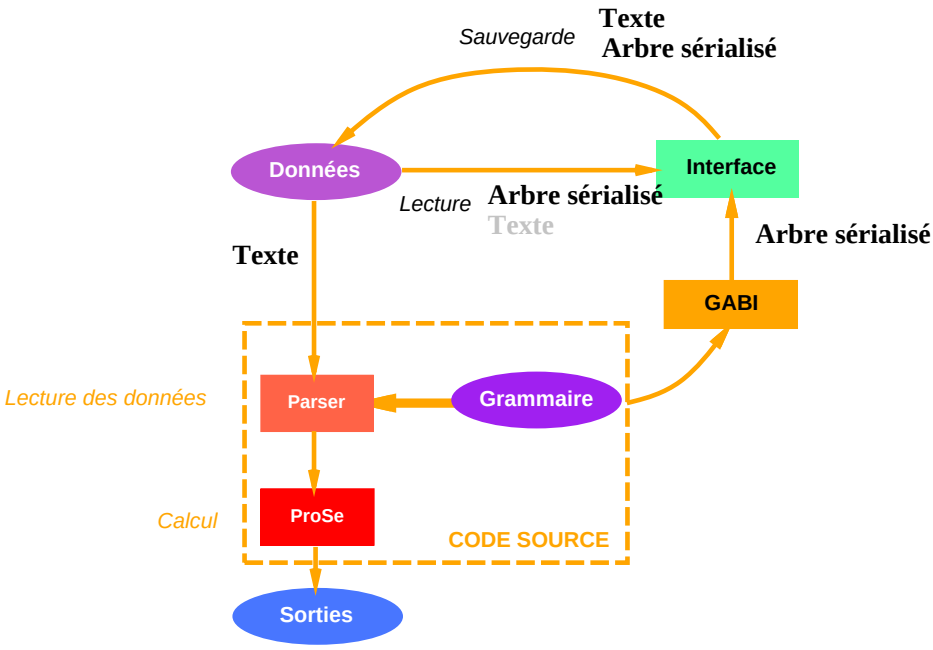

Figure 15: *Représentation des formats de données manipulés par le logiciel scientifique et son interface.*

Pour mettre en œuvre cette nouvelle fonctionnalité, deux nouveaux fichiers `jlex` et `cup` sont désormais crées lors de l'analyse des fichiers `.lex` et `.yacc` contenant la description de la grammaire initiale (figure 16). Ils sont la transcription, en java, de la grammaire initiale. L'analyse des fichiers `.lex` et `.yacc` est elle même réalisée à l'aide de parsers java `lexer.jlex` et `lexer.cup` pour l'analyse du fichier `.lex` et `parser.jlex` et `parser.cup` pour le fichier `.yacc`.

Les fichiers `jlex` et `cup` nouvellement crées à chaque analyse des fichiers `.lex` et `.yacc`, sont compilés par l'outil GABI.

6. Conclusion

L'outil GABI est aujourd'hui opérationnel et a été appliqué sur le logiciel ProSe. La prise en compte des fonctions commentaires et d'inclusion ont été améliorées. Des améliorations pourront cependant être encore apportées dans la gestion des inclusions pour gérer au mieux une information abondante : fonctions de repli ou d'ouverture de l'arbre des données commandées par les tags d'inclusion et non plus seulement au niveau des nœuds, utilisation des codes de couleurs, ...

La fonction permettant une lecture de fichiers texte en entrée de l'interface a été développées.

D'autres fonctionnalités peuvent et devront encore être apportées à l'outil ; cependant l'accent sera mis à l'avenir sur les aspects présentation et ergonomie de l'interface.

Bibliographie

Bonniez, S., 2001. Générateur automatique d'interfaces graphiques pour logiciels scientifiques. Rapport de stage 3A - ENSTB.

Even, S., N. Flipo, M. Poulin, S. Bonniez, et R. Keryell, 2002. Développements opérationnels des outils de modélisation de l'eau de la Seine : ProSe à tubes de courant, version 3. Rapport

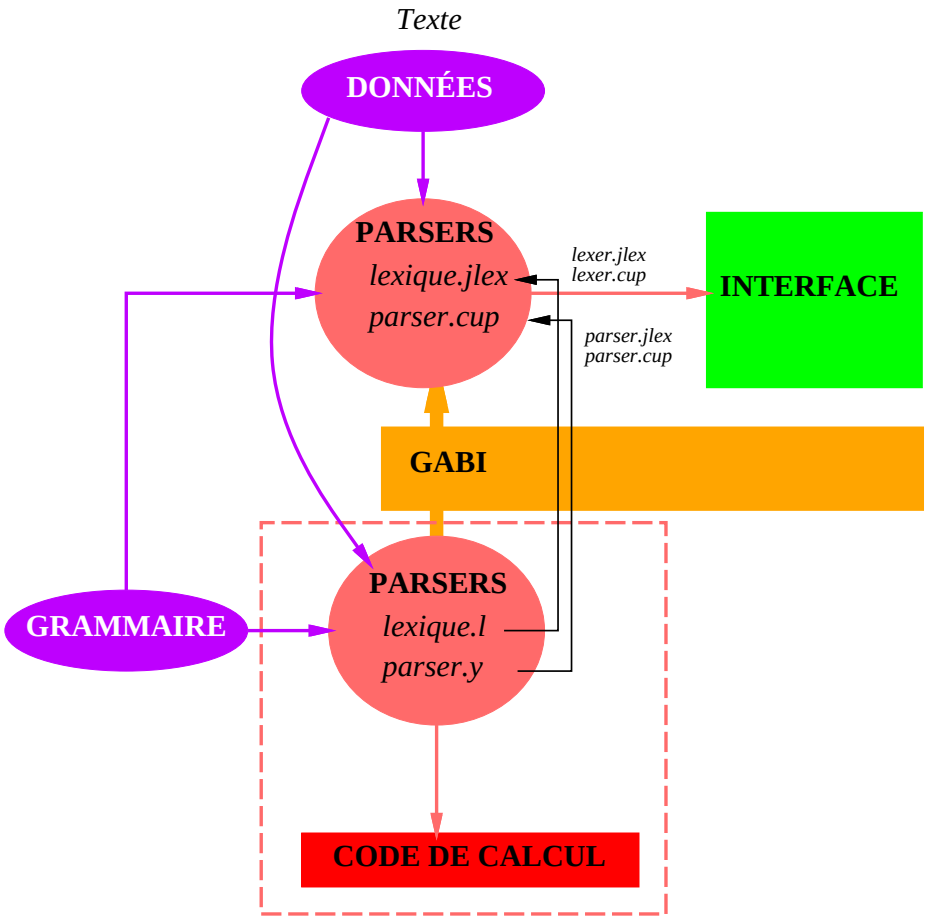

Figure 16: *Représentation graphique des différents niveaux d'analyse des données et des inter-relations entre les différents parsers.*

technique, Piren Seine, rapport d'activité 2002.

Even, S., R. Keryell, N. Flipo, et M. Poulin, 2003. Développements et interfaçages de PROSE 3.5. Rapport technique, Piren Seine, rapport d'activité 2003.

Even, S., R. Keryell, N. Flipo, et M. Poulin, 2004. Développements et interfaçages de PROSE 3.5, Contribution du Centre d'Informatique Géologique de l'Ecole des Mines de Paris au programme de recherche Piren Seine. Rapport technique, École des Mines de Paris LHM/RD/04/04.

Schlee, M., 2002. Generative Programming of Graphical User Interfaces. Ph. D. thesis, Fachhochschule Kaiserlautern.