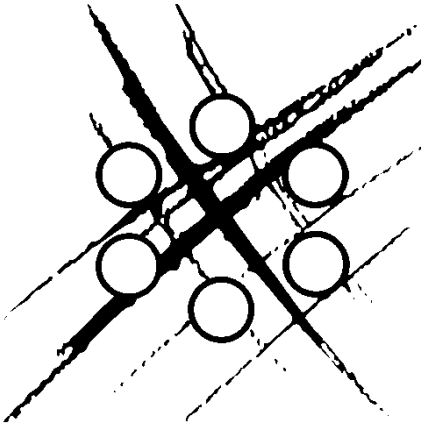
DIRECTION DE L'ENERGIE NUCLEAIRE

DIRECTION DELEGUEE AUX ACTIVITES NUCLEAIRES DE SACLAY

DEPARTEMENT DE MODELISATION DES SYSTEMES ET STRUCTURES

SERVICE D'ETUDES MECANIQUES ET THERMIQUES



Module EPXDATA
Corrections/améliorations et interprétation de la syntaxe de commande du programme EUROPLEXUS pour la relecture de jeux de données existants

DEN/DANS/DM2S/SEMT/DYN/RT/12-029/A

V. Faucher, E. Adam[1]

[1] DEN/DANS/DM2S/STMF/LGLS

Commissariat à l'énergie atomique et aux énergies alternatives
DEN/DANS/DM2S/SEMT/DYN
Centre de Saclay BAT 607 - PC 117
91191 Gif/Yvette cedex - France
Tél. : 01 69 08 23 60 Fax : 01 69 08 76 19 Courriel : catherine.borgoltz@cea.fr
Établissement public à caractère industriel et commercial
RCS Paris B 775 685 019

Module EPXDATA
Corrections/améliorations et interprétation de la syntaxe de commande du programme EUROPLEXUS pour la relecture de jeux de données existants

NIVEAU DE CONFIDENTIALITE				
DO	DR	CCEA	CD	SD
X				

PARTENAIRES/CLIENTS	REFERENCE DE L'ACCORD OU DU CONTRAT	TYPE D'ACTION
EDF	11-445	CE

REFERENCES INTERNES CEA			
DIRECTION D'OBJECTIFS	PROGRAMME	PROJET	EOTP
DISN	Simulation	MECAN	A-MECAN-03-01

S'IL S'AGIT DU LIVRABLE D'UN JALON :

JALON	INTITULE DU JALON
LIVRABLE MAJEUR CPCH	Livrable n°3 de la fiche d'action MIDES - Procédure de relecture des jeux de données existants : description, exemples de mise en œuvre et conseils d'utilisation

S'agit-il d'une synthèse ? ☐ oui ☒ non

SUIVI DES VERSIONS			
INDICE	DATE	NATURE DE L'EVOLUTION	PAGES, CHAPITRES
A	26/11/2012	Document initial	

	NOM	FONCTION	VISA	DATE
REDACTEUR	V. FAUCHER, E. ADAM			26/11/2012
VERIFICATEUR	F. BLIARD			26/11/2012
APPROBATEUR	T. LAPORTE	Chef de laboratoire		26/11/2012
AUTRE VISA				
ÉMETTEUR	X. AVERTY	Chef de service		26/11/2012

Module EPXDATA
Corrections/améliorations et interprétation de la syntaxe de commande du programme EUROPLEXUS pour la relecture de jeux de données existants

MOTS CLEFS

EUROPLEXUS, module EPXDATA, SALOME, import et export de jeux de données

RESUME / CONCLUSIONS

Le module EPXDATA pour la plateforme SALOME est développé depuis 2010 dans le cadre de la fiche MECAN/MIDES. Il a pour objectif de mettre à disposition des utilisateurs du programme EUROPLEXUS une solution intégrant géométrie, maillage et mise en données sous forme graphique. Les deux premiers items de la liste précédente sont assurés par les modules GEOM et SMESH de la plateforme, le présent travail portant sur le troisième item.

Un quatrième ingrédient pour une intégration complète d'EPX dans un environnement graphique correspond aux outils de post-traitement, fonctionnels pour le code via le support qu'il offre des formats de sortie PVTK et MED, permettant l'utilisation des modules VISU et, préférentiellement, PARAVIS de la plateforme.

Enfin, ces outils peuvent être complétés par un gestionnaire d'étude à destination des utilisateurs en environnement industriel, pour simplifier et automatiser la soumission de calculs EPX mis en données et dépouillés sur la base des modules cités ci-dessus. EDF propose dans la plateforme SALOME-MECA, dérivée de la plateforme SALOME pour le calcul mécanique, un module EPX assurant cette fonction. La compatibilité du module EPXDATA est assurée sans réserve avec la plateforme SALOME-MECA à tous les stades de son développement, pour compléter un ensemble d'outils complémentaires couvrant tous les besoins identifiés des utilisateurs industriels du code EPX.

Les actions de développements dans la fiche MECAN/MIDES ont porté en 2010 et 2011 sur la construction *ex nihilo* d'un jeu de données pour EPX à partir des commandes disponibles figurant dans le manuel utilisateur du programme. Un rappel de ces développements et des orientations stratégiques auxquels ils répondent est proposé en première partie du présent document.

La suite du document est consacrée à l'action complémentaire de la première, à savoir l'interprétation de jeux de données EPX existants, sous la forme d'un fichier de maillage et d'un fichier de commande textuel contenant des instructions dans la syntaxe propre du programme. Ce point est fondamental pour permettre aux utilisateurs de modifier graphiquement des jeux de données correspondants à des activités avec EPX antérieures à la disponibilité du module EPXDATA ou issus de la base de connaissance du programme (tests de non-régression et tests de validation) mis à leur disposition pour servir d'exemples d'utilisation du programme.

La dernière partie du document est dédiée aux tâches de corrections/améliorations du module effectuées en réponse aux remarques des utilisateurs de la version *alpha* formulées lors du pré-déploiement du module en 2011.

RESUME / CONCLUSIONS de niveau DO en cas de niveau confidentialité supérieur du document

Sans objet

Diffusion initiale interne CEA

Diffusion électronique du document complet

CEA/DEN/DANS :

DM2S/SEMT/DYN	T. Laporte, F. Bliard, H. Bung, P. Galon, V. Faucher
DM2S/SEMT/LM2S	S. Pascal, O. Fandeur, J.L. Fayard
DM2S/SEMT/DIR	X. Averty, P. Verpeaux
DM2S/STMF/LGLS	V. Bergeaud, M. Tajchman
DM2S/STMF/LMES	E. Adam
DM2S/DIR	E. Proust, C. Chaigneau, M.P. Bohar

CEA/DEN :

DISN/Simulation	D. Caruge, J.P. Deffain
DISN/Gen2&3	M. Durin
DPIE/SA2P	Y. Kayser

Diffusion résumé

CEA/DEN/DANS :

DM2S/SEMT/BCCR
DM2S/SEMT/EMSI
DM2S/SEMT/LISN
DM2S/SEMT/LM2S
DM2S/SEMT/LTA

DM2S/STMF
DM2S/SERMA

DPC/SCCME

DMN/SRMA
DMN/SEMI

Diffusion initiale externe CEA

Diffusion électronique du document complet

EDF :

R&D/AMA	F. Waeckel, V. Godard, F. Crouzet, S. Potapov, C. Durand, F. Daude. A. Assire, I. Fournier
R&D/DIR	O. Marchand

SOMMAIRE

Liste des tableaux

Liste des figures

1 Eléments de contexte

Le module EPXDATA pour la plateforme SALOME est développé depuis 2010 dans le cadre de la fiche MECAN/MIDES. Il a pour objectif de mettre à disposition des utilisateurs du programme EUROPLEXUS (EPX dans la suite du texte) une solution intégrant géométrie, maillage et mise en données sous forme graphique. Les deux premiers items de la liste précédente sont assurés par les modules GEOM et SMESH de la plateforme, le présent travail portant sur le troisième item.

Un quatrième ingrédient pour une intégration complète d'EPX dans un environnement graphique correspond aux outils de post-traitement, fonctionnels pour le code via le support qu'il offre des formats de sortie PVTK et MED, permettant l'utilisation des modules VISU et, préférentiellement, PARAVIS de la plateforme.

Enfin, ces outils peuvent être complétés par un gestionnaire d'étude à destination des utilisateurs en environnement industriel, pour simplifier et automatiser la soumission de calculs EPX mis en données et dépouillés sur la base des modules cités ci-dessus. EDF propose dans la plateforme SALOME-MECA, dérivée de la plateforme SALOME pour le calcul mécanique, un module EPX assurant cette fonction. La compatibilité du module EPXDATA est assurée sans réserve avec la plateforme SALOME-MECA à tous les stades de son développement, pour compléter un ensemble d'outils complémentaires couvrant tous les besoins identifiés des utilisateurs industriels du code EPX.

Les actions de développements dans la fiche MECAN/MIDES ont porté en 2010 et 2011 sur la construction *ex-nihilo* d'un jeu de données pour EPX à partir des commandes disponibles figurant dans le manuel utilisateur du programme. Un rappel de ces développements et des orientations stratégiques auxquels ils répondent est proposé en première partie du présent document.

La suite du document est consacrée à l'action complémentaire de la première, à savoir l'interprétation de jeux de données EPX existants, sous la forme d'un fichier de maillage et d'un fichier de commande textuel contenant des instructions dans la syntaxe propre du programme. Ce point est fondamental pour permettre aux utilisateurs de modifier graphiquement des jeux de données correspondants à des activités avec EPX antérieures à la disponibilité du module EPXDATA ou issus de la base de connaissance du programme (tests de non-régression et tests de validation) mis à leur disposition pour servir d'exemples d'utilisation du programme.

La dernière partie du document est dédiée aux tâches de corrections/améliorations du module effectuées en réponse aux remarques des utilisateurs de la version *alpha* formulées dans [1].

2 Rappels des orientations stratégiques pour la conception du module EPXDATA

2.1 Considérations stratégiques

Dans le cadre de la conception d'un outil graphique de mise en données pour un programme de simulation tel qu'EPX, les risques principaux identifiés au moment de la définition du cahier des charges pour le module EPXDATA sont :

- ✓ la compatibilité d'un nombre limité, voire restreint, de commandes figurant dans le manuel EPX, limitant les possibilités d'utilisation du programme sur ce mode et conduisant fréquemment à sa marginalisation,
- ✓ le non-respect du lien systématique entre les évolutions du programme et de son manuel et l'outil de mise en données, pouvant conduire à une obsolescence de l'outil si des divergences importantes s'accumulent entre le manuel et l'outil,

Sur la base de ces risques et compte-tenu des ressources disponibles pour ces développements et leur maintenance dans le temps (soit *0.3 homme.an par an depuis 2010*), les orientations stratégiques suivantes ont été établies conjointement par les laboratoires DEN/DANS/DM2S/SEMT/DYN, principal contributeur avec le *Joint Research Center* de la Commission Européenne pour l'environnement logiciel EPX, et DEN/DANS/DM2S/STMF/LGLS, expert dans le domaine des IHM pour les codes de simulation.

1. La référence des commandes pour une version d'EPX fournie aux utilisateurs est donnée par le manuel qui lui est associé et l'IHM doit s'y conformer et respecter la sémantique, parfois complexe, des commandes.

Une restructuration majeure de la syntaxe des fichiers EPX n'est pas envisageable compte-tenu des moyens attribués à ces actions, de la compatibilité à conserver avec les jeux de données existants et du refus d'une situation transitoire à l'issue incertaine en raison du volume de travail.

2. L'Atelier Logiciel EPX [2] assure la gestion de configuration du code source au format LaTeX du manuel : c'est ce support textuel qui contient la syntaxe de référence pour les commandes EPX et qui doit servir d'entrée à toute procédure de construction du catalogue des commandes, sans modifier en profondeur la procédure de rédaction du manuel par les développeurs du consortium.
3. L'intégration des évolutions du manuel au contenu de l'IHM doit être automatique et doit pouvoir être effectuée aussi souvent que nécessaire sans mobiliser de ressources spécifiques.
4. Le contenu de référence d'un jeu de données EPX doit rester le fichier de commande, associé à un fichier de maillage éventuel.

   Tout autre format intermédiaire, par exemple une sauvegarde d'étude dans SALOME, peut présenter des risques d'obsolescence et n'est pas directement compatible avec la base des tests existants dans le consortium EPX.

Ceci conduit à la démarche logique suivante pour le module EPXDATA dans SALOME :

1. une interprétation du code source LaTeX du manuel construit le catalogue des commandes disponibles et de leur syntaxe pour une version donnée d'EPX,
2. le catalogue est transposé automatiquement en menus et boîtes de dialogue pour SALOME à l'aide d'un outil logiciel dédié à des tâches de cette nature, à savoir XDATA, développé et maintenu au laboratoire DEN/DANS/DM2S/STMF/LGLS [3],
3. des commandes sont ajoutées via XDATA pour l'export et l'import de fichiers de données textuels utilisables par EPX, sur la base de la structure de données générée par XDATA.

La procédure d'interprétation du manuel et l'export de fichiers au format EPX à partir de données renseignées interactivement dans SALOME ont été mis en œuvre et évalués en 2011 [4]. Le présent rapport est dédié à la tâche d'import des fichiers EPX, ainsi qu'à la description des tâches principales d'amélioration/correction du module réalisées en 2012

Le Tableau 2-1 résume les contraintes et les orientations stratégiques associées.

Contraintes	**Stratégie**	**Solution technique**
Assurer une **adéquation maximale entre le manuel EPX** rédigé par les développeurs **et les commandes du module EPXDATA**.	Fonder l'outil sur une **interprétation systématique du code source LaTeX du manuel EPX**	**Scripts python construisant le catalogue des commandes** à partir des blocs de syntaxe extrait du code LaTeX
Ressources limitées pour les tâches de développement/maintenance.		
Mettre à jour automatiquement le module en cas de modification apportée au manuel pour éviter toute obsolescence.	Générer les menus et boîtes de dialogue dans SALOME à l'aide d'un **outil maîtrisé et pérenne**	Utilisation du **logiciel XDATA** développé et maintenu au laboratoire DEN/DANS/DM2S/STMF/LGLS pour l'IHM dans SALOME. **Génération automatique des classes XDATA à partir du catalogue des commandes** par un script python.

Contraintes	Stratégie	Solution technique
Conserver le fichier de données au format EPX, associé à un fichier de maillage éventuel, **comme élément de référence** d'un jeu de données EPX.	Assurer **l'import et l'export de fichiers au format EPX** dans l'outil EPXDATA	Scripts python réalisant ces fonctions. Eléments logiciels additionnels nécessaires pour l'import, décrits dans la suite du rapport.

Tableau 2-1 : Stratégie de développement du module EPXDATA

2.2 Inconvénients connus de la stratégie retenue et voies d'amélioration

Le choix, justifié ci-dessus, du code source LaTeX du manuel EPX comme référence pour la syntaxe des commandes à intégrer à l'IHM conduit aux réserves suivantes.

1. Les règles de mise en données pour une commande EPX sont peu contraignantes pour le développeur et peuvent conduire à des syntaxes dont la sémantique est très riche, avec de nombreux niveaux d'embranchement (cf. § 3.2.1) pour un exemple illustrant cette situation).

   Si l'expression des commandes reste le plus souvent relativement compacte dans le fichier au format EPX, grâce à l'analyseur syntaxique intégré au programme, une sémantique complexe nuit à l'ergonomie de l'outil, forçant l'utilisation de nombreuses boîtes de dialogue successives pour reproduire les embranchements logiques. C'est une situation fréquemment observée par les utilisateurs des versions de développement du module (cf. [1])

2. La procédure de rédaction des pages de manuel par les développeurs du consortium EPX ne comprend aucune relecture et contrôle de la qualité.

   D'une part, cela peut conduire à des non-conformités dans la rédaction qui peuvent induire en erreur l'utilisateur du module EPXDATA ou produire des fichiers de données au format EPX non-fonctionnels après export.

   D'autre part, une syntaxe correcte rédigée dans l'optique d'une lecture humaine peut s'avérer maladroite dans le cas d'une interprétation systématique, conduisant à une ergonomie défaillante du module (cf. [4]).

Les voies d'amélioration envisagées pour pallier ces inconvénients et leurs limites sont alors les suivantes.

1. Compte-tenu de la généralité et de la robustesse attendue pour le module EPXDATA, la mise en données *ex nihilo* d'un cas de calcul sans aucune hypothèse reste un processus entaché de lourdeur, sur laquelle on peut modérément influer par des améliorations ponctuelles dans l'ergonomie de l'IHM générée par XDATA.

   Une solution plus pertinente s'appuyant sur le travail réalisé est de mettre en œuvre, à partir d'hypothèses faites sur la modélisation utilisée dans le cas de calcul, des assistants graphiques (*wizards*) rassemblant le renseignement de plusieurs commandes avec des boîtes de dialogue synthétiques. Cette stratégie est adaptée pour des unités d'ingénierie produisant de nombreux calculs de nature similaire.

   De tels *wizards* peuvent être définis par exemple pour des calculs de structures typiques avec prise en compte du contact, pour des calculs typiques en interaction fluide-structure, etc…

   Il convient de souligner d'une part que la simplicité d'un assistant est inévitablement inversement proportionnelle à sa portée et d'autre part, que l'ordre logique des développements est bien celui proposé, à savoir la construction d'une structure de mise en données générique sur laquelle viennent se greffer des assistants. Une stratégie inverse ne permettrait pas d'assurer l'impératif de portée et de pérennité pour le module donné au § 2.1. De plus, disposer d'un outil générique permet de compléter l'utilisation d'un *wizard* par des modifications/compléments sur le jeu de données sans quitter l'environnement graphique de SALOME.

2. Dans le cadre de la mise à jour du Plan Qualité Logiciel EPX (cf. [5] pour la version actuelle du document), un processus de vérification de la conformité de l'IHM associée à la rédaction du manuel d'une nouvelle commande peut être ajouté.

   Ce point sera proposé au Comité Technique du Consortium EPX lors de sa prochaine réunion (décembre 2012).

   Les règles d'amélioration de la rédaction de la syntaxe des commandes pour une meilleure interprétation automatique sont connues et sont diffusées parmi les développeurs. Le travail de reprise des syntaxes existantes défaillantes est en cours, selon un calendrier difficile à maîtriser totalement et intégré à l'activité de support à l'utilisation du module EPXDATA.

Les considérations ci-dessus sont résumées dans le Tableau 2-2.

Inconvénient	Voie d'amélioration	Remarque(s)
Lourdeur de la mise en données en raison de la **sémantique complexe** des commandes EPX.	Remplacer la mise en données générique d'un cas *ex nihilo* par des **assistants spécialisés**.	La perte de généralité peut être compensée par des modifications ponctuelles du fichier de commande produit par un assistant à l'aide des fonctionnalités génériques du module.
Absence de contrôle de la qualité des pages de manuel servant de base à l'IHM pour les commandes EPX.	Ajout d'un **processus de vérification de la conformité de l'IHM** lors de la rédaction d'une nouvelle page de manuel ou la modification d'une page existante.	Règles connues et en cours de diffusion parmi les développeurs EPX pour la rédaction des pages de manuel pour assurer une interprétation correcte des commandes. Travail en cours de correction des syntaxes existantes dont l'interprétation pour l'IHM ne donne pas satisfaction.

Tableau 2-2 : Inconvénients et voies d'amélioration pour le module EPXDATA

3 Interprétation de la syntaxe EPX et relecture des jeux de données existants dans le module EPXDATA

3.1 Principe

Pour renseigner les commandes correspondant à un jeu de données EPX existant dans l'IHM de SALOME produite par XDATA, le principe majeur est la rédaction d'un script python intermédiaire contenant la déclaration des classes XDATA utilisées dans le fichier de données EPX, avec les paramètres numériques fournis par l'utilisateur. L'exécution de ce script dans l'environnement du module EPXDATA reconstruit les objets correspondant au contenu du jeu de données, permettant ainsi leur modification comme s'ils avaient été générés interactivement.

Un exemple de tel script python peut être obtenu aisément depuis l'IHM dans SALOME en exportant une étude créée dans le module EPXDATA au format python (cf. Annexe 1).

3.1.1 Structure du fichier de données EPX et analyseur syntaxique intégré au code

Le fichier de données au format EPX se présente comme une suite de mots-clés et de valeurs, sans délimitation entre les différentes directives du programme (affectation des éléments, déclarations des matériaux et des conditions aux limites, etc…), en dehors de la mise en forme et des commentaires apportés par l'utilisateur et non-normalisés (cf. Figure 3-1 pour un court exemple).

```
Impact sur un corps creux avec auto-contact
```

```
ECHO
MEDL
'/home/europlex/Etudes_EPX/tests_EPX_Salome/bm_contact_auto.med'

TRID

DIME
   Q4GS 2080
   MXLI 100
   GLIS 100 200
   BLOQ 500 DEPL 50 FCOEF 1 LIAI 500
TERM

GEOM Q4GS S1 TERM

COMPLEMENT EPAIS 9.14E-4 LECT S1 TERM

MATE VMIS ISOT RO 7800 YOUNG 2.E11 NU .3 ELAS 2.5E8
          TRAC 9   2.5E8    0.00125
                   4.0E8    0.02
                   5.5E8    0.1
                   6.25E8   0.2
                   6.5E8    0.3
                   6.55E8   0.4
                   6.55E8   0.5
                   6.55E8   1.
                   6.55E8   100.
          LECT S1 TERM

LINK COUP
   BLOQ 123456 LECT LGP1 TERM
   BLOQ 12456  LECT LGP2 TERM
   VITE 3  -15. FONC 1 LECT LGP2 TERM

LINK DECO
   GLIS 8
          PENA SELF PFSI 1.E-1 PGAP 1.E-3
          CMAI LECT S11 TERM INTE LECT PI1 TERM CESC LECT S1 TERM
          PENA SELF PFSI 1.E-1 PGAP 1.E-3
          CMAI LECT S11 TERM INTE LECT PE1 TERM CESC LECT S1 TERM
          PENA SELF PFSI 1.E-1 PGAP 1.E-3
          CMAI LECT S12 TERM INTE LECT PI1 TERM CESC LECT S1 TERM
          PENA SELF PFSI 1.E-1 PGAP 1.E-3
          CMAI LECT S12 TERM INTE LECT PE2 TERM CESC LECT S1 TERM
          PENA SELF PFSI 1.E-1 PGAP 1.E-3
          CMAI LECT S13 TERM INTE LECT PI1 TERM CESC LECT S1 TERM
          PENA SELF PFSI 1.E-1 PGAP 1.E-3
          CMAI LECT S13 TERM INTE LECT PE3 TERM CESC LECT S1 TERM
          PENA SELF PFSI 1.E-1 PGAP 1.E-3
          CMAI LECT S14 TERM INTE LECT PI1 TERM CESC LECT S1 TERM
          PENA SELF PFSI 1.E-1 PGAP 1.E-3
          CMAI LECT S14 TERM INTE LECT PE4 TERM CESC LECT S1 TERM

FONC 1 TABL 2 0. 1. 1. 1.

ECRI TFREQ 1.E-4 NOPO NOEL
     FICH ALIT TFREQ 1.E-6
          ELEM LECT 176 1615 1839 TERM
     FICH PVTK TFREQ 1.2E-4 GROUP AUTO
          VARI DEPL VITE ACCE FEXT ECRO

OPTION NOTEST STEP IO
```

```
CALCUL TINI 0. TEND 6.E-3

SUITE

FIN
```

Figure 3-1 : Exemple de fichier de commandes au format EPX

Crash d'un corps creux avec auto-contact – En vert, les mots-clés introduisant de nouvelles directives dans EPX

Le décodage des données est réalisé dans EPX sur la base d'une analyse syntaxique intégrée à la construction de la structure de données de calcul : le fichier est lu et décodé séquentiellement, avec un catalogue des commandes contenu implicitement dans les routines du code. Ainsi, la sémantique exacte des commandes est invisible dans le jeu de données.

Extraire cette sémantique des routines d'EPX réalisant la lecture du jeu de données est une tâche complexe, car ces routines sont très nombreuses et ne sont pas prévues pour l'écriture des données lues à un format correspondant à un catalogue de commandes, en indiquant tous les embranchements logiques nécessaires. Il en résulterait un volume de développement important, avec le risque sous-jacent exprimé au § 2.1 d'une implémentation incomplète et difficile à maintenir.

Le recours au catalogue *implicite* contenu dans le code EPX est ainsi exclu pour préparer les données pour le module EPXDATA.

3.1.2 Stratégie retenue

Pour interpréter un jeu de données EPX, il est nécessaire de confronter la syntaxe du fichier de commandes à un catalogue explicitement connu, au moyen d'un outil logiciel compact aisé à maintenir.

Un tel catalogue est disponible après interprétation des blocs de syntaxe de la documentation au moment de l'installation du module EPXDATA. Il est logiquement en adéquation complète avec le catalogue *implicite* évoqué ci-dessus, toute divergence devant être considérée comme une erreur de rédaction du manuel et traitée comme une anomalie classique dans l'Atelier Logiciel EPX. Quelques commandes de ce catalogue sont données à titre d'exemples sur la Figure 3-2.

```
GROUPC1MATERIALS__SOLIDMATERIALS__LINEARELASTICITY__content
K_ROOT = LINE
|  K_KEYWORD = LINE
|  |  K_JUST_ONE = RO
|  |  |  K_NODE = RO
|  |  |  |  K_KEYWORD = RO
|  |  |  |  |  K_NAME = rho
|  |  |  |  K_KEYWORD = YOUN
|  |  |  |  |  K_NAME = young
|  |  |  |  K_KEYWORD = NU
|  |  |  |  |  K_NAME = nu
|  |  |  |  K_OPTIONAL = VISC
|  |  |  |  |  K_KEYWORD = VISC
|  |  |  |  |  |  K_NAME = visc
|  |  |  |  |  K_KEYWORD = KRAY
|  |  |  |  |  |  K_NAME = kray
|  |  |  |  |  K_KEYWORD = MRAY
|  |  |  |  |  |  K_NAME = mray
|  |  K_PROCEDURE = LECTURE
```

(a) Catalogue de la commande de déclaration du matériau élastique linéaire (syntaxe MATE LINE…)

```
GROUPDLINKS__RELATIONS__content
K_ROOT = RELA
|  K_KEYWORD = RELA
|  |  K_ITER = nrel
|  |  |  K_NAME = ngroup
|  |  |  K_PARENTH = nrel
|  |  |  |  K_NAME = nrel
|  |  |  |  K_ITER = coef
|  |  |  |  |  K_NAME = nterm
|  |  |  |  |  K_PARENTH = coef
|  |  |  |  |  |  K_NAME = coef
|  |  |  |  |  |  K_NAME = icomp
```

```
|   |   |   |   |   |   K_ONE_MAXI = nuneu
|   |   |   |   |   |   |  K_NODE = nuneu
|   |   |   |   |   |   |   |  K_NAME = nuneu
|   |   |   |   |   |   |   |  K_NAME = ipas
|   |   |   |   |   |   |   |  K_NODE = LECTURE
|   |   |   |   |   |   |   |   |  K_PROCEDURE = LECTURE
|   |   |   |   |   |   |   |   |  K_OPTIONAL = SHIF
|   |   |   |   |   |   |   |   |   |  K_KEYWORD = SHIF
|   |   |   |   |   |   |   |   |   |   |  K_NAME = s
|   |   |   |   K_KEYWORD = EGAL
|   |   |   |   |   K_PROCEDURE = LECDDL
|   |   |   |   |   K_PROCEDURE = LECTURE
```

(b) Catalogue de la commande de déclaration des relations cinématiques entre degrés de liberté (syntaxe LINK COUP RELA…)

Figure 3-2 : Exemples de commandes dans le catalogue produit par l'interprétation du manuel d'EPX lors de l'installation du module EPXDATA

Les différents opérateurs sémantiques (mots-clés, embranchements conditionnels, listes, valeurs numériques…) sont distingués avec des couleurs différentes

Pour simplifier le développement et la maintenance de l'outil confrontant la syntaxe du fichier de commandes EPX au catalogue, il est convenu en collaboration avec le laboratoire CEA/DEN/DANS/DM2S/STMF/LGLS de séparer la tâche d'extraction de la sémantique du fichier EPX et la tâche d'écriture du script python reprenant cette sémantique pour créer les objets correspondant dans l'environnement SALOME/XDATA.

Cette séparation est faite en recourant à un fichier intermédiaire au format XML contenant les commandes interprétées et présentées selon la sémantique du catalogue. La mise en œuvre d'un outil pour la génération de ce fichier, logiquement nommé *epx2xml*, est de la responsabilité du laboratoire DEN/DANS/DM2S/SEMT/DYN. Le fichier XML est ensuite relu par un script python intégré au module EPXDATA, nommé *xml2py.py*, écrit au laboratoire DEN/DANS/DM2S/STMF/LGLS, détenant la compétence pour ce type de développements.

La Figure 3-3 donne une vision synthétique de la stratégie adoptée entre les deux laboratoires.

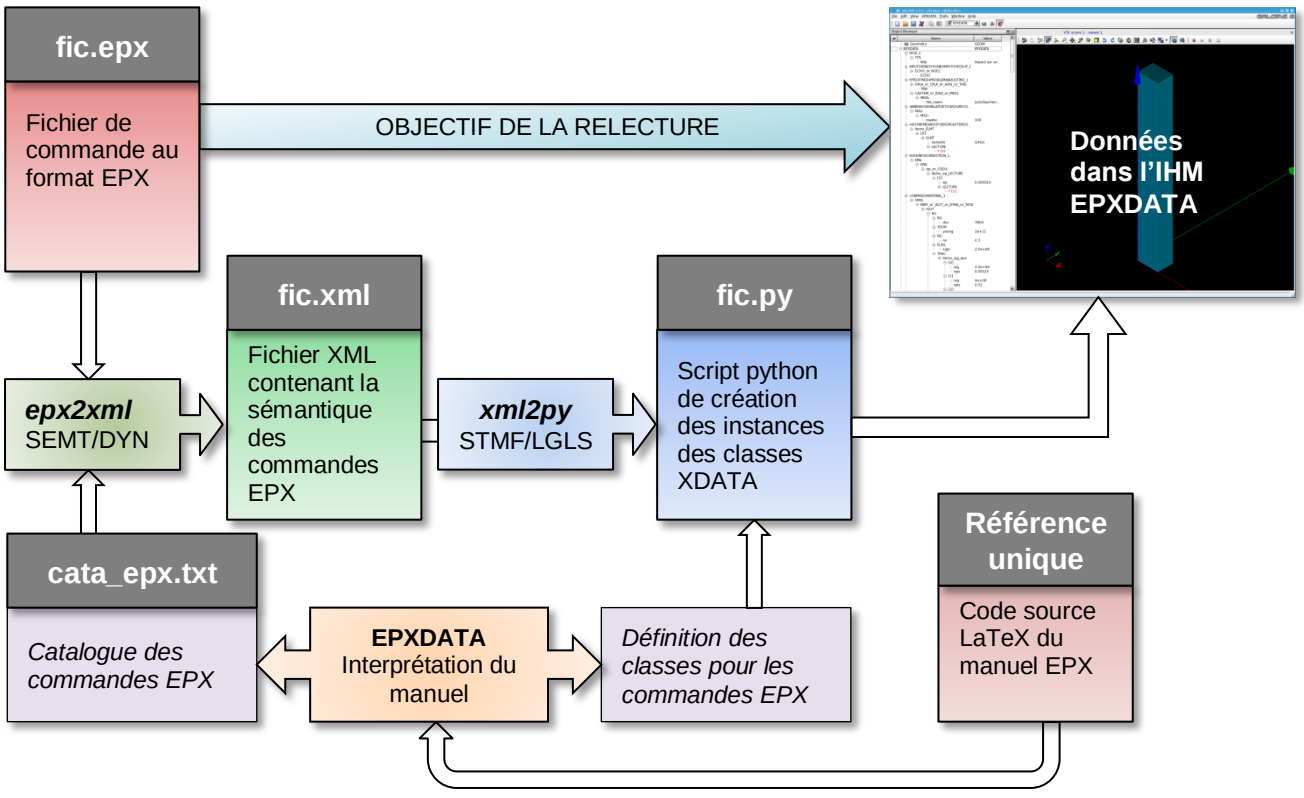


Figure 3-3 : Stratégie collaborative pour la relecture des jeux de données EPX

3.2 Programme *epx2xml*

3.2.1 Description

epx2xml est un programme FORTRAN, utilisant les routines de base de l'analyseur syntaxique d'EPX pour parcourir le fichier de commande. La syntaxe d'utilisation est la suivante (à partir des noms de fichiers génériques de la Figure 3-3) :

```
epx2xml fic.epx cata_epx.txt
```

Le fonctionnement logique est le suivant :

1. lecture d'un mot-clé dans le fichier de commande EPX (en dehors du cas particulier du titre, chaque directive dans le programme commence par un mot-clé) et identification de la directive correspondante dans le catalogue fourni en argument,
2. au sein de la directive trouvée, lecture séquentielle des mots-clés et des valeurs numériques ou textuelles et confrontation aux entrées possibles proposées par le catalogue, en parcourant les embranchements en respectant leurs règles propres le cas échéant (exclusions mutuelles notamment),
3. si pas de correspondance trouvée dans le catalogue pour une valeur numérique ou textuelle : erreur de syntaxe remontée à l'utilisateur,
4. si pas de correspondance trouvée pour un mot-clé, sortie de la directive et recherche d'une nouvelle directive commençant par le mot-clé courant.

L'étape 2 est le point-clé de l'algorithme, en raison de la multiplicité des embranchements possibles pour certaines commandes.

La Figure 3-4 donne la conversion au format XML d'un fichier au format EPX contenant uniquement les deux exemples de commandes présentés sur la Figure 3-2. La comparaison entre la compacité des commandes EPX et la complexité de la sémantique contenue dans leur équivalent XML illustre la difficulté associée à la conception du module EPXDATA depuis le début des travaux (cf. [4]).

```
Exemple epx2xml

MATE LINE RO 7800. YOUN 210.E9 NU 0.3 LECT STRUCT TERM

LINK COUP
     RELA   2  6  2   1.  3 LECT GRN1 TERM
                    -2.  3 LECT GRN2 TERM

            EGALITE 3 LECTURE GRN3 TERM

FIN
```

(a) Commandes EPX

```
<?xml version="1.0"?>
<EPX_xml_dataset version="0.1">
   <Command name="TITL" group="A" content="TITLE__content">
      <Keyword name="TITL">
         <Variable name="titl" type="string">Exemple epx2xml</Variable>
      </Keyword>
   </Command>

   <Command name="LINE" group="C1" content="GROUPC1MATERIALS__SOLIDMATERIALS__LINEARELASTICITY__content">
      <Keyword name="LINE">
         <Syntaxblock name="RO  ">
            <Node name="RO  ">
               <Keyword name="RO  ">
                  <Variable name="rho " type="integer">      7800</Variable>
               </Keyword>
               <Keyword name="YOUN">
                  <Variable name="youn" type="double"> 2.10000E+11</Variable>
               </Keyword>
               <Keyword name="NU  ">
```

```
                <Variable name="nu  " type="double"> 0.30000    </Variable>
             </Keyword>
          </Node>
       </Syntaxblock>
       <Procedure name="LECT">
          <Itemlist name="LECT">
             <Item>
                <Variable name="LECT" type="string">STRUCT</Variable>
             </Item>
          </Itemlist>
       </Procedure>
    </Keyword>
 </Command>

 <Command name="COUP" group="D" content="GROUPDLINKS__LINKCATEGORY__content">
    <Syntaxblock name="COUP">
       <Node name="COUP">
          <Keyword name="COUP">
          </Keyword>
       </Node>
    </Syntaxblock>
 </Command>

 <Command name="RELA" group="D" content="GROUPDLINKS__RELATIONS__content">
    <Keyword name="RELA">
       <Itemlist name="nrel">
          <Item>
             <Variable name="nrel" type="integer">         6</Variable>
             <Itemlist name="coef">
                <Item>
                   <Variable name="coef" type="integer">         1</Variable>
                   <Variable name="icom" type="integer">         3</Variable>
                   <Syntaxblock name="nune">
                      <Node name="LECT">
                         <Procedure name="LECT">
                            <Itemlist name="LECT">
                               <Item>
                                  <Variable name="LECT" type="string">GRN1</Variable>
                               </Item>
                            </Itemlist>
                         </Procedure>
                      </Node>
                   </Syntaxblock>
                </Item>
                <Item>
                   <Variable name="coef" type="integer">        -2</Variable>
                   <Variable name="icom" type="integer">         3</Variable>
                   <Syntaxblock name="nune">
                      <Node name="LECT">
                         <Procedure name="LECT">
                            <Itemlist name="LECT">
                               <Item>
                                  <Variable name="LECT" type="string">GRN2</Variable>
                               </Item>
                            </Itemlist>
                         </Procedure>
                      </Node>
                   </Syntaxblock>
                </Item>
             </Itemlist>
             <Keyword name="EGAL">
                <Variable name="dof1" type="integer">         0</Variable>
                <Variable name="dof2" type="integer">         0</Variable>
                <Variable name="dof3" type="integer">         1</Variable>
                <Variable name="dof4" type="integer">         0</Variable>
                <Variable name="dof5" type="integer">         0</Variable>
                <Variable name="dof6" type="integer">         0</Variable>
                <Variable name="dof7" type="integer">         0</Variable>
                <Procedure name="LECT">
                   <Itemlist name="LECT">
                      <Item>
                         <Variable name="LECT" type="string">GRN3</Variable>
```

```
                        </Item>
                      </Itemlist>
                    </Procedure>
                  </Keyword>
               </Item>
            </Itemlist>
         </Keyword>
      </Command>

</EPX_xml_dataset>
```

(b) Syntaxe XML correspondante

Figure 3-4 : Exemple de conversion du format EPX vers le format XML

3.2.2 Gestion des erreurs de lecture

Pour autoriser une rédaction défaillante des pages de manuel, pouvant conduire à des erreurs de catalogue et à une mise en défaut de l'interprétation de la syntaxe EPX, une erreur dans la confrontation entre les commandes et le catalogue ne provoque pas l'arrêt de l'interprétation.

La directive courante est ignorée dans la sortie XML et l'erreur est consignée dans un fichier `fic.log` accompagnant l'exécution du programme. Le mot-clé suivant est alors cherché et la confrontation reprend. Ceci permet d'interpréter le volume maximal de données contenu dans le fichier, même si ça n'assure pas que le contenu interprété reste fonctionnel au sens de l'exécution d'EPX dans le cas d'une incompatibilité entre les commandes EPX et le catalogue courant.

Même sans erreur dans le catalogue, ce cas peut être rencontré dans la mesure où certaines obsolescences dans la syntaxe EPX sont tolérées par le programme, pour conserver la compatibilité avec les tests de non-régression les plus anciens, alors qu'elles ne figurent plus dans le manuel et donc dans le catalogue, les rendant impossible à interpréter.

3.2.3 Gestion de configuration

Le code source du programme *epx2xml* est intégré à celui d'EPX, dont la gestion de configuration est assurée au laboratoire DEN/DANS/DM2S/SEMT/DYN via l'Atelier Logiciel utilisé dans le consortium EPX [2].

Il bénéficie ainsi du référentiel qualité associé au développement d'EPX [5]. Au moment de la compilation et de l'édition des liens, on utilise les outils de filtrage du code source disponibles sur l'Atelier Logiciel pour construire au choix un exécutable d'EPX ou d'*epx2xml*.

3.3 Script python *xml2py.py*

3.3.1 Description

La syntaxe d'utilisation est la suivante :

```
python xml2py.py fic.xml
```

Le script python *xml2py.py* interprète le contenu du fichier au format XML et écrit dans le fichier *fic.py* en sortie la syntaxe python correspondant à chacune des commandes du fichier. Pour chaque commande, chaque balise de type `Keyword`, `Node`, `Itemlist` ou `Variable` correspond à une instance de classe XDATA à créer, les autres balises étant simplement lues dans le fichier XML.

Pour chacune des balises concernées, le script recherche dans les classes du module EPXDATA la classe de même nom.

Les classes du module EPXDATA sont stockées par directive, chaque directive étant une page du manuel EPX. Le nom de chaque directive est construit à partir du titre de la page correspondante et de l'arborescence des commandes dans le manuel. Par exemple, la directive contenue sur la page de titre *LINK CATEGORY* dans le groupe *D* correspondant aux liaisons cinématiques (*LINKS*, cf. [6]) porte le nom `GROUPDLINKS__LINKCATEGORY`. Les classes XDATA correspondant aux commandes de cette directive sont alors écrites dans le fichier *GROUPDLINKS__LINKCATEGORY.py.*

Pour aider la recherche en identifiant le fichier contenant la classe, le nom de la directive concernée dans le manuel EPX est passé dans le fichier XML avec la balise `Command` via la chaîne `content` (cf. Figure 3-4).

Une fois la classe trouvée pour une balise donnée, la syntaxe python correspondant à la création d'une instance de cette classe avec les paramètres utilisateurs contenus dans le fichier XML, est écrite. La structure hiérarchique d'une commande dans le fichier XML se retrouve dans l'imbrication des créations d'instances dans la syntaxe python (cf. Figure 3-5).

La Figure 3-5 présente la conversion du fichier XML de la Figure 3-4 via le script *xml2py.py*.

```
tITLE_1=TITLE(TITL=TITLE_TITL(title='Exemple epx2xml'))

from GROUPC1MATERIALS__SOLIDMATERIALS__content import LINEARELASTICITY_LINE_RO_RO
from GROUPC1MATERIALS__SOLIDMATERIALS__content import LINEARELASTICITY_LINE_RO_YOUN
from GROUPC1MATERIALS__SOLIDMATERIALS__content import LINEARELASTICITY_LINE_RO_NU
from GROUPC1MATERIALS__SOLIDMATERIALS__content import LINEARELASTICITY_LINE_RO
from GROUPC1MATERIALS__SOLIDMATERIALS__content import LINEARELASTICITY_LINE
from GROUPC1MATERIALS__SOLIDMATERIALS__content import LINEARELASTICITY

lINEARELASTICITY_1=LINEARELASTICITY(LINE=LINEARELASTICITY_LINE(RO=LINEARELASTICITY_LINE_RO(RO=LINEARELASTI
CITY_LINE_RO_RO(rho=7800),YOUN=LINEARELASTICITY_LINE_RO_YOUN(young=2100000000000.0),NU=LINEARELASTICITY_LIN
E_RO_NU(nu=0.3),VISC=None),LECTURE=['STRUCT']))

from GROUPDLINKS__content import LINKCATEGORY_COUP_COUP
from GROUPDLINKS__content import LINKCATEGORY_COUP
from GROUPDLINKS__content import LINKCATEGORY

lINKCATEGORY_1=LINKCATEGORY(COUP_or_DECO_or_LIAI=LINKCATEGORY_COUP(COUP=LINKCATEGORY_COUP_COUP(SOLV=None))
)

from GROUPDLINKS__content import RELATIONS_RELA_items_nrel_coef_EGAL_items_coef_icomp_nuneu_LECTURE
from GROUPDLINKS__content import RELATIONS_RELA_items_nrel_coef_EGAL_items_coef_icomp_nuneu
from GROUPDLINKS__content import RELATIONS_RELA_items_nrel_coef_EGAL_EGAL
from GROUPDLINKS__content import RELATIONS_RELA_items_nrel_coef_EGAL
from GROUPDLINKS__content import RELATIONS_RELA
from GROUPDLINKS__content import RELATIONS

rELATIONS_1=RELATIONS(RELA=RELATIONS_RELA(items_nrel_coef_EGAL=[RELATIONS_RELA_items_nrel_coef_EGAL(nrel=6
,items_coef_icomp_nuneu=[RELATIONS_RELA_items_nrel_coef_EGAL_items_coef_icomp_nuneu(coef=1,icomp=3,nuneu_o
r_LECTURE=RELATIONS_RELA_items_nrel_coef_EGAL_items_coef_icomp_nuneu_LECTURE(LECTURE=['GRN1'],SHIF=None)),
RELATIONS_RELA_items_nrel_coef_EGAL_items_coef_icomp_nuneu(coef=-
2,icomp=3,nuneu_or_LECTURE=RELATIONS_RELA_items_nrel_coef_EGAL_items_coef_icomp_nuneu_LECTURE(LECTURE=['GR
N2'],SHIF=None))],EGAL=RELATIONS_RELA_items_nrel_coef_EGAL_EGAL(dof_1=0,dof_2=0,dof_3=1,dof_4=0,dof_5=0,do
f_6=0,dof_7=0,LECTURE=['GRN3']))]))
```

Figure 3-5 : Exemple de conversion d'un fichier au format XML en script python

Le script python ainsi obtenu peut être exécuté dans l'environnement du module EPXDATA dans SALOME pour construire l'arborescence des commandes contenues dans le fichier XML.

3.3.2 Gestion des erreurs de transcription

Dans les classes XDATA générées à partir des pages du manuel EPX, la notion d'argument obligatoire pour un mot-clé donné est prise en compte. Cela représente un outil de vérification de la conformité des données fournies par l'utilisateur dans le fichier EPX par rapport au manuel, qui vient en complément de l'interprétation effectuée lors de la conversion des données au format XML.

Il est donc possible, en cas d'argument manquant pour une commande donnée, que le script python obtenu en sortie de *xml2py.py* produise une erreur lors de son exécution dans l'environnement du module EPXDATA. Comme au § 3.2.2, il est choisi de ne pas terminer l'exécution du script *xml2py.py* en cas d'erreur, pour permettre de reconstruire dans l'IHM l'intégralité des données valides. Les erreurs sont signalées sur la sortie standard pour permettre à l'utilisateur de corriger éventuellement les données.

3.3.3 Gestion de configuration

La gestion de configuration du script *xml2py.py* est intégrée à celle du module EPXDATA.

3.4 Outils auxiliaires

3.4.1 Import EPX *dans le module EPXDATA*

A partir des outils précédents, une commande *Import EPX* est ajoutée au menu du module EPXDATA. Elle réalise les actions suivantes, à partir de la seule donnée du fichier de données au format EPX (cf. boîte de dialogue sur la Figure 3-6) :

1. interprétation du fichier EPX et génération du fichier XML via *epx2xml* (qui doit alors être accessible dans le PATH),
2. conversion du fichier XML en script python via *xml2py.py*,
3. exécution du script python obtenu pour créer les instances des classes XDATA avec les paramètres fournis par l'utilisateur.

Lors de la dernière étape, une barre de progression est affichée (cf. Figure 3-7).

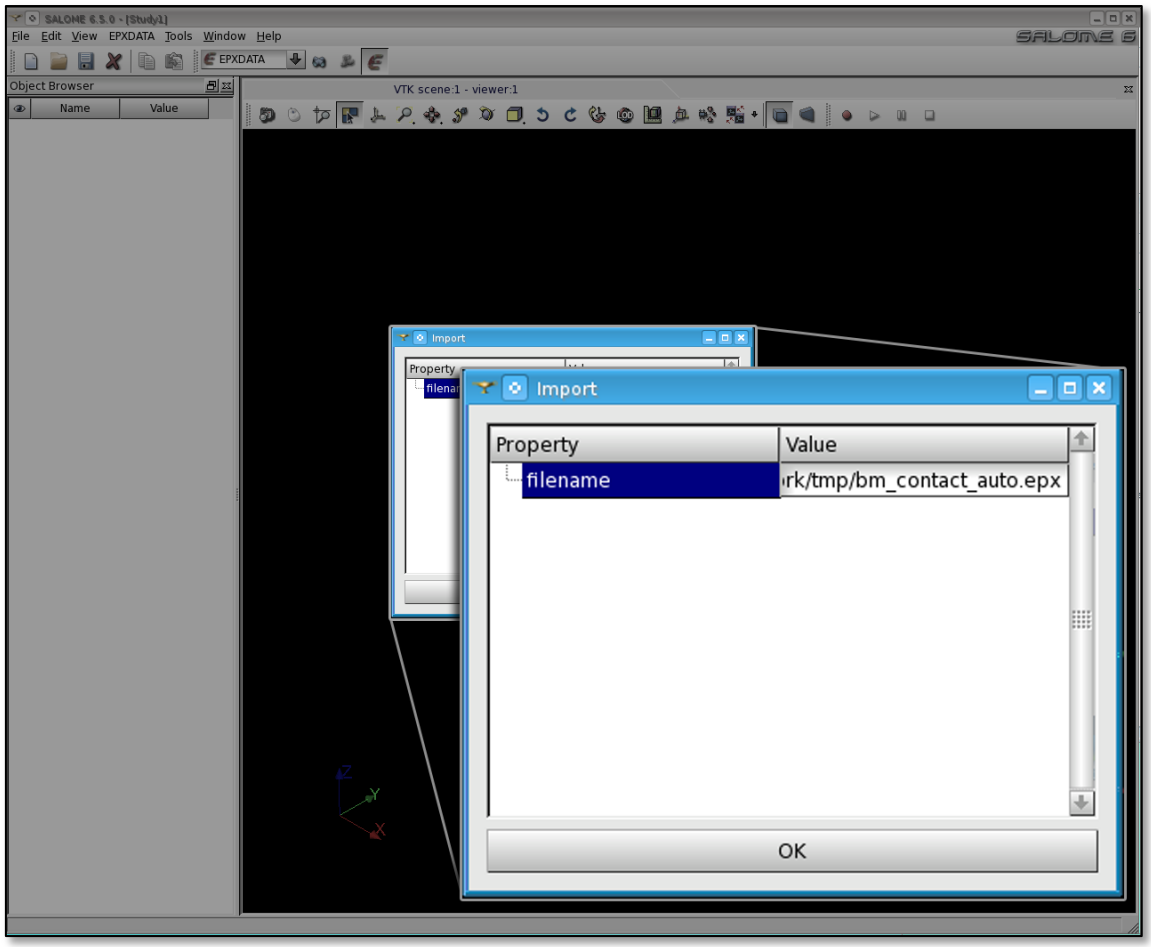


Figure 3-6 : Boîte de dialogue pour l'import d'un fichier EPX

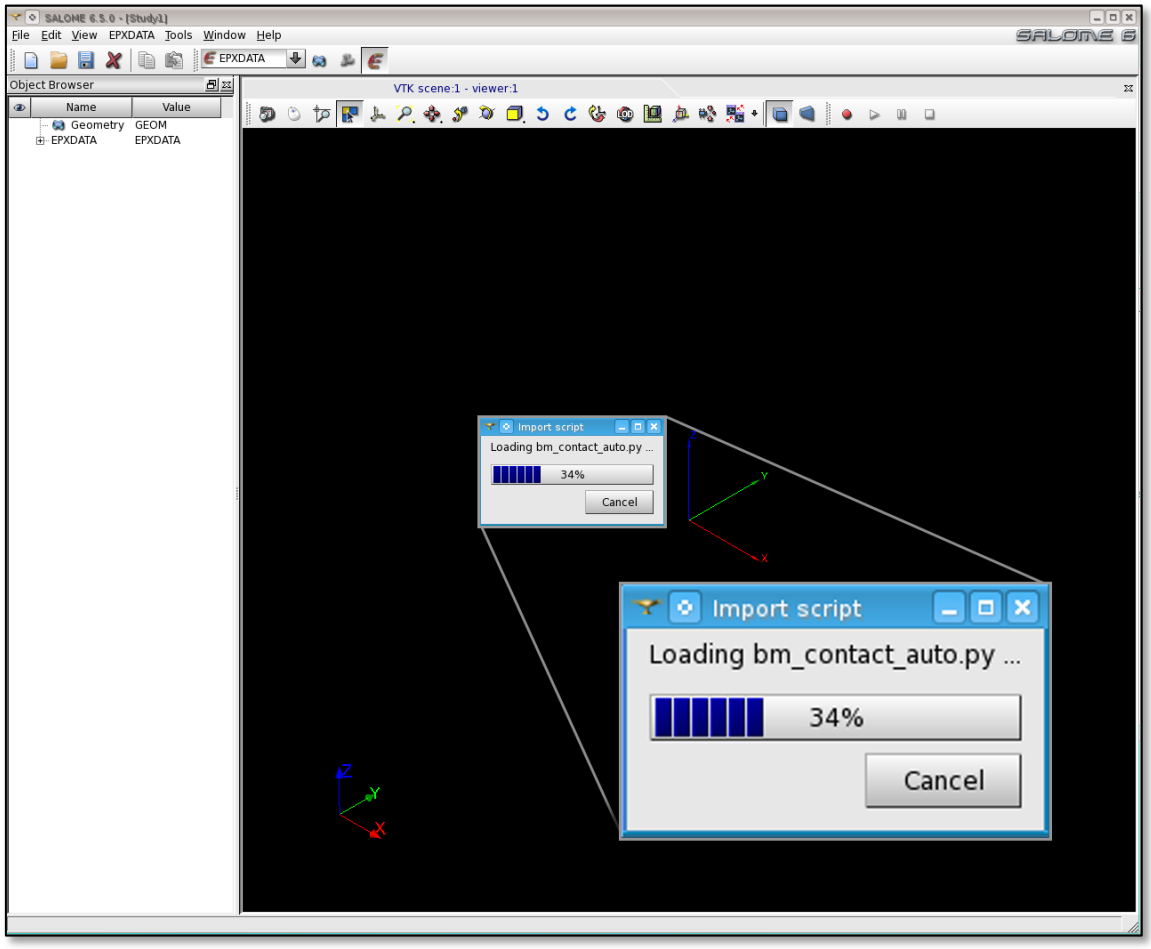


Figure 3-7 : Barre de progression lors de la création des instances des classes XDATA

3.4.2 Reconstruction des liens entre les données EPXDATA et le maillage

Dans la majorité des cas, un jeu de données EPX contient une commande de lecture d'un fichier de maillage (cf. Figure 3-1, commande `MEDL`). Si ce fichier est dans un format lisible par le module SMESH de SALOME, il est judicieux de le charger dans l'interface pour permettre à l'utilisateur d'intervenir simultanément sur les données du maillage et sur les données EPX.

De plus, au moment d'affecter des données à des objets du maillage dans le module EPXDATA, un lien existe entre les modules EPXDATA et SMESH, permettant de sélectionner interactivement les objets dans le maillage, au lieu de renseigner leur nom comme des chaînes de caractères. Ce lien permet réciproquement de visualiser dans SMESH le support de données affectées dans le module EPXDATA sans recourir au nom de l'objet dans l'arborescence des groupes du maillage.

Lors de l'import d'un fichier EPX dans le module EPXDATA, on procède donc aux deux actions suivantes :

1. recherche dans le fichier EPX d'une référence à un fichier de maillage et import du fichier correspondant dans le module SMESH (si le module est disponible et s'il accepte le format du fichier de maillage naturellement),
2. parcours des instances de classe XDATA créées et remplacement des noms d'objets du maillage par des instances de groupes (de nœuds ou d'éléments) de même nom s'ils existent dans le maillage.

Remarque – Le processus de reconstruction suppose que les noms de groupes sont uniques dans le maillage, ce qui n'est pas obligatoire pour des maillages au format MED, où un groupe de nœuds et un groupe d'éléments

peuvent avoir le même nom. Dans ce cas, la première instance de groupe avec le nom recherché rencontrée sera utilisée, ce qui n'assure pas que la correspondance entre le nom et l'objet du maillage soit correcte.

La Figure 3-8 présente le résultat à l'écran de l'import du fichier EPX présenté sur la Figure 3-1 sans reconstruction du lien avec le maillage. Au contraire, la Figure 3-9 présente le résultat avec la reconstruction, l'affichage dans la fenêtre 3D étant obtenu à partir de l'objet signalé en rouge dans l'*Object Browser* (à savoir le support auquel est affecté l'élément fini nommé Q4GS, auparavant simplement identifié par la chaîne de caractère `S1`).

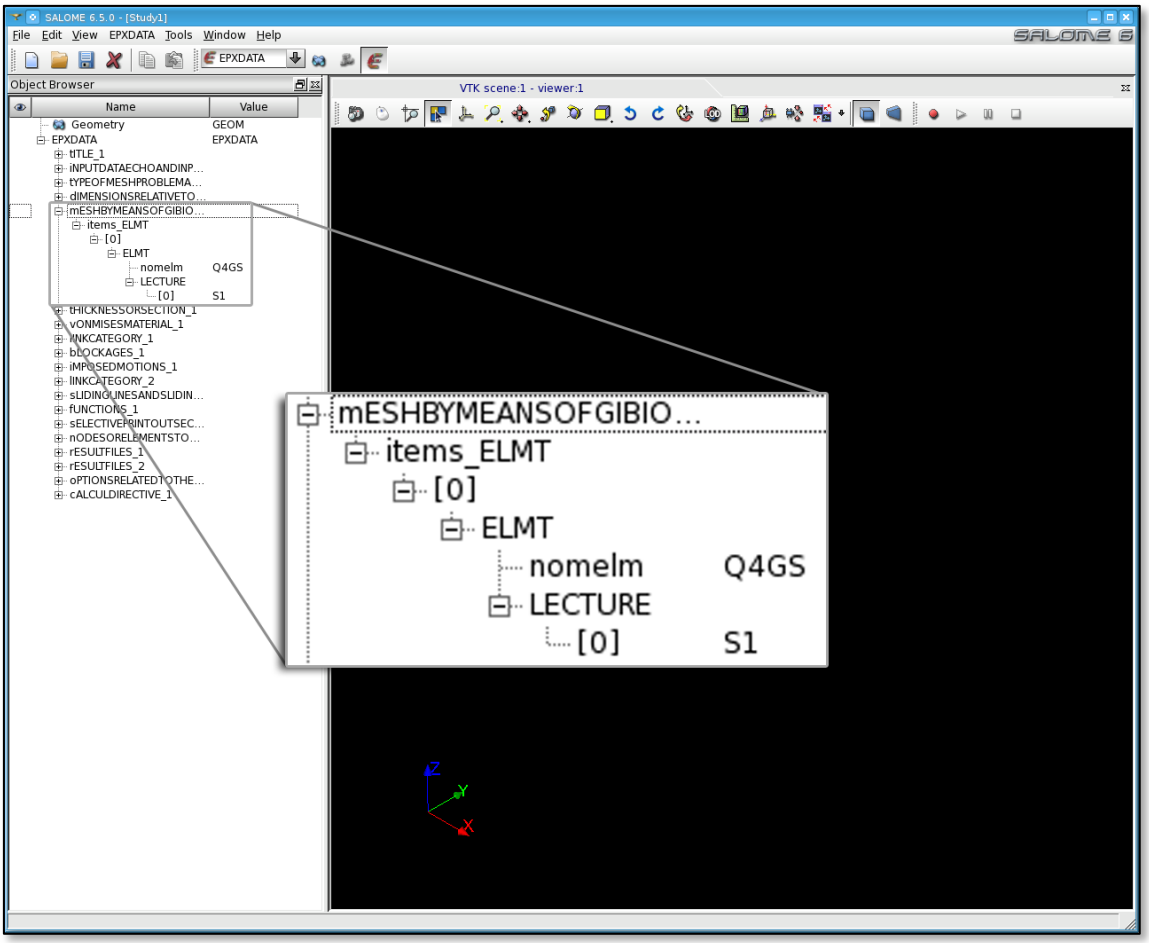


Figure 3-8 : Import d'un fichier EPX sans reconstruction du lien avec le maillage

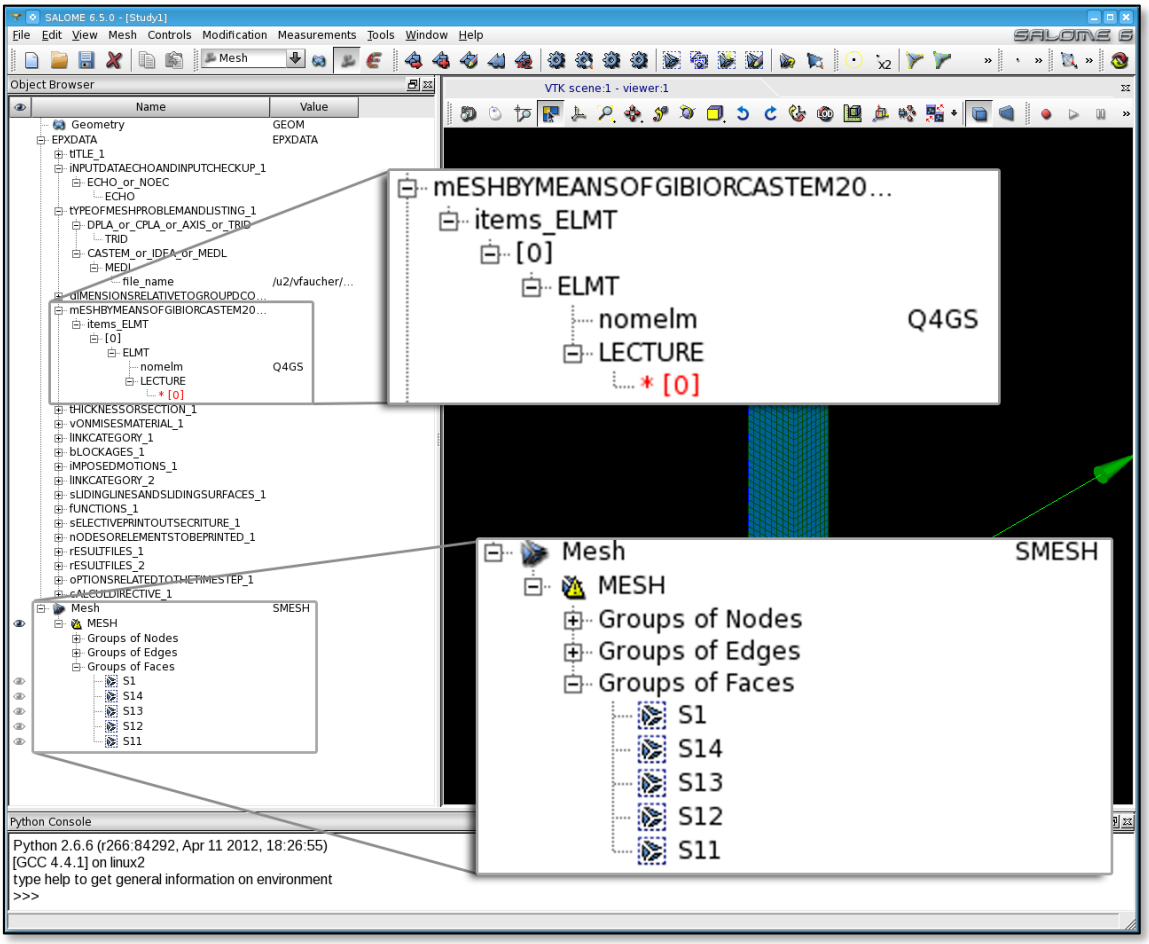


Figure 3-9 : Import d'un fichier EPX avec reconstruction du lien avec le maillage

La reconstruction du lien avec le maillage permet de disposer après import d'un environnement d'étude très proche de celui obtenu si on crée le jeu de données intégralement dans SALOME.

3.4.3 *Script python* py2epx.py

Pour permettre de tester automatiquement, sans recours à l'interface graphique de la plateforme SALOME, le résultat de l'interprétation d'un fichier EPX *fic.epx* et de la transcription du fichier XML intermédiaire *fic.xml* en script python *fic.py*, un second script python *py2epx.py* est proposé.

L'utilisation est la suivante :

```
python py2epx.py fic.py dir_export fic_export.epx
```

Ce script réalise l'export dans le répertoire *dir_export* du fichier *fic_export.epx* au format EPX correspondant au contenu du fichier *fic.py* comme si ce dernier avait été exécuté dans l'environnement du module EPXDATA, et suivi d'un export à l'aide de la commande *Export EPX* [4].

La Figure 3-10 donne le résultat de l'exécution du script *py2epx.py* à partir du script python de la Figure 3-5.

```
$ TITLE
Exemple epx2xml



MATE
$ LINEAR ELASTICITY
LINE RO 7800 YOUN 2.100000E+11 NU 3.000000E-01 LECT 'STRUCT' TERM
```

```
LINK
$ LINK CATEGORY
COUP


$ RELATIONS
RELA 1 6 2
1 3 LECT 'GRN1' TERM
-2 3 LECT 'GRN2' TERM
EGAL 3 LECT 'GRN3' TERM



FIN
```

Figure 3-10 : Exemple de résultat de l'exécution du script *py2epx.py*

Le script *py2epx.py* est intégré au module EPXDATA, ce qui assure sa gestion de configuration.

3.4.4 Script shell epx2epx.sh

Le dernier élément pour la vérification automatique du fonctionnement conjoint de tous les outils ci-dessus est fourni sous la forme d'un script shell *epx2epx.sh*, réalisant, à partir de l'entrée d'un fichier *fic.epx* au format EPX les actions :

```
epx2xml fic.epx fic.xml
python xml2py.py fic.xml
python py2epx.py fic.py . fic_chk.epx
```

Le script *epx2epx.sh* vérifie l'installation correcte de tous les outils nécessaires à son exécution et produit, en plus du fichier de vérification *fic_chk.epx* au format EPX, à comparer au fichier d'entrée *fic.epx*, un fichier d'information *fic.log* contenant les impressions sur la sortie standard produites par *epx2xml*, *xml2py.py* et *py2epx.py*.

Le script *epx2epx.sh* est intégré à l'ensemble des procédures d'utilisation du code EPX au laboratoire DEN/DANS/DM2S/SEMT/DYN, pour assurer son suivi. Une exécution systématique de ce script est envisagée dans le cadre de la modification de la procédure d'évolution des pages de manuel via l'Atelier Logiciel EPX (cf. § 2.2).

4 Actions d'amélioration/correction du module EPXDATA

Le présent paragraphe s'appuie sur les retours des utilisateurs après le pré-déploiement du module, figurant dans [1]. Les deux paragraphes suivants détaillent précisément la réponse apportée à deux réserves majeures formulées sur l'ergonomie du module. Le troisième est dédié à un parcours systématique des retours et des actions associées.

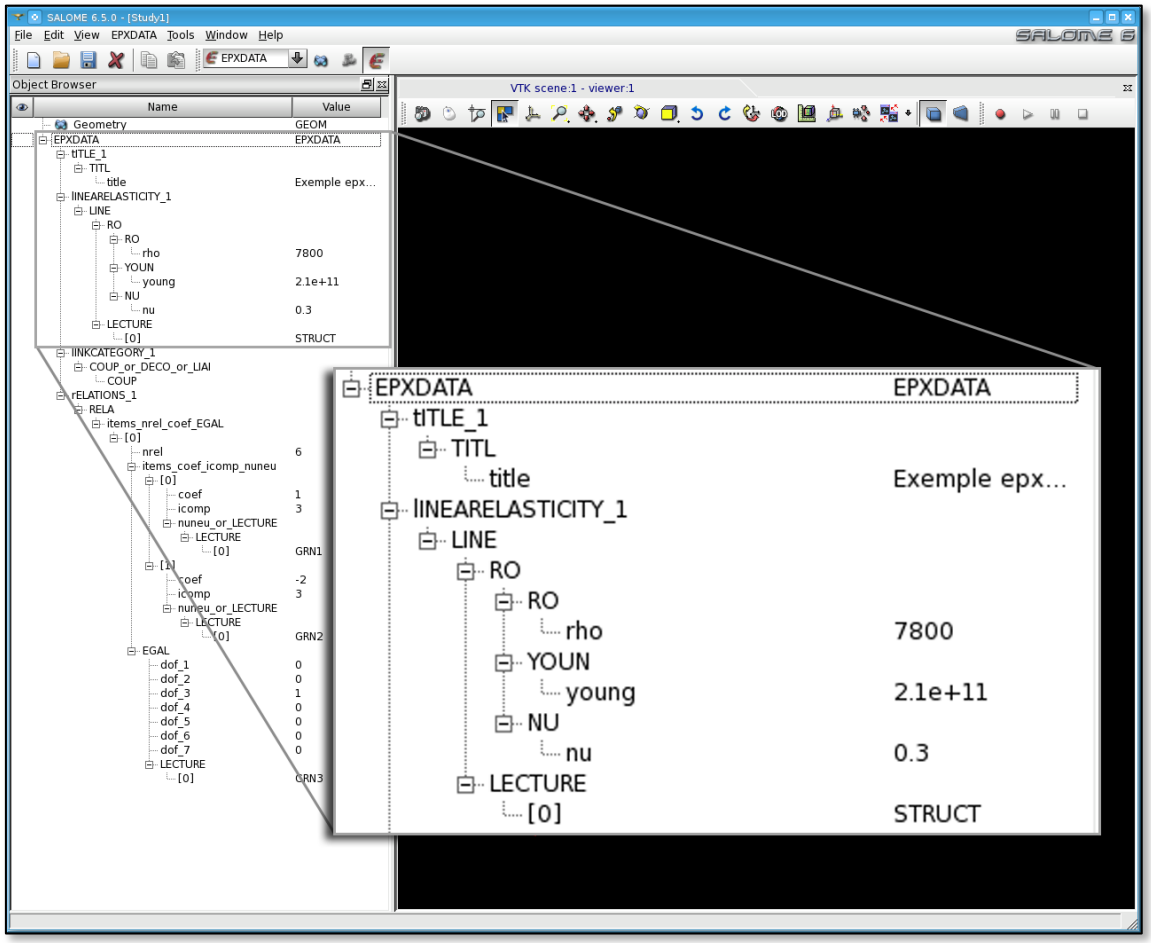
4.1 Choix entre l'*Object Browser* de SALOME ou le *Data Explorer* d'XDATA pour représenter les données dans le module EPXDATA

4.1.1 Motivation et description

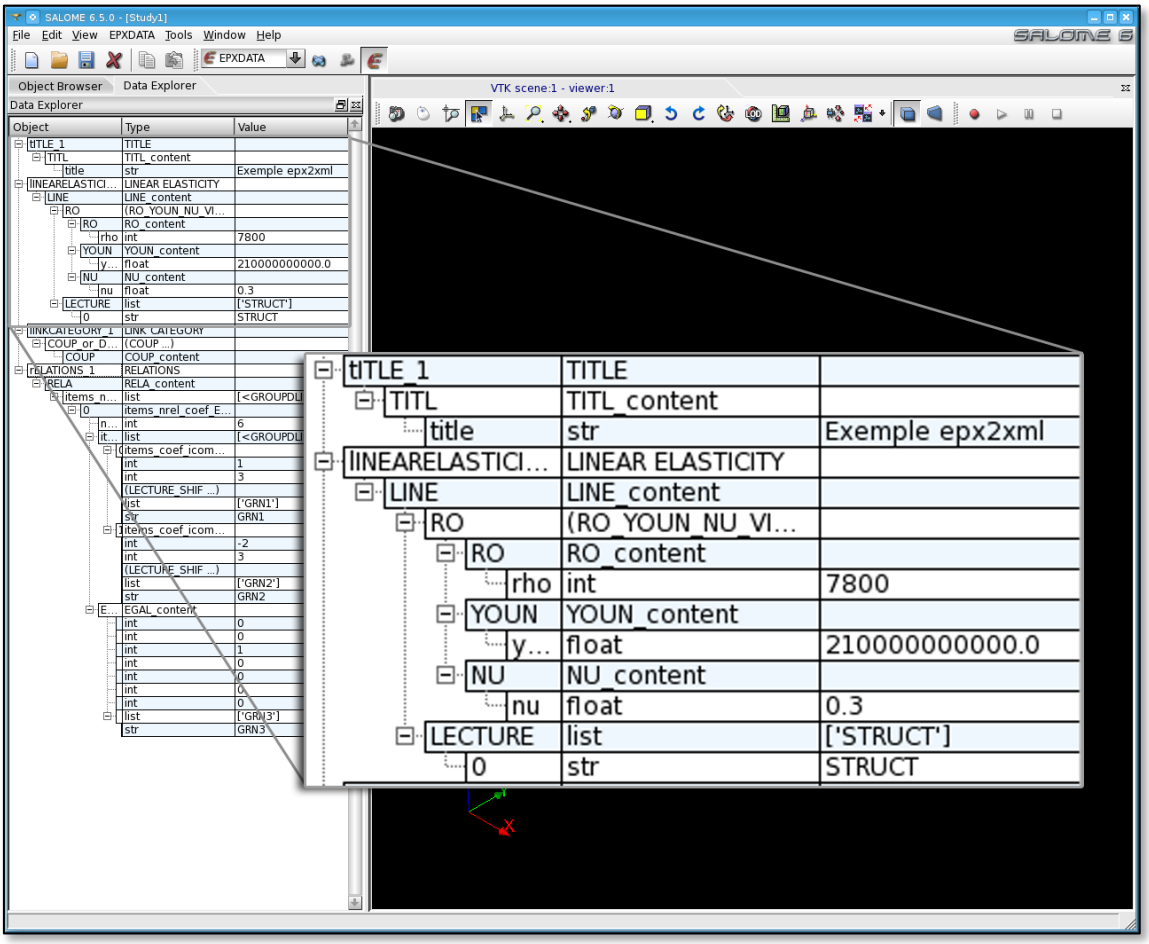
Cette action répond à une limitation pénalisante pour les utilisateurs du module EPXDATA, à savoir l'impossibilité de réaliser une tâche de type *copier/coller* dans l'*Object Browser* de SALOME. Pour certains jeux de données EPX contenant une répétition de commandes très voisines, comme notamment la définition de réseaux de tuyauteries (voir [1] pour un exemple significatif), il en résulte une lourdeur importante de la mise en données, obligeant l'utilisateur à répéter manuellement les actions.

En attendant une intégration éventuelle de ces fonctionnalités dans l'*Object Browser* de SALOME (cf. [1]), un contournement est proposé. Il repose sur les fonctionnalités propres à XDATA, utilisables notamment via l'outil *xdatagui* [3], qui propose un *frontend* avec une représentation spécifique des données nommée *Data Explorer*, gérant le *copier-coller.*

Lors de l'utilisation du module EPXDATA dans l'environnement SALOME, il est prévu par XDATA de pouvoir remplacer l'*Object Browser* par le *Data Explorer*. La Figure 4-1 montre les deux modes d'affichage des données pour le cas simple de la Figure 3-4.



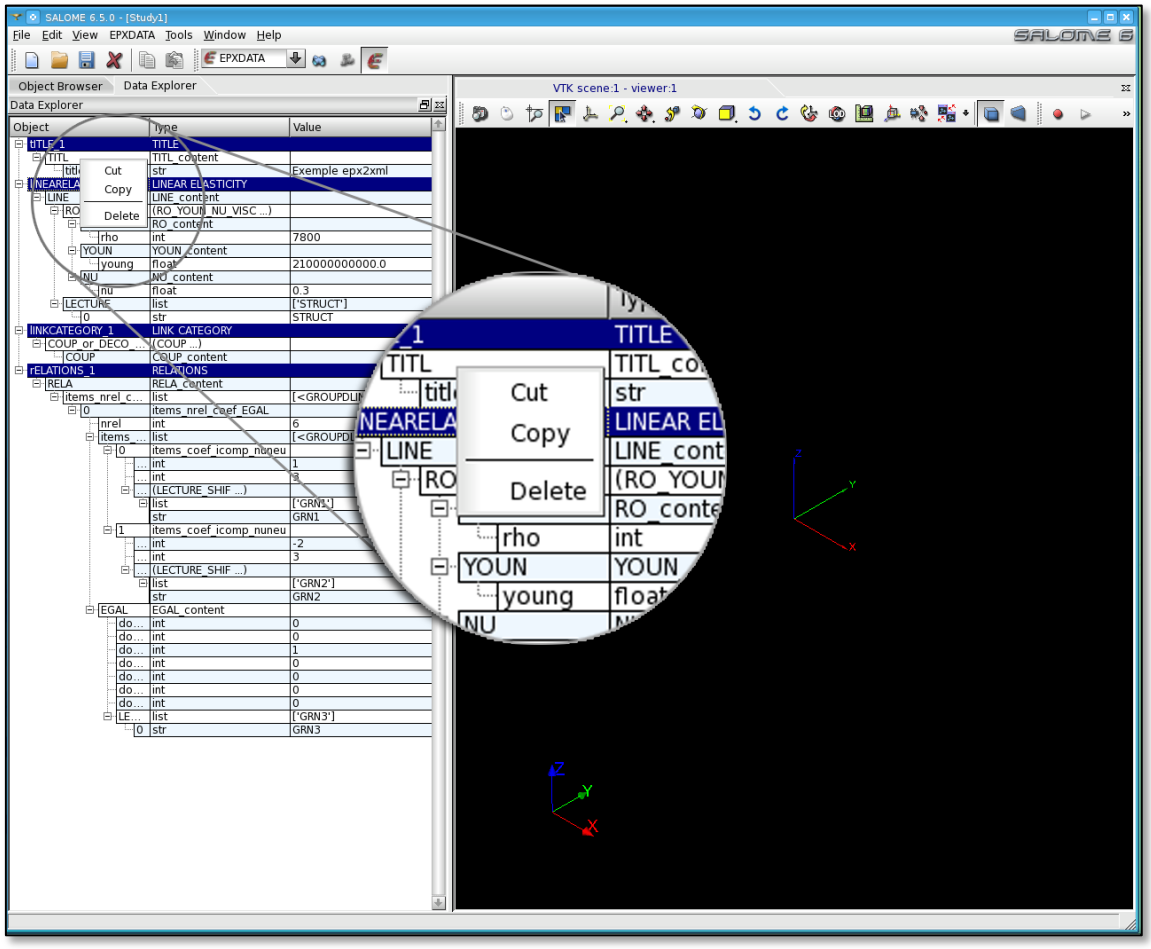
(a) Représentation des données dans l'*Object Browser*



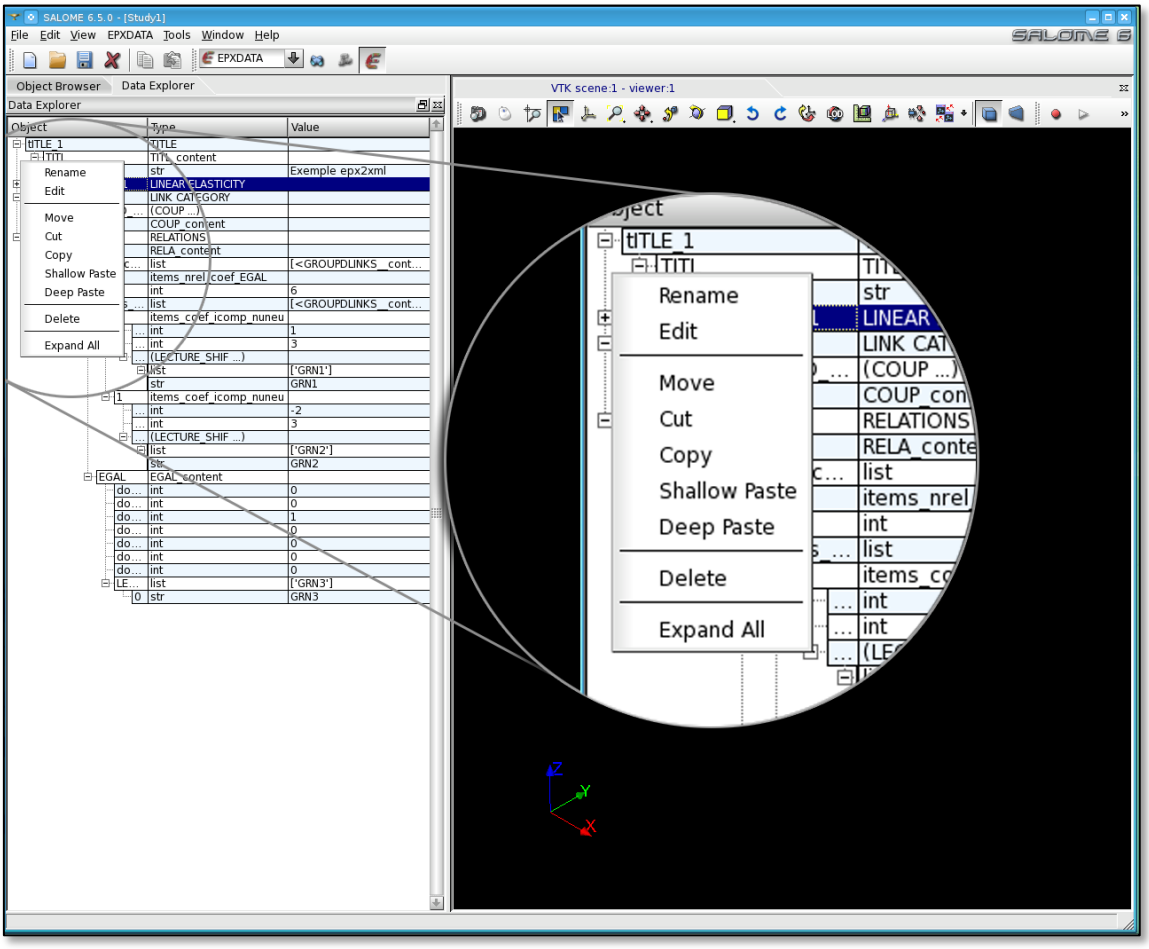
(b) Représentation des données dans le *Data Explorer*

Figure 4-1 : *Object Browser* et *Data Explorer* pour la représentation des données dans le module EPXDATA

La Figure 4-2 présente à titre d'exemple la duplication de la commande de définition et d'affectation du matériau élastique linéaire nommé *lINEARELASTICITY_1* dans le *Data Explorer.*

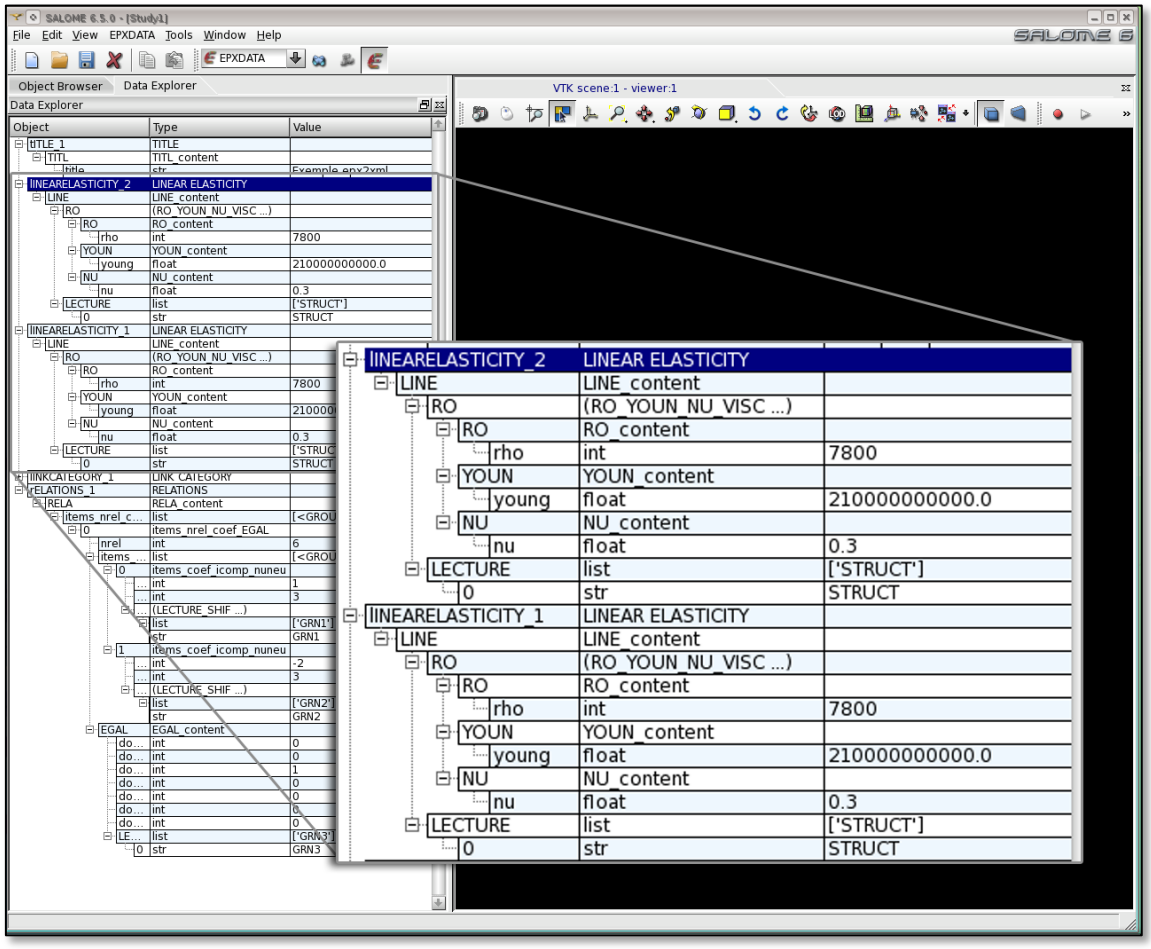


(a) *Copier* de la commande *lINEARELASTICITY_1*



(b) *Coller* de la commande copiée

*Le choix est proposé entre une copie avec duplication (*Deep Paste*) ou sans duplication (*Shallow Paste*) – Dans le cas des données pour EPX, où la copie sert le plus souvent à modifier certains paramètres numériques ou l'affectation de propriétés à des objets du maillage, la copie avec duplication est la plus courante.*



(c) Résultat du *copier/coller* dans le *Data Explorer* (*Deep Paste*)

Figure 4-2 : *Copier/Coller* dans le module EPXDATA en utilisant le *Data Explorer*

4.1.2 Limitations et stratégie pour l'utilisateur

L'utilisation du *Data Explorer* présente une limitation principale pour le module EPXDATA : ce mode de représentation des données ne propose pas le lien avec le module de maillage SMESH qu'apporte l'*Object Browser* (cf. § 3.4.2). Le remplacement des chaînes de caractères pour les noms d'objets du maillage dans les données EPX par des instances de groupes visualisables dans SMESH (cf. § 3.4.2) est alors inopérant, ce qui prive l'utilisateur d'une fonctionnalité importante.

Par défaut, le choix d'un mode de représentation pour les données dans un module reposant sur XDATA est figé au moment de la conception du module. Dans le cas présent, devant l'absence (éventuellement temporaire) de mode présentant l'ensemble des fonctionnalités souhaitées, la possibilité est offerte à l'utilisateur de choisir entre les deux modes à chaque lancement de SALOME, via le positionnement de la variable d'environnement *EPX_SALOME_MODE*.

La valeur *light* pour cette variable active l'utilisation du Data Explorer, alors que toute autre valeur active l'utilisation de l'Object Browser. L'utilisateur souhaitant bénéficier alternativement des fonctionnalités propres à chaque mode de représentation doit ainsi effectuer une sauvegarde de l'étude courante (via un export au format EPX) et un redémarrage de SALOME pour passer de l'un à l'autre.

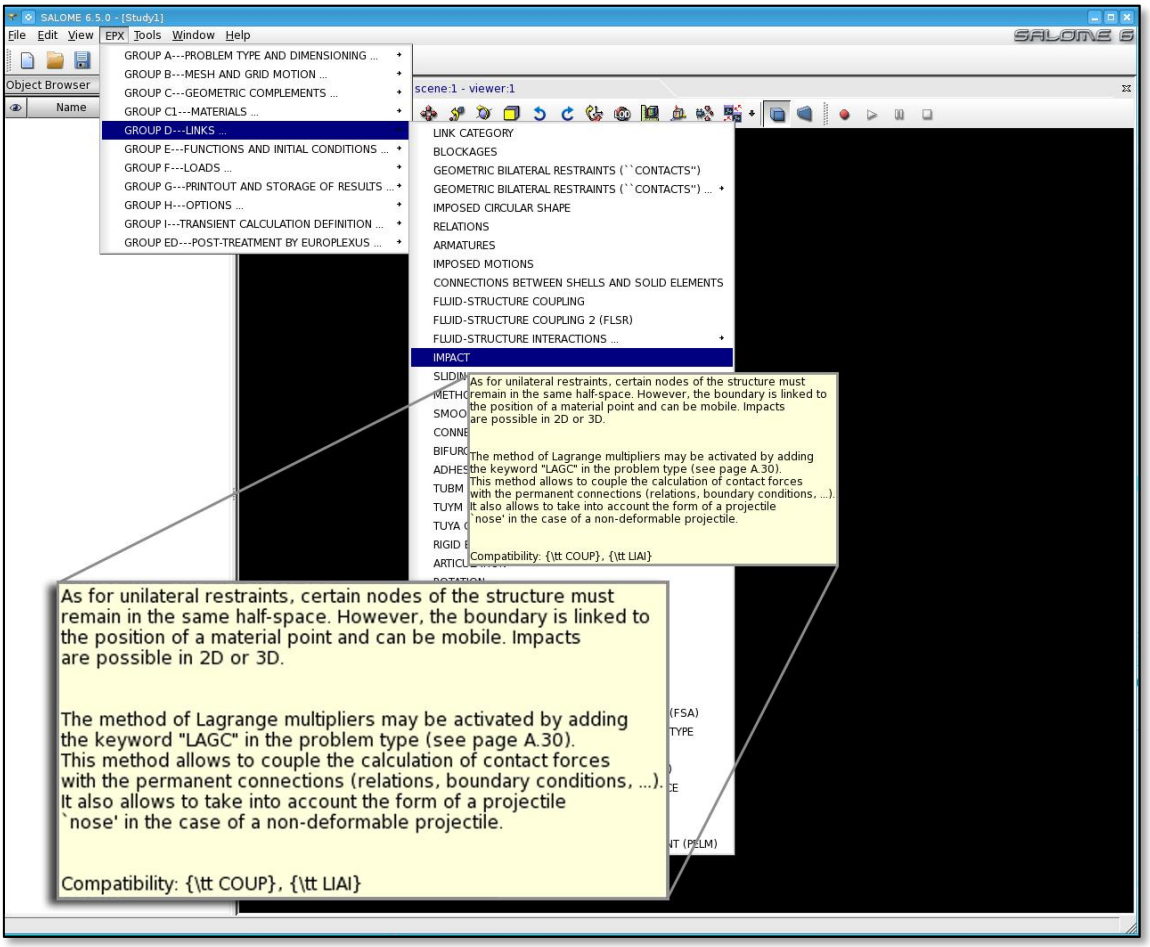
4.2 Génération de bulles d'aide lisibles pour les commandes EPX

Dans l'environnement SALOME, lors du survol d'un item du menu EPXDATA, une bulle d'aide (*tooltip*) est affichée temporairement pour donner à l'utilisateur des informations sur l'item concerné. Le contenu de cette bulle

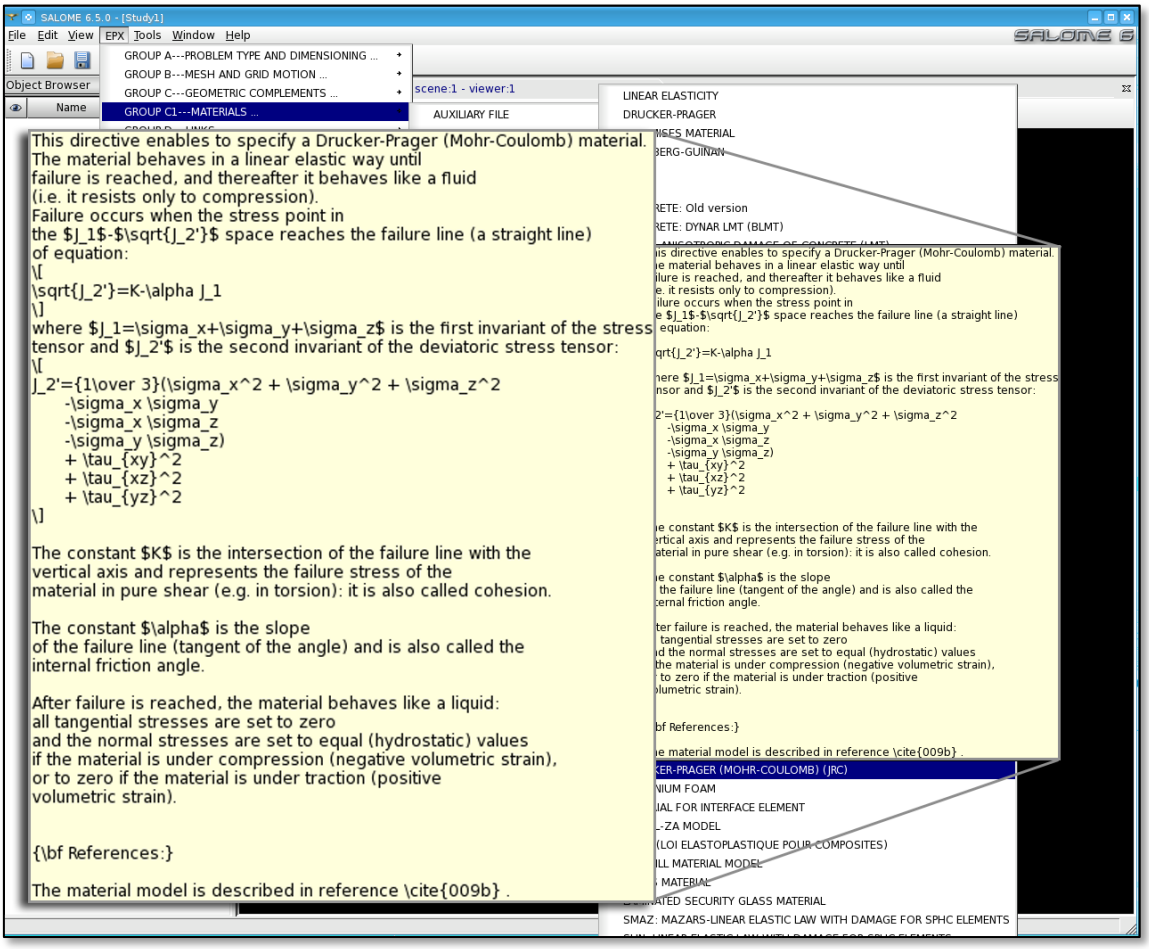
est extrait automatiquement de la section *Object* de la page du manuel EPX correspondant à l'item, toujours pour préserver l'adéquation permanente entre la mise en données via le module EPXDATA et le contenu du manuel.

La réserve formulée dans [1] pour ce fonctionnement tient à la nature du texte figurant dans la section *Object* citée ci-dessus : il s'agit de code source LaTeX, destiné à être interprété avant d'être consulté par un utilisateur. Cette interprétation demandant des développements spécifiques, le texte brut était passé dans les bulles d'aide dans les versions antérieures du module EPXDATA.

La Figure 4-3 présente deux situations, l'une acceptable où la lecture du code source LaTeX ne diffère que peu de celle du texte final, l'autre inacceptable où le code source affiché à l'écran contient un grand nombre d'éléments de syntaxe LaTeX spécifiques.



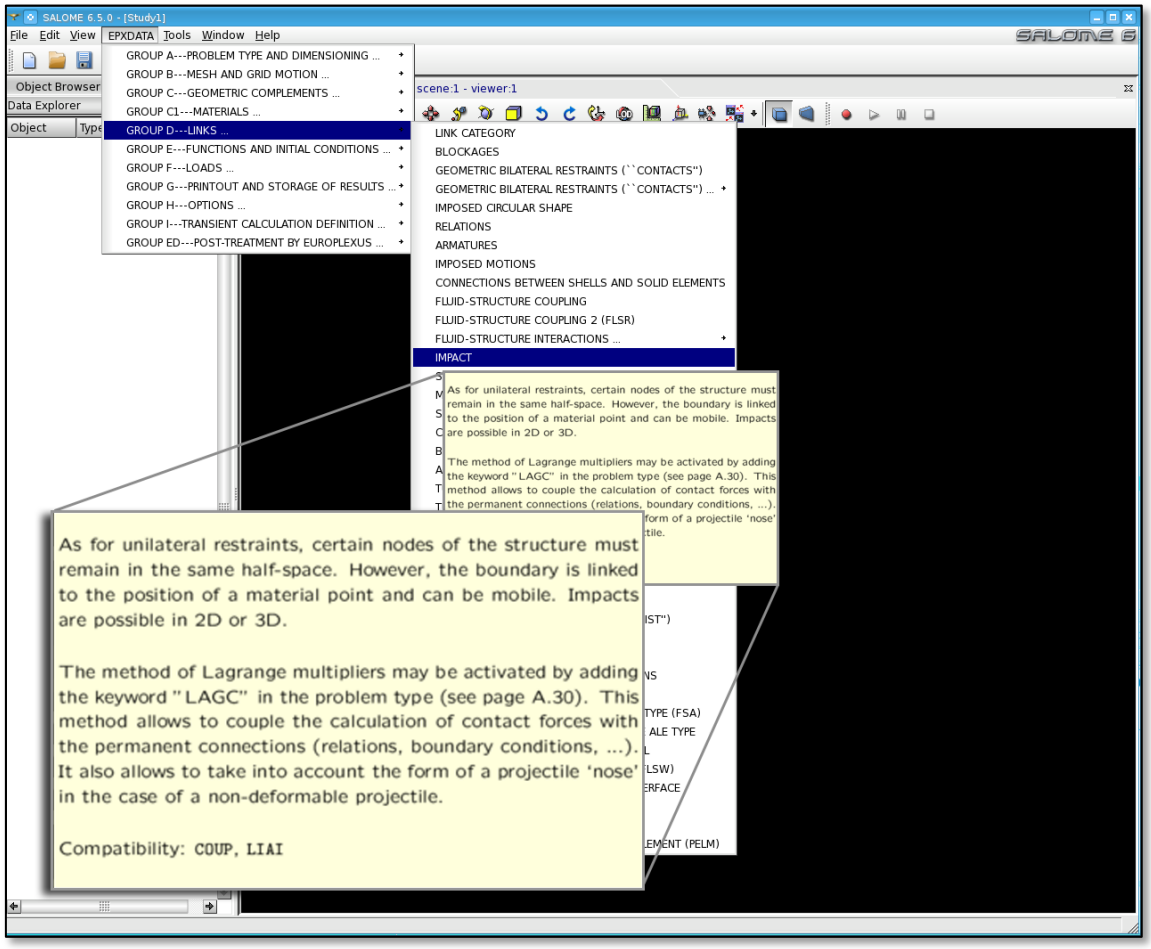
(a) Code source LaTeX lisible dans la bulle d'aide



(b) Code source LaTeX avec un grand nombre d'éléments de syntaxe spécifiques

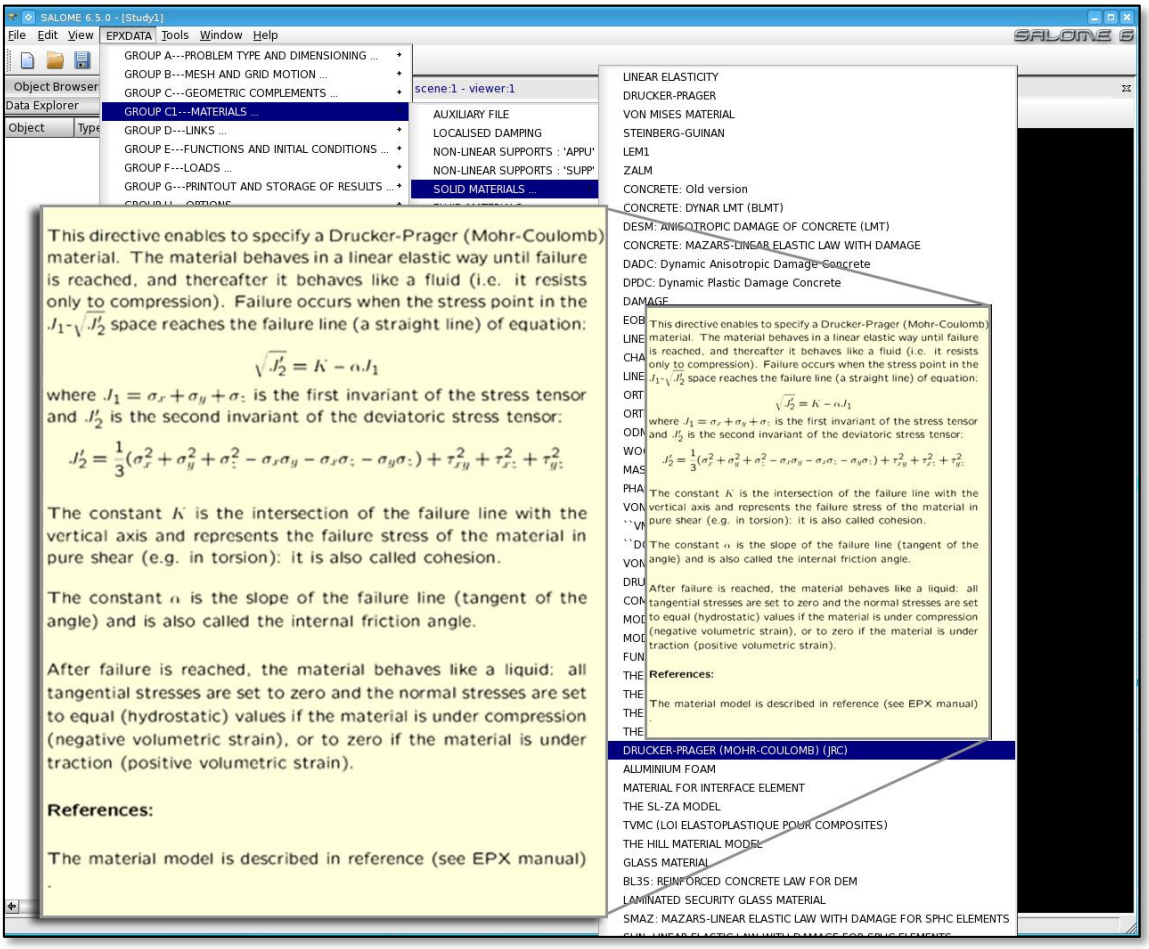
Figure 4-3 : Exemples de bulles d'aide contenant du code source LaTeX brut

Pour pallier le problème lié à la syntaxe LaTeX, on intègre à l'interprétation des pages de manuel EPX la compilation via LaTeX du texte de la section *Object*, avec une sortie sous la forme d'une image remplaçant le texte brut dans la bulle d'aide. La Figure 4-4 présente le résultat pour les deux cas précédents.



(a) Image remplaçant du texte avec peu de syntaxe LaTeX

On vérifie en comparant avec la Figure 4-3(a) que la compilation via LaTeX n'améliore que peu la lisibilité de cette bulle d'aide



(b) Image remplaçant du texte avec de nombreux éléments de syntaxe LaTeX

Au contraire du cas précédent, la bulle d'aide devient beaucoup plus lisible grâce à la compilation du code source LaTeX, en particulier pour les formules mathématiques

Figure 4-4 : Bulles d'aide après compilation du code source LaTeX de la section *Object* des pages de manuel EPX

4.3 Suivi des actions correctives

On reprend la classification des remarques formulées par les utilisateurs dans [1]. Elles sont classées selon 4 catégories, avec une mesure de gravité de 1 (plus grave) à 3 (moins grave) en fonction de la gêne dans l'utilisation du module qu'elles décrivent. Les catégories sont les suivantes (la catégorie nommée *Outil* dans [1] est renommée à présent *EPXDATA*, pour prendre en compte le nom définitif du module) :

1. remarques relatives au module EPXDATA proprement dit (ergonomie, dysfonctionnement) (nom : EPXDATA),
2. remarques relatives à SALOME et à son environnement (nom : Salome),
3. remarques relatives à XDATA (nom : XDATA)
4. remarques relatives à la documentation d'EPX, dont une écriture modifiée, sans changer le sens des commandes pour le programme, permet, soit de simplifier la mise en données, soit de la corriger si le fichier généré à partir de la version actuelle de la documentation est incorrect (cf. exemples ci-dessous) (nom : Doc_EPX).

Une 5^{e} catégorie est considérée dans [1], regroupant des remarques relatives à EPX, lorsque le code est mis en œuvre à partir d'un jeu de données généré avec l'outil proposé (nom : EPX). Après examen, cette catégorie était vide et elle n'est donc pas prise en compte dans le présent document.

Le détail des commentaires associés à chaque retour des utilisateurs est à lire dans [1]. On ajoute à chaque remarque un état des lieux de sa prise en compte dans la version actuelle du module EPXDATA. Cet état des lieux est identifié avec une couleur verte si une correction, au moins partielle, a été apportée en réponse au problème soulevé. La gravité de certains dysfonctionnements a été réévaluée, à la hausse uniquement. Le cas échéant, cette réévaluation est signalée dans l'état des lieux proposé.

4.3.1 Remarques relatives au fonctionnement du module EPXDATA

Intitulé	**Description**	**Gravité**
EPXDATA-1	Lors de la destruction d'un objet dans l'*Object Browser*, il n'est pas totalement supprimé. Lors de la sortie du fichier de commande EPX correspondant, l'élément supprimé figure dans le jeu de données.	**1**
Etat	Correction effectuée	

Intitulé	**Description**	**Gravité**
EPXDATA-3	Lors de la saisie du nom du fichier maillage, le cadre de saisie est trop petit, en comparaison à la longueur du texte (on doit entrer le chemin complet du fichier maillage).	**2**
Etat	Non traité	

Intitulé	**Description**	**Gravité**
EPXDATA-4	Problème pour définir manuellement des groupes pour la sortie PVTK dans la directive ECRITURE.	**2**
Etat	Correction effectuée	

Intitulé	**Description**	**Gravité**
EPXDATA-5	Problème pour définir les fichiers de sortie dans la directive ECRITURE (fichier PVTK ou ALIT).	**2**
Etat	Correction effectuée	

Intitulé	**Description**	**Gravité**
EPXDATA-6	Le nom des classes apparaissant dans l'Object Browser est peu lisible.	**2**
Etat	Amélioration du nom des instances des classes XDATA	

Intitulé	**Description**	**Gravité**
EPXDATA-7	Dans les menus déroulant permettant le choix de blocs de syntaxe associés à certains mots-clés, pour suivre la structure hiérarchique complexe de la documentation EPX, les différents blocs possibles sont affichées entre parenthèses et accompagnés de points de suspension. Ce n'est pas conforme à la description du rapport [4].	**2**
Etat	Amélioration difficile à spécifier - voir commentaire associé à cette remarque dans [1]	

Intitulé	Description	Gravité
EPXDATA-10	On regroupe dans ce retour toutes les remarques concernant la lisibilité des menus, des boîtes de dialogue et des entrées dans l'Object Browser. 1. L'arbre dans l'Object Browser est parfois peu lisible, en particulier en raison du choix de la nomenclature des blocs de syntaxe hiérarchiques, entraînant éventuellement une répétition de mots-clés. 2. Le menu EPX est très fourni, en raison du grand nombre de directives d'EPX, et sa manipulation est parfois difficile : répétition d'actions, navigation difficile avec superposition de sous-menus…	**2**
Etat	Amélioration du nom des instances des classes XDATA	

Intitulé	Description	Gravité
EPXDATA-11	La mise en données des blocages est trop compliquée.	**2**
Etat	Amélioration effectuée : mettre la valeur 0 par défaut pour LECDDL	

Intitulé	Description	Gravité
EPXDATA-15	Le *tooltip* d'aide s'affiche automatiquement en cas de survol prolongé d'un item du menu EPX et peut alors cacher le contenu du reste du menu, rendant difficile la navigation.	**2**
Etat	Voir § 4.2 pour l'amélioration des bulles d'aide (*tooltips*). Le masquage du contenu du menu est difficile à empêcher.	

Intitulé	Description	Gravité
EPXDATA-18	La mise en données étant relativement fastidieuse, il serait souhaitable de disposer d'une sauvegarde automatique pour éviter de tout reprendre en cas de plantage de l'application ou du système.	**2**
Etat	A implémenter. La situation est moins contraignante en disposant d'une relecture fonctionnelle des jeux de données existants, limitant la création fastidieuse de jeux de données *ex nihilo*.	

Intitulé	Description	Gravité
EPXDATA-2	Lors de la création successive de 2 instances de type TYPEOFMESHPROBLEMANDLISTING, elles sont nommées TYPEOFMESHPROBLEMANDLISTING_1 et TYPEOFMESHPROBLEMANDLISTING_2. Après la suppression de la deuxième, une troisième est créée et elle porte à nouveau le nom TYPEOFMESHPROBLEMANDLISTING_1, au lieu de porter le numéro 3.	**3**
Etat	Correction effectuée	

Intitulé	Description	Gravité
EPXDATA-8	Lorsqu'on sélectionne un mot-clé, en particulier lorsqu'il n'a pas d'attribut, on peut s'attendre à ce que la valeur du champ dans la fenêtre correspondante ait le nom de l'objet (par exemple : mot-clé TRID dans le groupe B). Au contraire, la valeur du champ est une chaîne de caractère dont la signification n'est pas forcément évidente pour l'utilisateur.	**3**
Etat	Non traité (voir commentaire dans [1])	

Intitulé	Description	Gravité
EPXDATA-13	La saisie des points pour l'affichage dans les sorties met en jeu un grand nombre de fenêtres successives, en particulier pour rentrer manuellement les numéros des nœuds.	**3**
Etat	Non traité (voir commentaire dans [1])	

4.3.2 Remarques relatives à la plate-forme SALOME

Intitulé	Description	Gravité
Salome-1	Les fonctions *Cut* et *Copy* ne fonctionnent pas dans l'environnement SALOME	**1**
Etat	Voir § 4.1	

Intitulé	Description	Gravité
Salome-2	Dans le cadre de l'interaction entre la mise en données et le maillage, les objets maillages sélectionnés apparaissent dans l'Object Browser sous la forme d'un numéro entre crochets suivi d'une étoile. Il serait préférable d'avoir le nom de l'objet tel qu'il figure dans le maillage.	**2**
Etat	Non traité	

Intitulé	Description	Gravité
Salome-3	L'interaction avec le maillage est fonctionnelle avec un maillage au format MED, mais certaines difficultés sont rencontrées avec un maillage initialement au format CAST3M/SAUV.	**2**
Etat	L'import direct des fichiers au format SAUV (disponible depuis SALOME 6) règle le problème.	

Intitulé	Description	Gravité
Salome-4	Lors de l'utilisation de *Expand all* dans l'*Object Browser* pour un arbre relativement fourni, l'ascenseur reste positionné au niveau du haut de l'arbre. Si l'objectif de l'expansion était de visualiser les dernières entrées de l'arbre, ce qui est une opération classique, il faut alors faire défiler la fenêtre sur une grande hauteur.	**3**
Etat	Non traité	

4.3.3 Remarques relatives au fonctionnement de XDATA

Intitulé	Description	Gravité
XDATA-2	Dans la procédure de saisie d'un objet maillage à partir de l'arbre de SMESH, on peut choisir sans erreur des catégories d'objets (Groups of Nodes par exemple). Le pointeur sur le maillage obtenu est naturellement incorrect.	**2**
Etat	Non traité	

Intitulé	Description	Gravité
XDATA-1	Dans une fenêtre de saisie avec plusieurs lignes, la ligne active s'élargit et recouvre partiellement la ligne qui suit, rendant difficiles la lecture et le passage d'une ligne à l'autre.	**3**
Etat	Non traité	

Intitulé	Description	Gravité
XDATA-3	Le temps de chargement du module est long (environ 25 s sur une station de travail standard).	**3**
Etat	Non traité	

4.3.4 Remarques relatives à la documentation d'EPX

Intitulé	Description	Gravité
Doc_EPX-1	Le complément géométrique de type RACCORD BIFU n'est pas disponible dans l'IHM.	**1**
Etat	Correction effectuée	

Intitulé	Description	Gravité
Doc_EPX-2	Le processus de saisie des paramètres pour certaines commandes ne correspond à la logique de la commande.	**1**
Etat	Correction partiellement effectuée La logique de correction est bien identifiée, mais il reste des pages de manuel à modifier. A réaliser en fonction des retours sur l'utilisation du module.	

Intitulé	Description	Gravité
Doc_EPX-4	Il manque le mot-clé LIST dans la commande PRESCRIBED CONSTANT FLOW RATE	**1**
Etat	Correction effectuée	

Intitulé	Description	Gravité
Doc_EPX-5	L'export de la directive ECRITURE est faux car la fréquence d'écriture est placée avant les grandeurs à sauver.	**1**
Etat	Correction effectuée	

Intitulé	Description	Gravité
Doc_EPX-6	La directive CELDI est incomplète. De même, les matériaux GLRC et BL3S ne sont pas proposés.	**1**
Etat	Correction effectuée pour CELDI et BL3S – Correction complexe de la page de manuel pour GLRC à effectuer.	

Intitulé	Description	Gravité
Doc_EPX-8	On ne peut pas choisir entre TEAU et TH2O pour entrer les données relatives aux tables de l'eau.	**1**
Etat	Correction effectuée	

Intitulé	Description	Gravité
Doc_EPX-9	Le mot-clé LIST est absent dans l'option CONTOUR de la directive GRILLE	**1**
Etat	Correction effectuée	

Intitulé	Description	Gravité
Doc_EPX-7	Proposition : joindre le mot-clé correspondant à un matériau à son titre (avant ou après), pour rendre plus lisible le menu EPX	**2**
Etat	Le mot-clé principal (matériau ou autre) d'une directive EPX est ajouté entre crochet avant son titre pour améliorer la lisibilité.	

Intitulé	Description	Gravité
Doc_EPX-3	Quand on saisit des conditions cinématiques dans le menu LIAISON, l'objet correspondant apparaît dans le fichier de données EPX exporté dans la directive LINK COUP.	**3**
Etat	*Réévaluée en gravité 1* Correction effectuée (unification de la syntaxe des commandes des liaisons dans le manuel et choix explicite d'un formalisme de traitement parmi LIAI, COUP ou DECO)	

4.3.5 Bilan des actions correctives

Le Tableau 4-1 compare numériquement les actions correctives menées aux remarques formulées dans [1].

	Ratio #corrections / #remarques	**Commentaires**
Remarques de gravité 1	**10 / 10** (dont 1 réévaluation depuis le niveau de gravité 3)	2 corrections partielles (**Doc_EPX-1** et **Doc_EPX-6**) et un contournement (**Salome-1**)
Remarques de gravité 2	**8 / 13**	
Remarques de gravité 3	**1 / 6** (1 réévaluation vers le niveau de gravité 1)	Plusieurs remarques (**EPXDATA-8**, **EPXDATA-13**, **Salome-4** notamment) doivent être précisées pour savoir si une action corrective est nécessaire ou non

Tableau 4-1 : Bilan des actions correctives pour le module EPXDATA

Toutes les remarques de gravité 1 ont été traitées, au moins partiellement, ainsi que la majorité des actions de gravité 2. Ce n'est pas le cas pour les remarques de gravité 3, mais celles-ci revêtent un caractère plus subjectif et demandent souvent à être précisées pour définir l'action corrective adéquate.

5 Conclusion

Le présent rapport décrit la stratégie de conception du module EPXDATA destiné à la construction interactive de jeux de données pour le code EPX dans la plateforme SALOME.

Les développements nécessaires à l'interprétation de jeux de données existants dans le module, élément indispensable pour proposer une solution ergonomique et fonctionnelle aux utilisateurs, sont ensuite présentés.

Le rapport est conclu par un suivi des actions de correction et d'amélioration effectuées sur la base du retour des utilisateurs lors du pré-déploiement du module à l'automne 2011. Les remarques les plus gênantes pour l'utilisation ont toutes fait l'objet d'actions spécifiques, en particulier pour permettre aux utilisateurs de disposer du *Copier/Coller* dans le module EPXDATA et pour améliorer la lisibilité des menus, des boîtes de dialogue et des bulles d'aide.

Le module EPXDATA en version 1.0.0 sera fourni aux utilisateurs du programme EPX à compter de janvier 2013, dans le cadre de la distribution (industrielle et académique) du code par le CEA.

Références

[1] Retour sur le pré-déploiement de l'outil de mise en données EPX dans Salome, V. Faucher, Compte-rendu de réunion CEA/DEN/DANS/DM2S/SEMT/DYN, référence DO 12-006, janvier 2012.

[2] Serveur Web de l'Atelier Logiciel EUROPLEXUS, H. Bung, Rapport CEA SEMT/DYN/RT/08-019/A, décembre 2008.

[3] XDATA User's manual, dans l'archive xdata-xxx.tar.gz (xxx = numéro de version) à l'adresse http://files.salome-platform.org/cea/adam/xdata/download

[4] Mise en données pour EUROPLEXUS dans SALOME avec XDATA – Description de l'outil et de son utilisation, V. Faucher, Rapport CEA SEMT/DYN/RT/11-022/A, septembre 2011.

[5] EUROPLEXUS – Plan Qualité Logiciel, V. Faucher, Document CEA SEMT/DIR/PQ/005/A, septembre 2007.

[6] EUROPLEXUS, User's Manual, http://europlexus.jrc.ec.europa.eu/public/manual_html/index.html

Annexe 1 – Rappel de la procédure d'installation du module EPXDATA

Cette annexe reprend brièvement la procédure d'installation du module EPXDATA décrite dans [4].

Les prérequis sont donnés par le Tableau A-1.

Composant requis	Commentaires
Plateforme SALOME	SALOME ou SALOME-MECA
Logiciel XDATA	Version fournie avec le module EPXDATA ou à trouver avec la plateforme SALOME
Documentation EPX	Pages de manuel au format LaTeX (extension .ttx) Nouveautés pour EPXDATA v1.0.0 On peut donner dans le répertoire du manuel EPX (variable **$EPX_TTX_DOC**, voir ci-dessous) : 1. le fichier **fulldoc.txt** (concaténation des pages de manuel produite lors d'une installation à partir des fichiers .ttx) à la place des pages individuelles, 2. les **images au format PNG** correspondant à la compilation du code LaTeX de la section *Object* des pages de manuel, pour éviter de les reconstruire et réduire significativement le temps d'installation du module ; ces images peuvent être récupérées après une installation complète du module.
Source du module EPXDATA	

Tableau A-1 : Prérequis pour l'installation du module EPXDATA

La procédure d'installation est alors la suivante.

1. Construction d'un fichier *env_EPXDATA.sh* définissant toutes les variables d'environnement utiles (cf. [4].

   Exemple de fichier :

```
source /opt/salome_6.5.0/env_products.sh
export XDATA_INSTALL=/data/EPXDATA_work/xdata-2012-09-28_install
export PYTHONPATH=$XDATA_INSTALL/lib/python2.6/site-packages/xdata:$PYTHONPATH
export PATH=$XDATA_INSTALL/bin:$PATH
export EPX_TTX_DOC=/data/EPXDATA_work/manuel_EPX
export EPXDATA_ROOT_DIR=/data/EPXDATA_work/EPX_INSTALL
export SALOME_MODULES=GEOM,SMESH,EPXDATA
```

2. Dans un répertoire **$EPXDATA** (par exemple **EPXDATA**=*/data/EPXDATA_work*), où on place le fichier *env_EPXDATA.sh* ci-dessus) :

   a. décompresser l'archive *EPX_SRC_1.0.0.tar.gz* (crée le répertoire **$EPXDATA**/*EPX_SRC*),
   b. créer le répertoire **$EPXDATA**/*EPX_BUILD*,
   c. dans le répertoire **$EPXDATA**/*EPX_BUILD*, exécuter les commandes suivantes :

```
source ../env_EPXDATA.sh
../EPX_SRC/build_configure
../EPX_SRC/configure -prefix=$EPXDATA/EPX_INSTALL
make install
```

Pour utiliser le module dans SALOME, il suffit de faire un *source* du fichier *env_EPXDATA.sh* avant de démarrer la plateforme.

Annexe 2 – Module EPXDATA et SALOME-MECA

On donne dans cette annexe un bref aperçu de l'utilisation du module EPXDATA avec la plateforme SALOME-MECA, dérivée de la plateforme SALOME générique et intégrant des outils pour le calcul mécanique (http://www.code-aster.org/V2/spip.php?article346), dont *Code_Aster* (http://www.code-aster.org).

A l'initiative d'EDF, SALOME-MECA intègre un module EUROPLEXUS, complémentaire du module EPXDATA. Il s'agit d'un gestionnaire d'études pour le code EPX. Les deux modules sont destinés à être utilisés séquentiellement, les données pour les études gérées par le module EUROPLEXUS étant préparées à l'aide du module EPXDATA. Il n'y aucune redondance dans les actions réalisables avec chacun des deux modules.

Pour l'instant, le lancement d'une étude dans le module EUROPLEXUS à partir d'un jeu de données chargé dans le module EPXDATA utilise comme intermédiaire le fichier ASCII au format produit par l'export dans le module EPXDATA, ce qui requiert l'intervention de l'utilisateur. Il est envisagé d'automatiser cette étape.

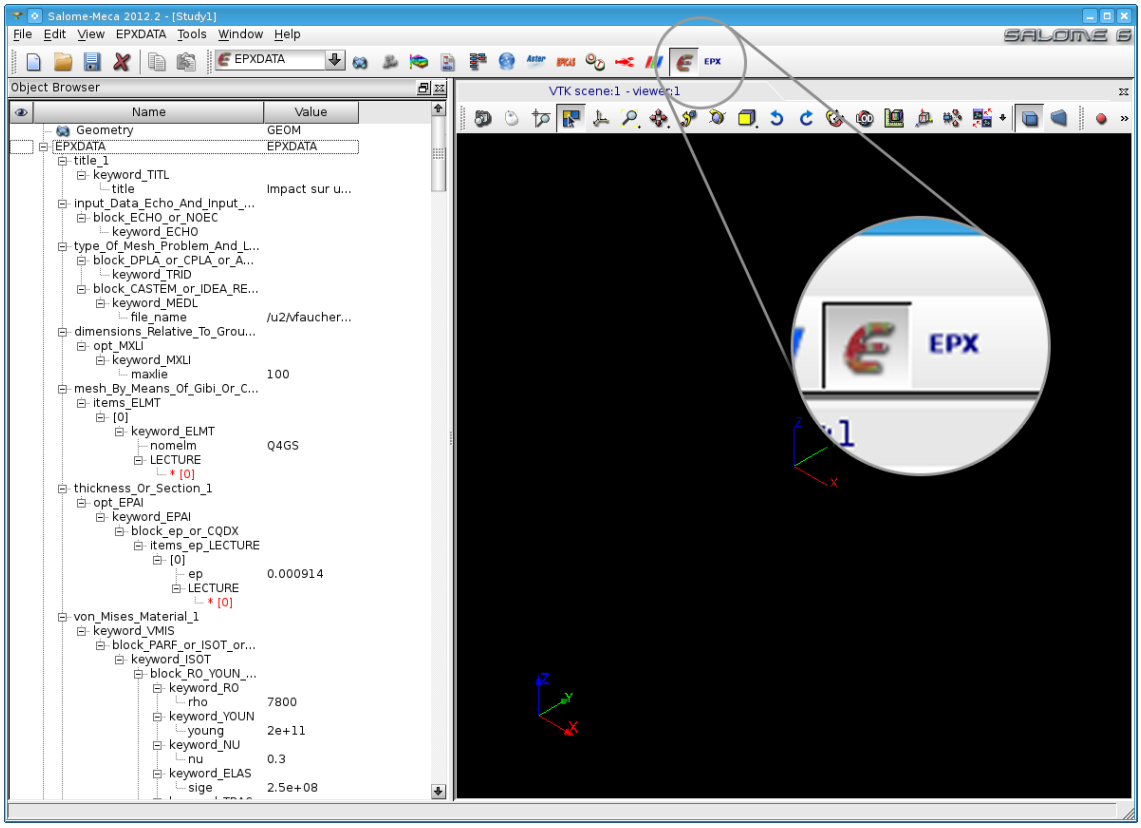


Figure A-1 : Modules EPXDATA et EUROPLEXUS dans SALOME-MECA 2012.2