TFEL/MFront 2.0.3 : note de version

T. Helfer

Octobre 2015

RÉSUMÉ

La version 2.0.3 de `TFEL` est une version de maintenance de la branche $2.0.x$. Par rapport à la version $2.0.2$, il s'agit essentiellement :

- d'un ajout mineur visant à améliorer la gestion des lois de comportement `MFront` dans l'architecture 2.0 (jalon SICOM « MFront ++ »n°71) ;
- de corrections d'anomalies mineures.

Ces évolutions sont détaillées dans cette note.

Nous donnons également différents éléments sur l'état du projet :

- plates-formes supportées ;
- indicateurs de qualité du développement ;
- documentation disponible.

SOMMAIRE

1 INTRODUCTION 3

1.1 OBJET ET PLAN DE CETTE NOTE 3

2 DESCRIPTION DES ÉVOLUTIONS 3

2.1 CORRECTIONS D'ANOMALIES 3

2.1.1 Gestion de lois sans propriété matériau dans MTest pour les interfaces aster et cyrano (Ticket #9) 3

2.1.2 Portage sur la libc++ (Ticket #10) 4

2.1.3 Correction d'une anomalie dans la classe CxxTokenizer 4

2.1.4 Anomalie dans l'interface `Code_Aster` (Ticket #11) 4

2.2 ÉVOLUTION DANS LA GÉNÉRATION DES MODÈLES 4

2.2.1 Paramètres 4

2.2.2 Une correction mineure 5

2.3 INTRODUCTION DE LA CLASSE `ExternalBehaviourDescription` 5

2.4 AMÉLIORATION DE LA GESTION DES BORNES DANS LES LOIS DE COMPORTEMENT 5

2.5 AJOUT DANS LE GLOSSAIRE 6

3 PLATES-FORMES SUPPORTÉES 6

4 INDICATEURS DE LA QUALITÉ DU DÉVELOPPEMENT 7

4.1 CAS TESTS 7

4.2 UN NIVEAU ÉLEVÉ D'AVERTISSEMENTS 8

4.3 ANALYSE STATIQUE DU CODE AVEC L'OUTIL `Coverity` 8

5 DOCUMENTATION 9

5.1 DOCUMENTATION DES MOTS CLÉS MFRONT ET MTEST 10

5.1.1 Documentation en ligne de commande 10

5.1.2 Documentation en ligne 10

5.2 PUBLICATIONS 10

6 DISPONIBILITÉ DU CODE 11

6.1 TÉLÉCHARGEMENT DEPUIS LE SITE INTERNET 11

6.2 ACCÈS AUX SOURCES DEPUIS LE DÉPÔT SVN 11

6.3 POUR LES MEMBRES DU PROJET `PLEIADES` 11

7 CONCLUSIONS 11

7.1 REMERCIEMENTS 12

RÉFÉRENCES 13

1 INTRODUCTION

TFEL est un projet qui fournit entre autres le générateur de code MFront. Il s'agit d'un composant central de la plate-forme Pleiades utilisé par l'architecture Pleiades et par différents composants (ramses, thermo-hydraulique, TMFFT).

TFEL est également utilisé hors de la plate-forme Pleiades au sein de différents départements de la DEN dans des codes tels que Cast3M [CEA 14b] ou AMITEX_FFT [CEA 14a] et par EDF-R&D via le code aux éléments finis Code_Aster [EDF 13] et le projet MAP (Material Ageing Platform).

Le développement de TFEL se poursuit suivant trois branches :

- la branche rliv-2.0 qui est une branche de maintenance de la version 2.0 utilisée en production. Il s'agit de corrections d'anomalies et éventuellement d'ajouts mineurs ;
- la branche trunk est une version de développement toujours basée sur le standard C++-98 dans laquelle différents ajouts sont apportés. Cette branche donnerait naissance à la série des versions $2.1.x$, si cela s'avérait nécessaire ;
- la branche rdev-3.0 est une version de développement basée sur le standard C++-11, ce qui implique une part significative de réécriture du code. Cette branche donnera naissance à la série des versions $3.0.x$.

1.1 OBJET ET PLAN DE CETTE NOTE

Nous décrivons ici la version $2.0.3$ de TFEL. Il s'agit d'une version de maintenance corrigeant les quelques anomalies détectées. Elle introduit également une nouvelle fonctionnalité nécessaire au développement de l'architecture 2.0 (jalon SICOM « MFront ++ »n°71). Ces évolutions sont décrites dans la section 2. Nous donnons dans la section 3 la liste des plates-formes supportées (c'est à dire des couples système d'exploitation/compilateur supportés). La section 4 donne quelques indicateurs sur la qualité du projet. La section 5 fait le point sur la documentation disponible. Enfin, la section 6 décrit comment accéder à cette version.

2 DESCRIPTION DES ÉVOLUTIONS

2.1 CORRECTIONS D'ANOMALIES

Les numéros de ticket correspondent au forum public de tfel consultable à l'adresse suivante :

http://sourceforge.net/p/tfel/tickets

2.1.1 Gestion de lois sans propriété matériau dans MTest pour les interfaces aster et cyrano (Ticket #9)

L'appel depuis MTest des lois de comportement générées par les interfaces aster et cyrano ne gérait pas le cas d'une loi sans propriété matériau.

Le ticket peut être consulté à l'adresse :

https://sourceforge.net/p/tfel/tickets/9/

2.1.2 Portage sur la libc++ (Ticket #10)

Chaque compilateur repose sur une implantation de la bibliothèque standard du C++. Le passage d'une implantation de la bibliothèque standard à une autre peut conduire à différents soucis de portabilité :

- on peut détecter l'oubli de certains fichiers d'entête qui sont inclus de manière indirecte par les autres implantations ;
- on peut détecter des erreurs dans l'utilisation de la bibliothèque standard (voir paragraphe suivant).

Le compilateur clang utilise sur Mac Os ou FreeBSD une bibliothèque nommée libc++. Cette bibliothèque peut également être utilisée de manière optionnelle sous Linux.

Le portage sur cette bibliothèque est une contribution de Thierry Thomas, le mainteneur du portage de Code_Aster sur FreeBSD. Plus de détails peuvent être trouvés à la page :

https://sourceforge.net/p/tfel/tickets/10/

2.1.3 Correction d'une anomalie dans la classe CxxTokenizer

Nous avons trouvé une anomalie importante dans la classe CxxTokenizer lors du portage vers la bibliothèque standard libc++ qui affecte la lecture d'un nombre réel dans les jeux de données. Ceci conduit à l'échec de la plupart des cas tests sur Mac Os ou FreeBSD.

2.1.4 Anomalie dans l'interface Code_Aster (Ticket #11)

La gestion du dépassement des bornes par une des variables n'est pas correctement gérée par l'interface Code_Aster. Plus précisément, la politique décrivant le comportement à suivre n'était pas initialisée [1].

Plus de détails peuvent être trouvés à la page :

https://sourceforge.net/p/tfel/tickets/11/

2.2 Évolution dans la génération des modèles

2.2.1 Paramètres

La classe MFrontModelParser gère l'analyse des modèles. La syntaxe des modèles a peu évolué depuis quelques années, ce qui a conduit à une sémantique obsolète qui peut être incohérente avec les évolutions récentes de la plate-forme Pleiades :

- un paramètre local, défini par le mot clé LocalParameter, désigne un paramètre dont la valeur peut être définie par spécialisation du modèle. L'idée est que ce paramètre est propre au modèle et n'est pas partagé avec d'autres modèles.
- un paramètre global, défini par le mot clé GlobalParameter, désigne un paramètre dont la valeur est définie dans le jeu de données par l'un des mots clés Var, Double, String ou Int. L'idée est que ce paramètre puisse être partagé entre différents modèles. En pratique, les paramètres globaux ne sont pas utilisés.

1. TFEL distingue aujourd'hui trois politiques en cas de dépassement des bornes :
   - None qui ignore le dépassement ;
   - Warning qui signale le dépassement par un message sur la sortie standard ;
   - Strict qui arrête le calcul.

L'adjectif « local » a aujourd'hui tendance à désigner une quantité propre à une tranche (un système physique dans le vocable `Licos`), tandis que « global » se rattache à une quantité utilisée pour le couplage entre les différentes tranches (les différents systèmes physiques).

Nous avons donc introduit les évolutions suivantes dans la classe `MFrontModelParser` :

- nous avons supprimé le mot clé `GlobalParameter` dans cette version ;
- par cohérence avec les propriétés matériau et les lois de comportement, nous avons introduit le mot clé `Parameter` en tant que synonyme de `LocalParameter`.

L'utilisation de `LocalParameter` est considérée comme obsolète et ce mot clé sera supprimé des versions futures au profit du mot clé `Parameter`.

2.2.2 Une correction mineure

D'un point de vue fonctionnel, les modèles utilisent un certain nombre de variables d'état du matériau (représentés par des champs) pour calculer les évolutions d'autres variables internes.

Le calcul de ces évolutions nécessite une certaine connaissance du matériau que l'on peut définir de deux manières :

- par des paramètres (voir paragraphes précédents) ;
- par des propriétés matériau constantes (et uniformes).

Les paramètres sont rattachés au modèle en tant que tel, tandis que les propriétés matériau constantes sont rattachées aux matériaux traités.

L'utilisation d'une propriété matériau constante, représentée par un scalaire, permet d'optimiser le code, par rapport au cas d'une propriété matériau standard, représentée par un champ.

La classe `MFrontModelParser` acceptait que l'on définisse une valeur par défaut à une propriété matériau constante. Ceci est incohérent : une propriété matériau constante est précisément une quantité qu'il faut fournir au modèle et qui lui est extérieure.

Cette possibilité a donc été éliminée. En pratique, les interfaces des modèles n'utilisaient pas cette information.

2.3 INTRODUCTION DE LA CLASSE `ExternalBehaviourDescription`

En plus des lois de comportement, `MFront` enregistre pour chaque loi de comportement un ensemble d'informations. Ces informations permettent de déclarer de manière automatique les lois de comportement et sont déjà utilisées par les applications `Licos`, `cyrano` et `Code_Aster`.

Pour simplifier l'implantation d'une fonctionnalité similaire dans l'architecture V2.0 de `Pleiades`, nous avons introduit une classe nommée `ExternalBehaviourDescription` qui permet de récupérer toutes ces informations par un appel unique.

2.4 AMÉLIORATION DE LA GESTION DES BORNES DANS LES LOIS DE COMPORTEMENT

Rappelons que l'on distingue dans `MFront`, pour une variable donnée :

- ses bornes physiques (une déformation plastique cumulée ne peut être négative, une porosité plus grande que `1`). Ces bornes sont précisées par le mot clé `PhysicalBounds` ;
- la plage de valeurs sur laquelle la loi de comportement/la propriété matériau/le modèle a été établi(e). Ces bornes sont précisées par le mot clé `Bounds`.

La violation d'une borne physique conduit toujours à une erreur.

La résolution du ticket #11 (voir paragraphe 2.1.4) a conduit à revoir la manière de préciser comment traiter un dépassement de bornes (de corrélation) par une variable. Pour rappel, il y a aujourd'hui trois possibilités :

— ne rien faire (choix par défaut, nommé `None` en interne) ;
— afficher un message (`Warning`) ;
— générer une erreur (`Strict`).

Les interfaces `cyrano` et `castem` reposaient sur la définition éventuelle d'une variable d'environnement (`CYRANO_OUT_OF_BOUNDS_POLICY` et `CASTEM_OUT_OF_BOUNDS_POLICY`) qui pouvait prendre les valeurs `WARNING` ou `STRICT` suivant ce que l'on voulait faire.

L'utilisation d'une variable d'environnement s'avère une solution peu pratique.

Pour améliorer cela, les interfaces `aster`, `cyrano` et `castem` génèrent dorénavant une fonction auxiliaire qui permet de modifier ce choix de manière plus souple. Le nom de cette fonction est le nom de la loi suffixé de `_setOutOfBoundsPolicy`. Cette fonction prend un entier en argument :
— une valeur de 0 correspond à `None` ;
— une valeur de 1 correspond à `Warning` ;
— une valeur de 2 correspond à `Strict`.

Ces fonctions n'ont pas vocation à être appelées directement : une méthode supplémentaire, nommée `setOutOfBoundsPolicy`, a été ajoutée à la classe `ExternalLibraryManager` pour simplifier leur utilisation.

Choix de traitement des dépassements de bornes dans Code_Aster La version de développement de `Code_Aster` permet à l'utilisateur de choisir le comportement à adopter en cas de dépassement des bornes de la corrélation à l'aide du mot-clé `VERI_BORNE`. Ce mot clé a trois options : `SANS`, `MESSAGE` et `ARRET` correspondant respectivement à `None`, `Warning` et `Strict`. Par défaut, cette option est à `ARRET` (`Strict`).

2.5 AJOUT DANS LE GLOSSAIRE

L'entrée `AxialGrowth` a été ajoutée dans le glossaire. Elle correspond à la croissance axiale sous flux (grandissement) que l'on rencontre dans les crayons combustibles.

3 PLATES-FORMES SUPPORTÉES

Plate-forme	Versions testées	Versions fonctionnant a priori
`gcc` sous LiNuX	4.4, 4.9	$\geq$ 3.4
`clang` sous LiNuX (`libstdc++`)	3.4, 3.5, 3.6	$\geq$ 3.2
`clang` sous LiNuX (`libc++`)	3.5	$\geq$ 3.5
`intel` sous LiNuX	15, 16	$\geq$ 10
`clang` sous FreeBSD (`libc++`)	3.4	$\geq$ 3.4
`mingw64` sous Windows	5.1, $4.6.3$	$\geq$ 4.6

TABLEAU 1 : Plates-formes supportées. Nous indiquons les versions des compilateurs utilisés pour réaliser les tests de cette version. Nous indiquons aussi, à titre indicatif, les versions que nous pensons être compatibles `a priori`. Ce sont généralement des versions utilisées dans le passé pour les versions précédentes de `TFEL`.

Le tableau 1 donne la liste des combinaisons de compilateur et de système d'exploitation supportées.

Accroître le nombre de plate-formes supportées présente essentiellement deux intérêts :
— cela accroît le nombre d'utilisateurs potentiels, c'est à dire, de testeurs des différentes fonctionnalités proposées. Il s'agit de l'effet de bord le plus positif de la mise en open-source ;
— c'est un signe indirect de la qualité du code, d'autant que chaque compilateur peut émettre des avertissements qui lui sont spécifiques (voir paragraphe 4.2 ci-dessous).

Soulignons tout l'intérêt de l'atelier logiciel Pleiades[2], et notamment l'application jenkins[3], qui est très utile, voire indispensable, pour gérer autant de combinaisons.

Notes générales sur les différents compilateurs gcc et clang sont les compilateurs de référence lors du développement TFEL. Les versions récentes de clang offrent des performances comparables à celle de gcc si l'on utilise l'implantation libstdc++ de la bibliothèque standard. Le temps d'exécution des cas tests est sensiblement réduit si l'on utilise l'implantation libc++. Ceci est essentiellement dû à une réduction du temps d'exécution de quelques cas tests particulièrement consommateurs de ressources : il s'agit de lois à plusieurs milliers de variables internes qui font appel à une gestion dynamique de la mémoire.

Le compilateur intel est le compilateur offrant les meilleures performances. Par exemple, le temps d'exécution de l'ensemble des cas tests est 20% plus court avec ce compilateur par rapport aux temps obtenus avec les compilateurs gcc et clang.

Notons que nous avons accès sans frais à ce compilateur grâce à une offre d'Intel faite aux projets open-source.

Le compilateur MinGW Nous avons malheureusement pu constater que le compilateur MinGW, qui est le portage sous la plate-forme Windows du compilateur gcc, est de qualité variable d'une version à l'autre. Donnons quelques exemples :

- la version 4.8.3 ne définit pas la constante symbolique SSIZE_MAX, définie par les versions précédentes et suivantes ;
- la version 5.1.0 ne définit pas les constantes symboliques M_PI, M_SQRT2 ou M_SQRT2_2.

La version livrée avec les version 14 et 15 de Cast3M a un bug gênant lors de la conversion d'une chaîne de caractère en double : '0' n'est pas égal au nombre 0 mais vaut $1.e-317$. Ceci conduit à l'échec de certains tests qui posent des bornes sur les valeurs des variables internes : nous déclarons que la déformation plastique cumulée ne peut être négative. Or, en début de calcul, la déformation plastique est nulle et nous avons un échec dû au fait que 0 est inférieur à $1.e-317$.

Le compilateur Visual C++ de Microsoft Le support de la plate-forme Windows est considéré comme partiel. En effet, mingw n'est pas le compilateur officiel de cette plate-forme qui est le compilateur Microsoft. Or, le support de ce compilateur demande un portage important. Ce portage a été effectué pour la version 3.0 : cette version, basée sur la norme C++11, est allégée de quelques morceaux de codes particulièrement pointus qui font maintenant partie du standard et qui auraient été particulièrement difficiles à implanter.

Mac Os Il est possible de compiler cette version sous Mac OS. Ce système présente cependant des subtilités[4] qui ne seront pleinement prises en charge qu'à la version 3.0 de TFEL. Le support de ce système, bien qu'a priori fonctionnel (l'ensemble des cas tests passent), est donc considéré comme expérimental. D'éventuels testeurs sont les bienvenus.

4 INDICATEURS DE LA QUALITÉ DU DÉVELOPPEMENT

4.1 CAS TESTS

Cette version est livrée avec 709 cas tests.

2. https://www-pleiades.intra.cea.fr/jenkins/
3. https://jenkins-ci.org/
4. Mac OS est l'un des rares systèmes faisant la distinction entre bibliothèques partagées et modules chargés dynamiquement.

548 cas tests sont des cas tests `mtest` qui couvrent une grande partie des fonctionnalités de `mfront` associées aux lois de comportement.

Les cas tests restants permettent de tester de manière unitaire différentes fonctionnalités de `TFEL`. La plupart des cas tests visent la partie `TFEL/Math` qui regroupe les objets mathématiques (vecteurs, matrices, tenseurs) ainsi que différents algorithmes numériques.

Les cas tests unitaires sont aujourd'hui insuffisants pour assurer une couverture de l'ensemble des fonctionnalités du code et un effort dans ce sens nous apparaît nécessaire. Cela est en grande partie dû au fait qu'une partie non négligeable du code de `MFront` est liée au traitement des erreurs dans les fichiers d'entrée et à la vérification de la cohérence des données fournies par l'utilisateur : il faudrait, pour tester ces parties de codes, créer des cas tests spécifiques. Une première analyse de la couverture du code, qui sera affinée dans les versions futures montre cependant que les parties numériques sont couvertes de manière assez satisfaisante : à grosse maille, on peut estimer que le taux de couverture pour ces parties est supérieur à $80\,\%$.

4.2 UN NIVEAU ÉLEVÉ D'AVERTISSEMENTS

Les compilateurs modernes permettent de détecter des erreurs de programmation ou des mauvaises pratiques en activant des flags de compilations spécifiques. Ceux utilisés pour la compilation de `TFEL` sont particulièrement exigeants :

```
-Wdisabled-optimization -Wlong-long -Wreorder
-Wundef -Wunknown-pragmas -Wredundant-decls -Wpacked
-Wno-deprecated-declarations -Wno-multichar
-Wmissing-format-attribute -Wsign-compare -Wno-endif-labels
-Wfloat-equal -Wreturn-type -Woverloaded-virtual
-Wnon-virtual-dtor -Wctor-dtor-privacy -Wwrite-strings
-Wcast-align -Wcast-qual -Wpointer-arith -Wshadow -pedantic
-Wextra -W -Wall
```

La plupart des avertissements générés sont corrigés.

4.3 ANALYSE STATIQUE DU CODE AVEC L'OUTIL `Coverity`

Nous avons utilisé lors du développement de `TFEL` différents outils d'analyse statique open-source : `scan-build`[5] ou `cppcheck`[6] par exemple. Bien qu'ils soient utiles, ces outils ne sont pas au niveau de certains outils propriétaires.

`Coverity` est une société spécialisée dans l'analyse statique de code. Cette société propose aux codes diffusés sous une licence libre de profiter de leur outil[7].

L'analyse statique permet de révéler la présence de défauts qui peuvent être soit des erreurs de programmation, soit des mauvaises pratiques. Le taux de défauts est le nombre de défauts pour 1000 lignes de code. `Coverity` indique qu'un logiciel dont le taux de défauts est inférieur à 1 est considéré de qualité industrielle.

L'analyse statique de `TFEL` (hors tests) révèle une « densité de défauts » de l'ordre de $0,3$. `Coverity` a permis de trouver une erreur « réelle », sans incidence sur les applications ou les utilisateurs. Cette erreur a depuis été corrigée.

Les autres défauts détectés sont essentiellement des « fautes de styles » sans incidence sur le comportement

5. `http://clang-analyzer.llvm.org/scan-build.html`
6. `http://cppcheck.sourceforge.net/`
7. `http://www.coverity.com/`

du code ou le fait que des données membre de structure n'ont pas de valeurs par défaut[8]. Ces fautes de style seront corrigées dans les futures versions.

Lorsque l'on inclut les tests, on trouve essentiellement des défauts supplémentaires liés au fait que :

- les fonctions principales des tests ne captent pas les exceptions potentiellement lancées par les fonctions appelées, ce qui est volontaire. En effet, une exception non traitée peut être analysée à l'aide d'un débogueur : on peut remonter au point d'émission de l'exception.
- les variables locales des lois de comportement ne sont pas initialisées dans les constructeurs de la loi mais dans une fonction membre dédiée, pour des raisons qu'il serait trop long d'exposer ici. Nous pourrions donner une valeur par défaut à ces variables, mais cela pose au moins deux problèmes :
  - cette initialisation a un coût, particulièrement si des tableaux de grandes tailles sont utilisés (certaines lois ont plusieurs milliers de variables internes).
  - l'utilisation d'une variable non initialisée est souvent plus facile à détecter, surtout si l'on utilise un outil de type `valgrind`[9].

  Néanmoins, il nous semble possible d'analyser l'initialisation des variables locales et de la déporter dans le constructeur quand cela est possible (ce qui correspond à la majorité des cas).

Dans l'ensemble, on peut raisonnablement être satisfait de la qualité du code de `TFEL` telle que mesurée par cet indicateur.

5 DOCUMENTATION

Le code est livré avec la documentation suivante :

- notes de référence `MFront` [Helfer 13a, Helfer 13b, Helfer 14a] ;
- notes de référence `MTest` [Helfer 15] ;
- un tutoriel [Helfer 14b].

On pourra également trouver des cours de formation ainsi qu'un support de travaux dirigés.

Ces documents sont accessibles à la page suivante :

`http://tfel.sourceforge.net/documentations.html`

À l'heure actuelle, il n'existe pas encore de guide de référence du code (qui permettrait d'en comprendre la structuration) ni de guide des cas tests.

Concernant les cas tests, nous avions développé l'utilitaire `tfel-doc` qui a été utilisé pour décrire les cas tests de l'application `Licos` [Helfer 12]. Nous pensons développer dans la version $3.0.x$ une extension à l'utilitaire `tfel-doc` pour extraire une documentation des fichiers `MTest`. Cependant, nous pensons qu'une version papier de cette documentation n'a que peu d'intérêt. Il serait intéressant de réfléchir à une forme plus adaptée au besoin des utilisateurs. Par exemple, il semble facile de concevoir une application graphique qui permettrait une recherche par mots clés à partir d'une base de données générée par `tfel-doc`.

Enfin, le code source est documenté au format `doxygen`. Cependant, cette documentation est loin de couvrir l'intégralité des fonctions et des classes.

8. Nous utilisons beaucoup de structures qui sont renseignées par une fonction spécifique. Nous accédons à ces données en lecture seule dans d'autres parties du code. Une telle pratique est saine et génère moins de code que l'emploi d'un constructeur. L'outil `Coverity`, ne pouvant comprendre la logique du développeur, émet un avertissement de principe. Il faut également noter que l'utilisation de variables non initialisées est souvent détectée par le compilateur ou les outils d'analyse statique. Par ailleurs, un outil de type `valgrind` permet également de détecter ces erreurs à l'exécution.

9. `http://valgrind.org/`

5.1 DOCUMENTATION DES MOTS CLÉS MFRONT ET MTEST

5.1.1 Documentation en ligne de commande

S'inspirant de l'outil `cmake`[10], la documentation des mots clés de `MFront` et de `MTest` est accessible en ligne de commande à l'aide des options suivantes :

- `--help-keywords-list`. Cette option affiche la liste des mots clés suivi de la mention `documented` ou `undocumented` suivant les cas. Pour `MFront`, il faut préciser le nom de l'analyseur pour lequel on veut avoir la liste des mots clés.
- `--help-keyword`. Cette option permet d'afficher l'aide associée à un mot clé spécifique. Pour `MFront`, il faut là aussi préciser le nom de l'analyseur.

La documentation des mots clés est écrite en anglais, au format `Markdown`[11]. Ce format est clair et lisible. Il présente l'avantage de pouvoir être analysé par l'utilitaire `pandoc`[12] et transformé dans différents formats, dont le format `html`. Ceci a été utilisé pour générer la documentation en ligne de ces mots clés.

5.1.2 Documentation en ligne

La documentation des mots clés de `MFront` est accessible aux pages suivantes :

- `http://tfel.sourceforge.net/DefaultDSL-keywords.html`
- `http://tfel.sourceforge.net/DefaultFiniteStrainDSL-keywords.html`
- `http://tfel.sourceforge.net/DefaultCZMDSL-keywords.html`
- `http://tfel.sourceforge.net/Implicit-keywords.html`
- `http://tfel.sourceforge.net/ImplicitFiniteStrain-keywords.html`
- `http://tfel.sourceforge.net/ImplicitII-keywords.html`
- `http://tfel.sourceforge.net/IsotropicMisesCreep-keywords.html`
- `http://tfel.sourceforge.net/IsotropicPlasticMisesFlow-keywords.html`
- `http://tfel.sourceforge.net/IsotropicStrainHardeningMisesCreep-keywords.html`
- `http://tfel.sourceforge.net/MaterialLaw-keywords.html`
- `http://tfel.sourceforge.net/Model-keywords.html`
- `http://tfel.sourceforge.net/MultipleIsotropicMisesFlows-keywords.html`
- `http://tfel.sourceforge.net/RungeKutta-keywords.html`

La documentation des mots clés `MTest` est accessible à la page :

`http://tfel.sourceforge.net/MTest-keywords.html`

5.2 PUBLICATIONS

Une liste des publications mentionnant `TFEL` ou `MFront` peut être trouvée à la page :

`http://tfel.sourceforge.net/publications.html`

6 DISPONIBILITÉ DU CODE

10. `http://www.cmake.org/`
11. `http://pandoc.org/README.html#pandocs-markdown`
12. `http://pandoc.org/`

6.1 TÉLÉCHARGEMENT DEPUIS LE SITE INTERNET

Le code source de TFEL est disponible sur le site internet du projet :

http://tfel.sourceforge.net

Les instructions pour compiler cette version sont disponibles à l'adresse :

http://tfel.sourceforge.net/install.html

6.2 ACCÈS AUX SOURCES DEPUIS LE DÉPÔT SVN

Les utilisateurs autorisés peuvent accéder au dépôt SVN du projet à l'adresse suivante :

https://svn-pleiades.cea.fr/SVN/TFEL/tags/TFEL-2.0.3

6.3 POUR LES MEMBRES DU PROJET PLEIADES

Les membres du projet PLEIADES peuvent accéder aux binaires de cette version dans le répertoire des prérequis de développement de la version 2.0 de la plate-forme [13] :

source /soft/pleiades/testing/BUILDS/PLEIADES-trunk/PleiadesEnv.sh

7 CONCLUSIONS

Pour les besoins de l'architecture V2.0 du projet PLEIADES, une version $2.0.3$ du projet TFEL/MFront a été produite. Cette note a présenté les évolutions qui ont conduit à cette nouvelle version.

Cette version est disponible sur la majorité des systèmes et peut être compilée avec la plupart des compilateurs, à l'exception notable du compilateur Visual C++ de Microsoft. La version 3.0 de TFEL/MFront prendra également ce compilateur en charge. Le support de Mac OS est aujourd'hui considéré comme expérimental et sera pleinement supporté par la version 3.0.

Cette note a également mis en avant les efforts faits dans le domaine de la qualité du code, que ce soit via les cas tests, ou via l'utilisation d'outils de développement tels que les analyseurs statiques de code. Le nombre de défauts trouvés par Coverity, si l'on exclut les erreurs de style, classe TFEL parmi des produits de très bonne qualité.

Des efforts sont cependant encore nécessaires :

- le taux de couverture des tests unitaires est insuffisant ;
- la documentation du code est encore trop partielle pour être jugée satisfaisante.

Nous avons également évoqué la question de la documentation des cas tests. Une évolution de tfel-doc permettrait de générer automatiquement la documentation de la plupart des cas tests basés sur MTest. Cependant, un document papier nous semble n'avoir que peu d'utilité. D'un point de vue utilisateur, il serait plus intéressant de disposer d'une application graphique permettant une recherche par mots clés à partir d'une base de données générée par tfel-doc.

13. Voir https://www-pleiades.intra.cea.fr/trac/wiki/ArchSoft.

7.1 REMERCIEMENTS

La page suivante donne les noms des principaux contributeurs à TFEL :

http://tfel.sourceforge.net/about.html

RÉFÉRENCES

[CEA 14a] CEA . *AMITEX_FFT*, 2014. http://www.maisondelasimulation.fr/projects/amitex/html/.

[CEA 14b] CEA . *Cast3M Web Site*, 2014. http://www-cast3m.cea.fr/.

[EDF 13] EDF . *Code_Aster Web Site*, 2013. http://www.code-aster.org.

[Helfer 12] HELFER THOMAS. *Cas tests unitaires de la version 1.0 de Licos*. Note technique 12-005, CEA DEN/DEC/SESC/LSC, Avril 2012.

[Helfer 13a] HELFER THOMAS et CASTELIER ÉTIENNE. *Le générateur de code mfront : présentation générale et application aux propriétés matériau et aux modèles*. Note technique 13-019, CEA DEN/DEC/SESC/LSC, Décembre 2013.

[Helfer 13b] HELFER THOMAS, CASTELIER ÉTIENNE, BLANC VICTOR et JULIEN JÉRÔME. *Le générateur de code mfront : écriture de lois de comportement mécanique*. Note technique 13-020, CEA DEN/DEC/SESC/LSC, Décembre 2013.

[Helfer 14a] HELFER THOMAS. *Prise en compte des hypothèses de contraintes planes dans les lois générées par le générateur de code MFront : application à Cyrano 3*. Note technique 14-013, CEA DEN/DEC/SESC/LSC, Septembre 2014.

[Helfer 14b] HELFER THOMAS et PROIX JEAN-MICHEL. *Écriture de lois de comportement avec MFront : tutoriel*. Note technique 14-023, CEA DEN/DEC/SESC/LSC, Décembre 2014.

[Helfer 15] HELFER THOMAS et PROIX JEAN-MICHEL. *MTest : Un outil de simulation du comportement mécanique d'un point matériel*. Note technique 15-010, CEA DEN/DEC/SESC/LSC, Juillet 2015.