Utilisation de la norme XML en éléments finis

Gérard Rio, Laëtitia Duigou

Utilisation de la norme XML en éléments finis

Gérard Rio, Laëtitia Duigou

Universite de Bretagne Sud - Centre de Recherche
Laboratoire de Génie Mécanique et Matériaux
Rue de Saint Maudé BP 92116, 56321 Lorient cedex
gerard.rio@univ-ubs.fr

Résumé :

La présentation concerne l'utilisation de la nouvelle norme XML dans le cadre des éléments finis en mécanique. Les entités manipulées lors du calcul apparaissent comme des candidats naturels à un traitement et à un formatage utilisant la norme XML aux travers des Schémas. Les outils associés (css, sax, dom ...) sont alors particulièrement intéressants en pré et post-traitement ainsi qu'en Restart.

Abstract :

The presentation relates to the use of new standard XML within the framework of the finite elements in mechanics. The entities handled during calculation seem natural candidates for a treatment and a formatting using standard XML through Schema. The associated tools (css, sax, dom...) are then particularly interesting into pre and postprocessing and restart.

Mots-clefs :

XML ; éléments finis ; format d'échanges de données ; API DOM SAX

1 Introduction

L'objectif est ici de mettre en place une utilisation rationnelle de la norme XML dans le cadre de nos développements qui utilisent la méthode des éléments finis. L'idée est ainsi de définir une syntaxe générale en mode texte, fortement typée, permettant d'exprimer les grandeurs manipulées dans le cadre des différentes entrées-sorties.

1.1 Contexte et état de l'art

D'une manière générale, de nombreux travaux ont été consacrés à la définition et la mise en place de formats dédiés aux calculs scientifiques avec comme objectif de favoriser l'échange entre des logiciels d'origines différentes. HDF5 développé par le "National Center for Supercomputing Applications" (NCSA) [1] en constitue un exemple représentatif. HDF5 est un format de stockage auto-documenté, qui est proposé avec une librairie de fonctions permettant l'ensemble des manipulations courantes. La librairie est libre, et permet de manipuler des informations de très grandes tailles. L'intérêt du format HDF est donc indéniable, cependant son utilisation est restreinte aux logiciels qui utilisent la librairie associée. A l'inverse, tout format fondé sur la norme XML, peut être exploité par un nombre important de logiciels d'horizons très divers : outils de formatage, outils du web ... De plus un format de type HDF n'est pas adapté à la "validation" (au sens XML) des données particulièrement utiles pour la constitution d'un fichier de commande de calculs éléments finis. C'est pourquoi nous nous intéressons à l'utilisation du XML.

Plusieurs travaux ont déjà été réalisés en vue de définir un format d'échange en XML. Par exemple, le projet femML [2] présenté par J. Michopoulos avait pour objectif de définir un standart général dédié aux éléments finis. Le formatage des données s'appuie principalement sur un fichier DTD ("Document Type Definition" [3]), qui constitue la première forme historique de formatage des fichiers XML. En fait, le typage obtenu avec une DTD est assez restreint d'où la mise en place des "Schema" XML (proposition 2001 du W3C) qui permettent un typage beaucoup plus riche. Cependant, du fait, sans doute, de la jeunesse des Schema, nous avons trouvé très peu de Schema dédiés au domaine des éléments finis.

Trois réalisations sont assez représentatives de la bibliographie. XSIL [4] [5] "Extensible Scientific Interchange Language" a été développé dans les années 2000 et s'appuie principalement sur une DTD. Un schéma existe également, mais simple et utilisant très peu les nouvelles possibilités des schémas. XDMF [6] "eXtensible Data Model and Format" qui est utilisé au sein du projet ICE "Interdisciplinary Computing Environment" est une surcouche XML au format HDF5. Les informations sont décomposées en deux catégories : les données "légères" (une date, un nombre, une référence) sont directement gérées en XML, les données complexes et volumineuses sont gérées en HDF5. L'idée est donc d'essayer d'utiliser l'existant (HDF5) tout en cherchant à bénéficier des nouvelles technologies XML. L'ensemble reste finalement complexe et dédié aux gros projets, il contient les avantages et inconvénients initiaux du format HDF5. Enfin certains codes de calcul, voir par exemple FeViaduct [7], XMDS [8], utilisent un système d'entrées-sorties XML. En entrée l'intérêt est de pouvoir valider des données, et en sortie de faciliter les transferts de données pour le post-traitement. Cependant, les DTD ou schémas utilisés restent simples (XMDS) ou inconnus (FeViaduct).

1.2 Intérêt d'XML

La norme XML fournit d'une part, des concepts permettant d'établir une grammaire adéquate à notre objectif, et d'autre part, un ensemble d'outils permettant de traiter les informations conformes au langage établi. Cette norme constitue ainsi pour nous un guide de formatage des données. Le fait qu'il s'agisse d'une norme internationale, utilisée dans de nombreux domaines, est une garantie de pérennité.

Les documents XML comportent la nécessité d'être "bien formés" c'est-à-dire d'obéir aux règles syntaxiques du langage XML. Différents outils permettent de vérifier cette qualité, en particulier les "parseurs" (voir par exemple Xerces de la fondation Apache [9]) et les éditeurs (exemple : Morphon [10] qui comportent également en interne un parseur). Le critère "bien formés" pourra être ainsi vérifié avant toute utilisation par le code de calcul. Un document XML peut également être "valide" par rapport à des règles fixées. Par exemple, on peut ainsi vérifier l'existence de différents mots clés, associés à un champ de valeurs possibles, un nombre d'occurrences, un type etc. Ceci correspond tout-à-fait à l'objectif que l'on recherche, à savoir construire un document texte, contenant des données, qui puisse être validé au regard d'un ensemble de règles (en plus des règles syntaxiques) qui garantissent qu'il n'y ait pas d'erreur de lecture et de compréhension lors de leur utilisation par le programme de calcul.

Initialement, XML n'a pas été conçu pour établir des règles concernant particulièrement des données scientifiques. En fait il s'adressait plutôt aux données textuelles (livres, articles, rapports), associées à des objectifs divers : stockage pérenne, format d'échange rationnel, outils de manipulations via le web ... Cependant, les possibilités introduites dans XML se sont révélées également pertinentes dans le cadre du traitement des données numériques (voir par exemple les interactions avec les bases de données [11]).

Initialement, le formatage des données s'est appuyé sur les DTD qui permettent de définir des structures de types de documents bien adaptés aux documents traditionnels dans le cadre de l'édition, l'affichage et l'impression. Dans le cadre des échanges de données structurées, les DTD souffrent d'un certain nombre de lacunes ([12],[13]) par exemple au niveau des domaines nominaux (namespace). Ainsi, bien que toujours utilisés, les DTD peuvent être avantageusement remplacés par les "schémas XML" apparus à partir de 1999 et finalisés en 2001 (cf. ressources W3C [3]). Les travaux présentés s'inscrivent dans ce contexte.

2 Constitution d'un Schéma général

On entend ici par Schema la définition du schéma XML associé aux structures de données que nous utilisons dans notre code éléments finis Herezh++ [14] développé en C++ suivant une approche objet. Une caractéristique intéressante des schémas est d'utiliser une structuration très proche des langages objets. D'une manière pratique, le schéma se présente sous forme d'un fichier texte, écrit en XML, lisible par tout traitement texte. Par exemple la présentation s'effectue avec des "balises" qui, contrairement au langage HTML, peuvent être choisies librement.

Nous distinguons a priori dans le schéma deux niveaux de données. Le premier concerne les types de bases classiques, tels que les coordonnées, les vecteurs au sens large c'est-à-dire sans limite de taille, les tenseurs, les matrices, les chaines de caractères particulières etc. Le second niveau est relatif aux classes plus complexes qui intègrent en général un grand nombre d'objets du premier niveau, et dont la structure interne peut en général se représenter sous forme d'arbres, contrairement aux types de bases. Concernant la portée des types, le premier niveau doit évidemment avoir une portée globale. Pour le second niveau, il est possible de restreindre la portée de certains types d'usages strictement internes, à une arborescence donnée. Ce dispositif permet notamment d'utiliser plusieurs fois un même identificateur générique pour plusieurs groupes de données différents suivant le contexte. Le schéma XML permet naturellement d'effectuer cette distinction. Les types de premier niveau sont implantés directement sous la racine ce qui leur assure une portée générale, alors que les types de second niveau peuvent être déclarés soit internes à un type plus général, et donc dans ce cas avec une portée restreinte ou soit également directement sous la racine.

L'information s'introduit sous forme d' "éléments" XML constitués d'un nom (utilisé comme balise), d'un type et d'attributs éventuels également typés. Ainsi un type unique peut être utilisé de nombreuse fois.

Nous introduisons ainsi, à la suite des définitions des types globaux, la définition d'un "élément" (au sens XML) particulier pour chaque entrée-sortie : un élément pour le fichier d'entrées de données et de commandes, un élément pour le fichier "restart", un élément pour les commandes de post-traitement ....

Un point également important est la distinction entre attribut et information interne dans une expression entre balises. A priori les spécialistes de bases de donnée sont plus favorables à l'utilisation de la notion d'attribut pour stocker les données en particulier, grâce à la concision de l'écriture dans le cas d'un grand nombre d'attributs. Cependant, l'utilisation des grandeurs attributs comporte des inconvénients majeurs : il n'est pas possible d'imposer l'ordre de leur apparition et l'accès dans l'arborescence globale des informations n'est pas possible via "un chemin direct" XPath. Pour ces raisons, nous utiliserons de préférence des "éléments" internes qui encapsulent les informations entre une balise ouvrante et une balise fermante.

Dans une première partie, nous examinons un certain nombre de propriétés utiles pour les structures de données que nous voulons mettre en oeuvre puis, dans une seconde partie, nous

introduisons des éléments d'implantation.

2.1 Quelques propriétés utiles

Il n'est pas possible ici de faire une présentation exhaustive de l'ensemble des possibilités permises par les schémas. Cependant, nous rappelons quelques propriétés intéressantes adaptées aux éléments finis.

Tout d'abord, on retrouve les différents types de bases classiques : chaîne de caractères (”string”), booléen (”boolean”), les réelles en simple (”float”) et double précision (”double”), les entiers signés ou non (”integer”, ”long”, ”int”, ”short” ...). Pour être plus précis différents types particuliers sont également déclinés comme : le temps absolu ou relatif, la durée, les dates récurrentes, les nombres décimaux (”decimal”). Ces derniers types seront, par exemple, utilisés pour décrire les paramètres temporels de calculs en transitoire ou dynamique.

Ces types de bases peuvent être complétés par des ”facettes” qui constituent des restrictions du type initial. Parmi les 14 facettes proposées par XML, on note la possibilité de définir une valeur minimale (”minInclusive” ou ”minExclusive”), une valeur maximale (”maxInclusive” ou ”maxExclusive”). Il est également possible d'introduire des ”extensions” aux types initiaux. Une succession de nombres, par exemple les coordonnées d'un tenseur, sera prise en compte par le type ”list”.

Deux types de commentaires sont disponibles : des commentaires dans le texte (usage classique), mais également de la documentation qui sera ensuite traitée par des outils standards d'extraction et de formatage.

Comme structure de base intéressante, pour les éléments on dispose : de ”sequence” qui permet d'imposer la présence ordonnée d'un ensemble d'éléments ; de”choice” qui au contraire impose un choix unique entre plusieurs possibilités et enfin ”all” qui permet un ordre quelconque d'apparition d'éléments. Ainsi pour introduire une chronologie dans le fichier XML de commande (par exemple, la dimension devra être tout d'abord définie avant d'introduire le maillage), nous utiliserons ”sequence” et il est possible de gérer le nombre d'apparitions d'éléments via les attributs : ”minOccurs”, ”maxOccurs”, ce dernier pouvant être non borné.

Les éléments sont susceptibles d' être groupés, d'appartenir à d'autres éléments, d'être présentés sous forme générique ... ce qui met en évidence la possibilité de définir une hiérarchie tout-à-fait analogue à celle rencontrée au sein des objets (ou classes) en langage orienté objet.

Enfin, plusieurs schémas d'origines différentes, peuvent être pris simultanément en compte au travers des domaines de noms (d'une manière équivalente au ”namespace” du C++), point utile par exemple pour définir une interface entre des logiciels d'origines diverses.

2.2 Implantation

XML offre en standard deux API (Application Programming Interface) permettant de lire, manipuler, transformer et écrire des informations XML. En fait, le W3C fournit des recommandations pour les API, mais leurs mises en oeuvre nécessitent l'utilisation d'une implémentation particulière.

D'une manière générale, deux grandes tendances existent, soit l'utilisation d'outils indépendants qui permettent par exemple de valider un document (vérification de la syntaxe, des règles ...), soit l'utilisation d'une bibliothèque permettant une jonction directe avec l'outil de calculs. De nombreux traitements standards sont alors possibles comme le filtrage et la transformation de sous-parties. Ceci se révèlera particulièrement appréciable dans le cadre du post-traitement pour

isoler des informations particulières et les transformer en format directement exploitable par des outils classiques de post-traitement : tracés de courbes, éditions de documents en pdf ou html ... L'utilisation de langages dédiés dans ce sens comme le PERL (cf. [15]) ou PYTHON, qui intègre en standard un Parser (API) XML, est alors très simple et performante. Il est à noter l'importance de ces deux langages, dans le cadre du calcul scientifique.

Pour notre part, nous nous sommes plus particulièrement intéressé à l'intégration des API dans le code de calcul, via une bibliothèque associée. Dû au foisonnement actuel du développement d'outils XML, de nombreuses possibilités existent. En particulier on notera les différentes librairies LIBXML existant sous tous les systèmes LINUX. Nous avons opté pour la librairie Xerces proposée par "The Apache Software Foundation" (cf. [9]) qui offre une implantation en C++ plus aboutie, conforme à une grande part des préconisations du W3C les plus récentes. De plus, les outils semblent particulièrement robustes et performants.

Pour notre objectif, deux principales API sont envisageables : SAX et DOM. La première API permet de parser simplement un flot d'informations. Au fil et en fonction de la lecture, elle génère des événements qui doivent être capturés par des méthodes implantées par l'utilisateur. Par exemple à la lecture d'une balise ouvrante d'un élément, SAX signale cette ouverture et peut ramener les différentes entités de l'élément lu. SAX agit ainsi comme un outil de lecture intelligent, qui tout en lisant les grandeurs typées, vérifie la cohérence des grandeurs avec le schéma ou le DTD associé, on parle de parser validant. L'intérêt immédiat est de ne pas avoir à se préoccuper de la mise en page du texte (enregistrement sur une ou plusieurs lignes ...). Cependant SAX ne stocke pas l'ensemble des informations, il est donc nécessaire d'enregistrer au fur et à mesure de la lecture, les données dans les instances de classes utilisateur associé. Dans le cadre d'une conception objet, lors de la lecture d'informations faisant intervenir un grand nombre de classes d'une manière non linéaire, ce fonctionnement par "gestion d'interruption" n'est pas toujours pratique, une méthode alternative proposée par Xerces est un fonctionnement pas à pas.

L'API DOM, effectue la lecture de données tout en construisant un arbre (ou bosquet) de stockage de ses informations. DOM fournit ensuite un ensemble de méthodes objets permettant de récupérer ces données, voir de les modifier. DOM est plus complexe et plus lente que SAX mais permet une interrogation beaucoup plus souple, en particulier par chaque objet de la structure éléments finis, de manière à générer directement et de manière indépendante les informations encapsulées. Elle est évidemment validante, ainsi on est sûr de disposer de données conformes au schéma. Dans le cas de petites tailles de fichiers (de l'ordre de quelques Mo maximum), cas typique d'un fichier d'entrées, l'API DOM apparaît alors tout-à-fait adaptée. Cependant l'utilisation de DOM aboutit à doubler la place de stockage au moment de la lecture : stockage avec DOM et stockage dans les objets.

En résumé, nous retenons les conclusions suivantes pour nos développements. L'API SAX permet de traiter aisément et efficacement des fichiers de petites ou de grandes tailles ou des fichiers à structures simples, comme un passage typé et fréquent de données entre des logiciels différents. L'API DOM est quant à elle plus intéressante pour des fichiers à structures complexes de faible taille, tels certains fichiers de commande. Cependant l'utilisation d'un parser introduit un temps de traitement supplémentaire. Dans le cas de flots de données situées en interface avec l'utilisateur ou des programmes externes, ce surcoût est la contrepartie au contrôle de la qualité des données. Dans le cas de données entièrement gérées par le logiciel de calcul, par exemple les données nécessaires à un redémarrage (fichier "restart") induisant parfois de très gros fichiers (100Mo, 1Go...), l'idée a été pour l'instant de conserver les méthodes d'entrée-sortie classiques des flots de données C++, tout en adoptant un formatage XML. En restart, la lecture est alors

réalisée sans vérification de types afin d'optimiser le temps de traitement.

3 Conclusion

L'étude concerne l'utilisation de la nouvelle norme XML dans le cadre des différentes entrées-sorties pour un calcul éléments finis. Les entités manipulées lors du calcul apparaissent comme des candidats naturels à un traitement et à un formatage utilisant la norme XML aux travers des schémas. Après avoir défini la structure du schéma retenu pour notre application, son implémentation systématique au sein de l'ensemble des classes est en cours de réalisation. Le schéma XML nous semble ainsi une solution pertinente, il reste maintenant à valider ce concept dans le cas d'utilisation concrête : en particulier pour des gros fichiers de maillage ou de sauvegarde, au regard des temps de traitement supplémentaires induits.

Références

[1] National Center for Supercomputing Applications. Information, support, and software from the hierarchical data format (hdf) group of ncsa. http ://hdf.ncsa.uiuc.edu/, 2005.

[2] J. Michopoulos, P. Mast, T. Chwastyk, L. Gause, and R. Badaliance. "femml for data exchange between fea codes", ansys users' group conference, university of maryland college park md, october 2 2001. http ://www.istos.org/femML/documentation.htm, 2001.

[3] The World Wide Web consortium. Extensible markup language (xml) (third edition) w3c recommendation. http ://www.w3.org/TR/REC-xml/, 2004.

[4] XSIL. Extensible scientific interchange language. http ://www.cacr.caltech.edu/SDA/xsil/.

[5] Giovanni Aloisio, Massimo Cafaro, and Roy Williams. The digital puglia project : An active digital library of remote sensing data. *Proceedings of the 7th International Conference on High Performance Computing and Networking Europe. Amsterdam, The Netherlands, (Springer Lecture Notes in Computer Science)*, 1593 :563, 1999.

[6] XDMF. extensible data model and format. http ://www.arl.hpc.mil/ice/, 2005.

[7] Pinecoast Software. Feviaduct. http ://www.pinecoast.com/feviaduct.htm, 2005.

[8] XMDS. extensible multi-dimensional simulator. http ://www.xmds.org/, 2005.

[9] The Apache Software Foundation. Xerces. http ://xml.apache.org/, 2005.

[10] Morphon Technologies. Morphon xml-editor 3.1.4. http ://www.morphon.com/, 2003.

[11] Michael Brundage Kevin Williams, Patrick Dengler, Jeff Gabriel, Andy Hoskinson, Michael Kay, Thomas Maxwell, Marcelo Ochoa, Johnny Papa, and Mohan Vanmane. *XML et les bases de données*. Edition Eyrolles, 2001, ISBN 2-212-009282-2.

[12] Eric van der Vlist. *XML Schema*. O'Reilly, 2002, ISBN 0-596-00252-1.

[13] Alain Michard. *XML langage et applications*. Ed. Eyrolles, 2000, ISBN 2-212-09206-7.

[14] Gérard Rio. Code éléments finis herezh++. http ://web.univ-ubs.fr/lg2m/, 2005.

[15] Erik T. Ray and Jason McIntosh. *Perl and XML*. O'Reilly, 2002, ISBN 0-596-00205-X.