XMLlab : un outil générique de simulation basé sur XML et Scilab

Stéphane Mottelet[1], André Pauss[2]

Résumé

Nous présentons un environnement de génération automatique de simulations entièrement basé sur les technologies XML. Le langage de description proposé permet de décrire des objets mathématiques tels que des systèmes d'équations différentielles, des systèmes d'équations non-linéaires, des équations aux dérivées partielles en dimension 2, ou bien de simples courbes et surfaces. Il permet aussi de décrire les paramètres dont dépendent ces objets. Ce langage est indépendant du logiciel et permet donc de garantir la pérennité du travail des auteurs ainsi que leur mutualisation et leur réutilisation. Nous décrivons aussi l'architecture d'une «chaîne de compilation» permettant de transformer ces fichiers XML sous forme de scripts et de les faire fonctionner dans le logiciel Scilab.

Mots-clés : simulation, interopérabilité

Abstract

We present an XML-based simulation authoring environment. The proposed description language allows to describe mathematical objects such as systems of ordinary differential equations, systems of non-linear equations, partial differential equations in two dimensions, or simple curves and surfaces. It also allows to describe the parameters on which these objects depend. This language is independent of the software and allows to ensure the perennity of author's work, as well as collaborative work and content reuse. We also describe the architecture of a «compilation chain» allowing to transform the XML files into Scilab scripts.

[1]Laboratoire de Mathématiques Appliquées de Compiègne, Département de Génie Informatique, Université de Technologie de Compiègne, BP 20529, 60205 COMPIEGNE CEDEX, FRANCE; stephane.mottelet@utc.fr

[2]UMR Génie des Procédés Industriels, Département de Génie Chimique, Université de Technologie de Compiègne, BP 20529, 60205 COMPIEGNE CEDEX, FRANCE; andre.pauss@utc.fr

Keywords : simulation, interoperability

1 Introduction

Le besoin de faire appel à des simulations répond en général à une problématique s'énonçant clairement : l'utilisateur a des équations modélisant un phénomène. Il voudrait pouvoir les résoudre, et si possible pouvoir faire varier certains paramètres pour voir leur influence sur les résultats de la simulation, puis sauvegarder les résultats (et les valeurs de paramètres associés) dans un format qu'il pourra relire par exemple dans un tableur. Les vertus pédagogiques de ce genre de démarche, lorsque l'on dispose des outils adéquats, n'est pas à démontrer.

Autant il est rapide de faire rentrer ses équations dans un logiciel, pourvu qu'il soit adapté à la discipline dont est issu le phénomène à simuler, dès qu'il s'agit de développer une interface utilisateur (boutons, menus, etc.) les temps de développement explosent, car il s'agit malheureusement de la phase la plus longue. L'auteur passe alors le plus gros de son temps à peaufiner l'interface, alors qu'il aurait pu le consacrer à travailler sur une autre simulation. De plus, plus le travail aura été peaufiné pour un cas particulier de phénomène, moins il sera réutilisable dans un contexte voisin. Le World Wide Web fourmille de multitudes d'applets JAVA, souvent de bonne qualité, mais parfois difficilement adaptables dans le contexte d'un enseignement particulier, car les modifier (quand l'auteur distribue le code source) nécessite de disposer de compétences dans un langage informatique de bas niveau (en général JAVA), ou bien de connaître un langage de scripts utilisé par Matlab ou Scilab ([3, 5, 8], quand le programme est destiné à être interprété par l'un de ces logiciels.

Nous sommes ici dans le domaine de l'artisanat, et non pas dans une logique industrielle. Le travail de l'auteur n'est en général pas réutilisable, sa pérennité n'est pas assurée, car il s'appuie un logiciel utilisant un format propriétaire, et surtout il ne s'échange qu'avec les au-

teurs utilisant le même logiciel (et la même version).

Autant cette prise de conscience a déjà eu lieu dans le domaine purement documentaire avec l'explosion des applications du langage de balisage XML ([2]), le domaine de la simulation reste encore largement sourd à ces préoccupations. On peut tout au plus signaler quelques initiatives. En biologie et en chimie un consortium d'universitaires et d'auteurs d'une vingtaine de logiciels de simulation sont à l'origine de `sbml`, un format d'échange de modèles biologiques et cinétiques utilisant XML ([7, 6]).

On peut aussi citer `xmds`, un outil lui aussi basé sur XML permettant de générer du code Scilab, Matlab ou C++ pour simuler des systèmes déterministes ou stochastiques, mais sans interface utilisateur ([4]). Par rapport à l'approche que nous allons présenter plus loin, on peut reprocher à cet outil de ne pas pousser assez loin la structuration, en ce sens qu'il n'utilise pas à fond les possibilités de structuration du langage XML.

Dans le cadre du projet XMLlab, il a été choisi de démontrer les intérêts d'une approche où le fond et la forme sont bien différenciés et relèvent chacun d'acteurs différents :

Le fond Il s'agit des équations du phénomène, de leur description, des paramètres constitutifs et de leur organisation thématique. Leur écriture est du ressort de l'auteur.

La forme Il s'agit de l'interface utilisateur, c'est à dire des divers «widgets» et autres menus qui viennent renforcer la convivialité du programme final, ainsi que des outils de visualisation. L'écriture du code faisant fonctionner cette partie du programme est du ressort soit d'une personne expérimentée, soit du ressort d'un outil capable de générer ce code automatiquement à partir de la description qui a été faite par l'auteur. C'est cette dernière option que nous avons retenue.

N'oublions pas le problème du choix des méthodes numériques. L'auteur n'est pas nécessairement capable de faire ce choix lui même, c'est pourquoi ce choix doit être fait automatiquement sachant quelles méthodes sont les mieux adaptées à la résolution d'un type particulier d'équation.

Pour que cette réflexion ne conduise pas à une spécification stérile, nous avons choisi dans les toutes premières phases du projet un logiciel cible particulier. Il s'agit de Scilab, logiciel *open source* développé par l'INRIA. Le but était non seulement de proposer une structuration des documents XML décrivant les simulations, mais aussi de développer une chaîne de compilation complète, permettant de transformer les documents XML en des fichiers exécutables par le logiciel cible. Le choix de Scilab est de plus motivé par le fait que ce logiciel permet d'utiliser le langage de scripts Tcl/Tk ([10, 9]) pour générer les interfaces utilisateur (nous y reviendrons plus bas).

2 Un aperçu de la structure d'une simulation au format XMLlab

Une simulation peut être divisée en un certain nombre d'éléments conceptuels : des paramètres, des modèles mathématiques d'objets dépendant du temps ou non et pour terminer un élément d'affichage des résultats de la simulation :

2.1 Paramètres

Il s'agit des paramètres constitutifs du phénomène à simuler, et le but est qu'au final l'utilisateur de la simulation puisse faire varier ces paramètres à l'aide de l'interface qui sera générée par la chaîne de compilation. D'autre part, il doit être possible de spécifier simplement si la valeur du paramètre est vue dans l'interface, mais non modifiable par l'utilisateur. De même il doit pouvoir exister des paramètres uniquement «pour usage interne» à la simulation.

Scalaires et matrices Dans une simulation XMLlables paramètres peuvent être des scalaires ou des matrices. On peut pour les scalaires préciser leurs valeurs minimum et maximum, le type de "widget" à utiliser (une glissière, ou bien un simple champ où l'utilisateur doit rentrer la valeur). Chaque paramètre doit avoir un nom symbolique qui pourra être réutilisé dans la description du modèle mathématique, et une valeur par défaut, qui sera utilisée lors de la première utilisation de la simulation.

Groupes de paramètres Ces paramètres peuvent être groupés dans des sections, par exemple, pour une équation différentielle, on peut vouloir différencier les paramètres physiques des paramètres de résolution (temps final, nombre de pas de temps, etc.). L'intérêt de cette structuration est qu'elle sert ensuite servir à structurer l'interface de la simulation.

Bases de données On donne la possibilité de stocker plusieurs instances d'un groupe de paramètres. Cela permet par exemple, en chimie, de constituer une base de données constitué de différents acides

et bases, en stockant leurs paramètres (constantes d'acidité, charge, etc.). L'utilisation qui peut en être faite est de proposer dans l'interface un menu permettant de choisir un groupe de paramètres donné.

2.2 Modèles mathématiques

On s'intéresse ici aux équations du phénomène à simuler. Les objets sont ici de plusieurs niveaux.

Domaines Les plus simples décrivent des intervalles de $\mathbb{R}$ ou bien des domaines de $\mathbb{R}^2$. S'il s'agit donc de simples intervalles du type $[a,b]$, où bien sûr les bornes peuvent dépendre de paramètres décrits dans la section précédente, ou alors de domaines bidimensionnels. Ces derniers peuvent être des rectangles définis à partir d'un produit cartésien d'intervalles, ou bien des domaines généraux définis à partir de la forme de leur frontière par des courbes paramétriques (nous y reviendrons plus bas). On peut préciser la façon dont ces domaines doivent être discrétisés le cas échéant (nombre de points de discrétisation, linéaire ou logarithmique).

Courbes et surfaces On peut ensuite définir des courbes, par exemple

$$y = f(x), \quad x \in [a,b].$$

Ici on voit que cette définition réutilise un intervalle. En pratique, on peut ainsi définir plusieurs courbes faisant référence à un même intervalle.
On peut aussi définir des courbes paramétriques du type

$$\left\{ \begin{array}{rcl} x & = & f(t), \\ y & = & g(t), \\ t & \in & [a,b]. \end{array} \right.$$

Les surfaces peuvent aussi être de type paramétrique ou non paramétrique, c'est à dire soit de la forme

$$z = f(x,y), \quad (x,y) \in \mathcal{D},$$

où $\mathcal{D}$ est un domaine de $\mathbb{R}^2$, soit

$$\left\{ \begin{array}{rcl} x & = & f(u,v), \\ y & = & g(u,v), \\ z & = & h(u,v) \\ u & \in & \mathcal{D}. \end{array} \right.$$

Précisons que les paramètres définis dans la section précédente peuvent intervenir à tous niveaux, que ce soit dans la définition des domaines ou dans les équations elles-mêmes.

Equations différentielles On peut décrire des systèmes d'équations différentielles. Par exemple

$$\left\{ \begin{array}{rcl} \frac{d}{dt}x(t) & = & f(x,y,t), \\ \frac{d}{dt}y(t) & = & g(x,y,t), \\ t & \in & [a,b], \end{array} \right.$$

avec des conditions initiales $x(a) = x_a$ et $y(a) = y_a$ données. On peut dans XMLlab garder la description naturelle, sans avoir à reformuler chaque inconnue $x(t)$ ou $y(t)$ comme élément d'un vecteur $X(t)$ avec $x(t) = X_1(t)$ et $y(t) = X_2(t)$. Le modèle de description choisi consiste à décomposer le système d'équations différentielles en

- Un intervalle temporel (ici $[a,b]$)
- Une liste «d'états». Pour chaque état (ici x ou y), on définit quelle est sa dérivée par rapport au temps, et quelle est sa valeur initiale.
- Une liste de «sorties». Il s'agit d'observations que l'on peut construire à l'aide des états, par exemple $z(t) = x(t) + y(t)$.

Equations non linéaires Il s'agit soit de systèmes généraux d'équations non linéaires, du type

$$\left\{ \begin{array}{rcl} f(x,y,z,t,\cdots) & = & 0, \\ h(x,y,z,t,\cdots) & = & 0, \\ & \cdots & \end{array} \right.$$

soit une équation non-linéaire dépendant d'un paramètre appartenant à un intervalle, de la forme

$$\left\{ \begin{array}{rcl} f(x,y) & = & 0 \\ x & \in & [a,b], \\ \cdots & & \end{array} \right.$$

permettant de prendre en compte certaines courbes définies par une équation implicite du type $f(x,y) = 0$. Ce genre de modèle est utilisé en particulier pour modéliser des courbes de neutralisation acide/base en chimie.

Equations aux dérivées partielles Il est possible de décrire des équations aux dérivées partielles de type diffusion, typiquement,

$$\left\{ \begin{array}{c} -\operatorname{div}\left(P \operatorname{grad} u\right)(x) + c(x)u(x) = f(x), \ x \in \Omega, \\ +\text{conditions aux limites} \end{array} \right.$$

Le domaine Ω est décrit à partir de sa frontière, à partir de courbes paramétriques (définies plus haut). Nous donnerons quelques détails sur la résolution numériques dans la section suivante. Ici encore, les paramètres doivent pouvoir intervenir à tous les niveaux, que ce soit dans la définition du domaine Ω

ou dans les paramètres physiques (matrice de diffusion P, terme de source $f(x)$, coefficient proportionnel $c(x)$.

2.3 Affichage des résultats

Il s'agit d'une description hiérarchique classique, sous forme de fenêtres et de systèmes d'axes, où l'utilisateur doit juste préciser ce qu'il veut afficher en faisant référence à des objets définis dans la section "Modèles mathématiques". Nous donnerons ici quelques précisions sur la façon dont la structuration se traduit dans la chaîne de compilation sous sa forme actuelle.

Fenêtres Une fenêtre contient des systèmes d'axes. L'utilisateur doit juste spécifier comment ils doivent être agencés s'il y en a plus d'un (la fenêtre est alors subdivisée dans sa hauteur et/ou sa largeur).

Système d'axes L'utilisateur doit juste spécifier s'il s'agit d'un système d'axe plan ou dans l'espace. Ensuite chaque système d'axe contient des références à ce qu'il faut y représenter. On peut ainsi faire référence à un état d'une équation différentielle par l'intermédiaire de son identificateur, où à la solution d'une équation aux dérivées partielles.

Objets à représenter La structure choisie permet de simplifier énormément le nombre de structures différentes. Par exemple, une surface peut être référencées dans un système d'axes à deux ou trois dimensions. Dans un système à trois dimensions on la représente dans l'espace, alors qu'en deux dimensions, on prend par exemple la convention qu'une représentation en pseudo-couleurs est plus adaptée. Dans les deux cas, la référence à la surface n'a pas changé, c'est uniquement le contexte «parent» qui a changé.

3 Un exemple de simulation

On donne sur la fichier 1 l'ébauche d'un document simulation. Nous allons montrer comment construire ce fichier pour décrire une petite simulation.

On considère le pendule représenté sur la figure 2. Nous supposerons que la tige reliant un poids de masse M à l'axe de rotation est de masse négligeable devant M. Nous choisissons de mesurer la déviation du pendule de la position verticale d'équilibre stable par l'angle $\theta(t)$ mesuré positivement comme défini sur la figure 2. Si l'on applique les relations de la dynamique pour les solides en rotation autour d'un axe, on obtient

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE simulation
SYSTEM "simulation.dtd">
<simulation>
  <header>
   ...
  </header>
  <notes>
     ...
</notes>
  <parameters>
     ...
  </parameters>
  <compute>
     ...
  </compute>
  <display>
     ...
  </display>
</simulation>
```

FIG. 1: Le squelette d'une description de simulation montrant les éléments de plus haut niveau.

l'équation différentielle suivante :

$$\begin{cases} \ddot{\theta}(t) &= -\dfrac{g}{L}\sin\theta(t), \quad t \in [0,T] \\ \theta(0) &= \theta_0, \\ \dot{\theta}(0) &= 0. \end{cases}$$

La valeur de θ_0 donne la déviation initiale du pendule, et on a considéré que la vitesse angulaire initiale est nulle. Si θ_0 est faible, on peut approcher $\theta(t)$ par

$$\phi(t) = \theta_0 \cos\left(\sqrt{\frac{g}{L}}t\right).$$

Nous allons montrer comment traduire tout cela en éléments XML, en procédant dans l'ordre du fichier représenté sur la figure 1.

3.1 Paramètres, éléments `parameters`

Nous devons prendre en compte les paramètres physiques g, L, θ_0, ainsi que le temps final T. Le fragment de code XML suivant permet de décrire ces paramètres :

```
<parameters>
  <title>Paramètres du pendule</title>
  <scalar label="L" unit="m">
    <name>Longueur du pendule</name>
```

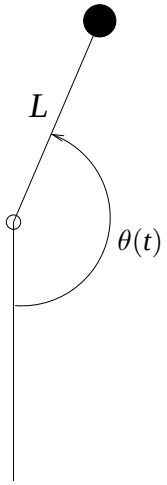

FIG. 2: Le pendule

```
    <value>1</value>
  </scalar>
  <scalar label="g0" unit="ms^-2">
    <name>Accélération de la
          pesanteur</name>
    <value>9.81</value>
  </scalar>
  <scalar label="theta_0" max="3.141"
          min="0.01" unit="rad"
          increment="0.001"
          widget="slider">
    <name>Angle initial</name>
    <value>0.1</value>
  </scalar>
</parameters>
<parameters>
  <title>Paramètres de résolution
  </title>
  <scalar label="T" unit="s">
    <name>Temps final</name>
    <value>2</value>
  </scalar>
</parameters>
```

Chaque paire de balises `parameter` permet de grouper des paramètres. La balise `scalar` permet de spécifier un paramètre scalaire, contenant son nom complet dans une balise `name` et sa valeur initiale `value`. Un scalaire possèdet un attribut obligatoire `label` qui représente son nom symbolique, qui pourra être utilisé dans les balises `value` d'autres éléments. L'attribut `widget` permet de signifier que l'on veut disposer d'une glissière).

3.2 Modèles mathématiques, élément `compute`

Voici les fragment correspondant à la description de l'intervalle $[0, T]$,

```
<defdomain1d label="t" unit="s">
  <name>Temps</name>
  <interval>
    <initialvalue>0</initialvalue>
    <finalvalue>T</finalvalue>
  </interval>
</defdomain1d>
```

ainsi qu'à la description de l'équation différentielle :

```
<ode label="pendule">
  <refdomain1d ref="t"/>
  <states>
    <state label="theta"
           unit="rad">
      <name>Solution réelle</name>
      <derivative>theta_point
      </derivative>
      <initialcond>theta_0
      </initialcond>
    </state>
    <state label="theta_point"
           unit="rad/s">
      <name>dérivée de l'angle</name>
      <derivative>-g0/L*sin(theta)
      </derivative>
      <initialcond>0</initialcond>
    </state>
  </states>
  <outputs>
    <output label="theta_lin">
      <name>Solution linearisee</name>
      <value>theta_0*cos(sqrt(g0/L)*t)
      </value>
    </output>
  </outputs>
</ode>
```

L'élément `ode` pour (ordinary differential equation) est constitué d'un élément vide `refdomain1d` faisant référence à l'intervalle $[0, T]$, et définissant ansi le nom symbolique de la variable d'intégration, d'un element `states` contenant la description de chaque état (θ et $\dot{\theta}$) dans un élément `state`. Chaque balise `state` contient le nom de l'état, sa dérivée par rapport

à la variable temps `derivative` ainsi que sa valeur à l'instant initial `initialcond`. Le dernier élément dans `ode` est une liste de sorties `outputs`. Ici la seule sortie dépend explicitement du temps mais ne dépend pas des états. Chaque état ou sortie dispose d'un attribut obligatoire `label` auquel sera fait référence dans la section d'affichage.

3.3 Affichage des résultats, élément `display`

Ici on cherche juste à superposer deux courbes dans un même système d'axes.

```
<display>
  <window>
    <title>Comparaison des deux
solutions</title>
    <axis2d>
      <drawcurve2d ref="theta"/>
      <drawcurve2d ref="thetalin"/>
    </axis2d>
  </window>
<display>
```

L'élément `display` ne contient qu'un élément `window`, ne contenant lui-même qu'un système d'axes à deux dimensions. Les deux éléments `drawcurve2d` permettent d'indiquer que l'on veut faire apparaître θ et sa version linéarisée superposés.

3.4 Remarques

Les différentes possibilités de structuration sont contraintes par une DTD (Document Type Définition), qui peut servir à la validation *a posteriori* d'un fichier simulation, ou bien qui peut être utilisée pour contraindre les possibilités d'édition d'un fichier à l'aide d'un éditeur XML. La figure 3 montre la vue que l'on peut avoir du fichier XML correspondant à la simulation construite précédemment.

Dans tous les éléments cités précédemment, il en est certains qui ont une importance particulière. Les balises `name` et `title` peuvent apparaître plusieurs fois, avec un attribut `lang` différent (pour l'instant `french` ou `english`. Le but est de pouvoir générer à partir du même fichier deux versions du programme «compilé», la langue à retenir étant alors spécifiée comme une option de compilation.

L'élément `header` contient des méta-données telles que le nom de l'auteur et des mots-clés. L'élément `notes`, pouvant lui aussi apparaître plusieurs fois avec

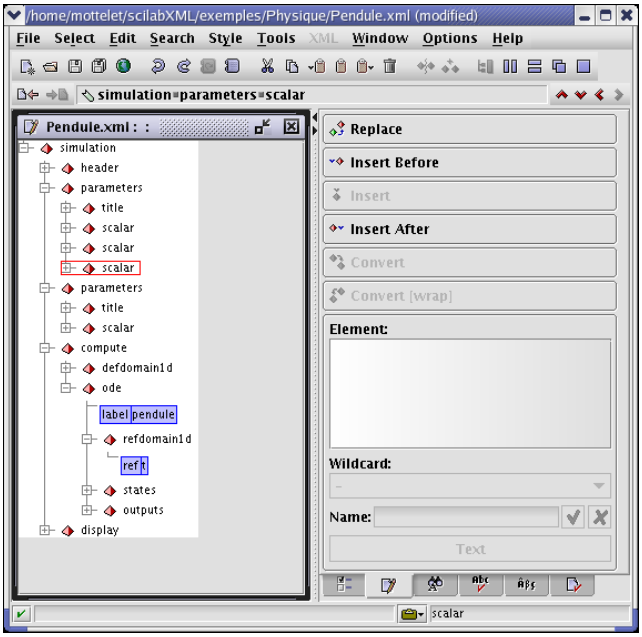

FIG. 3: Le fichier XML décrivant la simulation du pendule, vu dans l'éditeur XXE, développé par la société PIXWARE, `http ://www.xmlmind.com/xmleditor1`

un attribut `lang` différent permet d'écrire quelques paragraphes de texte destinés à décrire la simulation et/ou à donner des conseils d'utilisation.

4 La chaîne de compilation

4.1 Quelques détails

Elle est entièrement basée sur une technologie XML : il s'agit de transformations spécifiées dans des feuilles de style XSL (eXtensible Stylesheet Language) que l'on fait subir au fichier simulation en utilisant un «processeur XSL». Cette technologie est bien connue pour pouvoir afficher du contenu HTML dynamique sur le World Wide Web, mais il s'avère qu'elle est aussi bien adaptée à la génération automatique de scripts. En particulier nous nous intéressons ici à un langage de scripts tel que celui utilisé par Scilab ou Matlab, ou bien au langage utilisé par les scripts Tcl/Tk, permettant de décrire des interfaces utilisateur.

Les différentes phases de la compilation sont résumées sur le diagramme représenté figure 4. En bas du diagramme, Le fichier `pendule.sce` est le fichier Scilab contenant tout ce qui concerne le calcul de la simulation, et l'affichage des résultats. Le fichier `pendule.tk` est le fichier contenant le code Tcl/TK de l'interface. Sur le diagramme on voit que la transformation n'est pas directe et nécessite une étape inter-

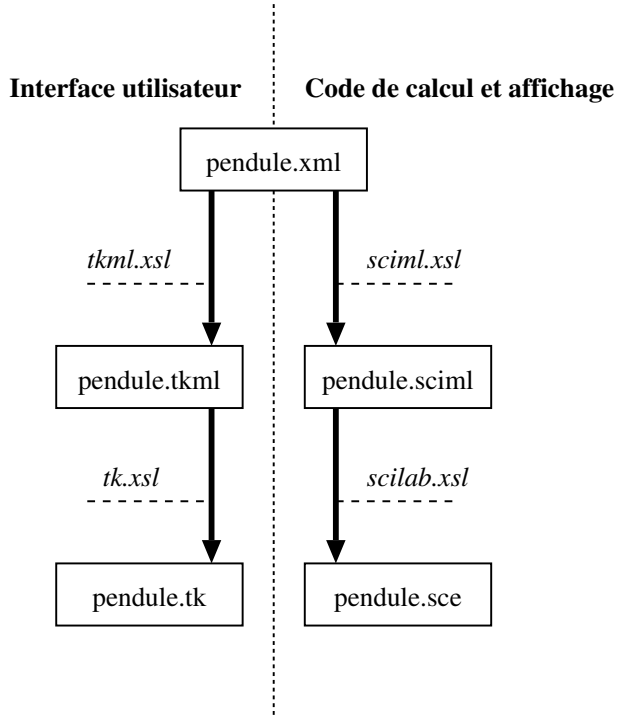

FIG. 4: Diagramme de la chaîne de compilation. Les flèches représentent les transformations XSL, et les noms en italiques les feuilles de style associées

médiaire, cela nécessite quelques explications.

De manière à ce que les feuilles xsl soient faciles à maintenir, et surtout pour que les choix techniques (Scilab et Tcl/Tk) ne soient pas définitivement figés, nous avons dû utiliser deux dialectes «pivots» :

tkml En ce qui concerne la transformation générant l'interface (côté gauche du diagramme), le fichier intermédiaire `pendule.tkml` est un fichier XML contenant une description de l'interface indépendante du langage final. Ce fichier contient la description des différents «widgets» (boutons, etc.) les uns par rapport aux autres. Ensuite, ce fichier pivot est effectivement traduit en Tcl/TK à l'aide d'une dernière transformation XSL.

sciml Pour la partie engendrant le code Scilab, il en est de même : on utilise un fichier XML pivot `pendule.sciml`, qui n'est traduit en Scilab qu'à la fin, à l'aide d'une transformation XSLT supplémentaire.

On peut ainsi envisager deux modes de distribution des simulation : les deux fichiers Tcl/Tk et Scilab peuvent être utilisés sans faire appel à la chaî ne de compilation. Cependant le mode le plus profitable à la communauté est de mettre à disposition le source XML (le premier mode permettant à l'auteur de protéger son travail).

L'ensemble de la chaîne de compilation accompagnée de Scilab, en faisant exception de l'éditeur, n'utilise que des logiciels open source (`xsltproc` du projet Gnome, scripts Tcl), et fonctionne sur toute plateforme (Windows, Mac OS X, Unix).

4.2 Commentaires sur l'exemple du pendule

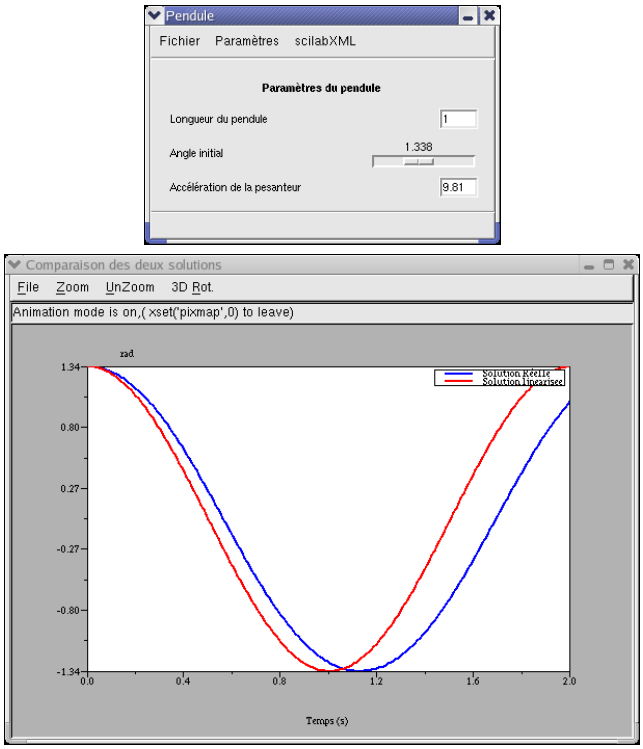

FIG. 5: L'interface et la fenêtre d'affichage des résultats pour la simulation du pendule

Nous faisons ici quelques commentaires sur la figure 5.

Interface utilisateur La fenêtre de l'interface est composée d'un espace central où apparaissent les «widget» permettant de changer les paramètres.
Le menu `Paramètres` contient deux items nommés «Paramètres du pendule» «Paramètres de résolution» correspondant aux deux groupes de paramètres spécifiés dans le fichier XML. Il suffit de sélectionner un item donné pour faire apparaître les paramètres correspondants. Pour cet exemple particulier.
Le menu `Fichier` contient deux items intéressants : «Sauver une session» et «Charger une session». Ils permettent de sauver les valeurs des paramètres dans un fichier au format texte, et de les relire ultérieurement. Cela permet de reprendre un travail «d'exploration» en cours, le script Scilab démarrant

toujours avec les valeurs initiales des paramètres.
Le menu `XMLlab` donne les coordonnées des auteurs de ScilabXML (item «A propos de XMLlab»), et des informations sur la simulation, récupérées dans l'élément `header` (nom de l'auteur, date) et permet de visualiser les notes de l'auteur décrivant la simulation (item «A propos de cette simulation»).

Fenêtre graphique La légende des deux courbes est composée des éléments `name`, de l'état `theta` ainsi que de la sortie `theta_lin`. En abscisse apparaît le nom de la variable de l'intervalle temporel `t`, et en ordonnée l'unité (attribut `unit` de l'élément `<state label="theta"`).
Il s'agit d'une fenêtre graphique Scilab, on a donc accès aux menus habituels, permettant de sauver la figure, par exemple au format PostScript, ou bien de l'imprimer.

Notons que l'utilisateur a toujours la possibilité d'avoir accès à toutes les variables de la simulation à partir de la ligne de commande de Scilab (paramètres et résultats de simulation), qui reste bien sûr disponible durant la simulation.

Remarque sur les performances Pour cet exemple particulier (système d'équations différentielles), le calcul prend un temps négligeable par rapport au temps passé à dessiner les courbes, et l'utilisateur peut ainsi voir en temps réel l'influence de l'angle initial sur la désynchronisation des deux courbes. Sur des exemples plus imposants (résolution d'équation aux dérivées partielles), le temps de réponse peut être supérieur, mais la réactivité reste toujours très bonne, même sur une machine modeste (Pentium 1Ghz). Pour cela Scilab est très appréciable, car les fonctions de base faisant appel à son noyau sont bien optimisées (résolution d'équations linéaires et non linéaires, équations différentielle, algèbre de matrice creuses, vectorisation d'opérations élémentaires dur les tableaux, etc.). De plus, il démarre quasi instantanément (à comparer avec les dernières versions de Matlab, très lourdes, car utilisant JAVA pour l'interface).

5 Conclusion et perspectives

Architecture Nous pensons avoir fait les bons choix de structuration du logiciel, car l'ajout d'un nouveau type d'équation est très rapide à mettre en oeuvre. A titre d'exemple, l'élément `stationary-pde`, permettant de décrire une équation aux dérivées partielles elliptique, a été développé en deux jours (Scilab disposant d'une «<PDE toolbox» sur laquelle nous nous appuyons).

Développements futurs Les priorités concernent en particulier
- la simulation de systèmes à temps discret,
- la simulation de systèmes stochastiques,
- des possibilités de visualisations sous formes d'animations.

En ce qui concerne l'édition des fichiers XML, nous prévoyons le développement de «Cascading Stylesheets» permettant de rendre encore plus conviviale l'édition des fichiers XML dans l'éditeur XXE (voir [1]). Il est aussi prévu de spécifier la structuration des simulations à l'aide d'un Schéma XML ([11]) à la place d'une DTD, pour mieux contrôler les différents types de données contenus dans les éléments et les attributs.

Champs d'application Des simulations ont déjà été écrites dans le domaine de la biologie (cinétiques enzymatique et microbienne), de la physique (pendule, fentes de Young, équation de Poisson, etc.), de la chimie (équilibres acide/base, cinétiques chimiques), du génie chimique (réacteurs idéaux). Nous souhaitons étendre les champs disciplinaires dans l'avenir. Un exemple d'application est présenté par ailleurs dans ce congrès (équilibres acide/base dans l'enseignement de chimie du premier cycle d'ingénieur).

Licence Nous prévoyons de distribuer XMLlab sous forme d'une Toolbox pour Scilab avec une licence de type Open Source. Nous espérons ainsi de nombreuses contributions et une stimulation forte de la communauté pour enrichir son champ disciplinaire.

Remerciements

Ce travail a été financé par le Pôle régional de Recherche de Picardie «Evaluation des Nouvelles Technologies dans l'Enseignement». Des enseignants de l'Institut Universitaire de Formation des Maîtres d'Amiens, de l'Université de Picardie Jules Vernes d'Amiens et de l'Université de Technologie de Compiègne contribuent très efficacement à la transdisciplinarité de ce projet.

References

[1] B. Bos, H. Wium Lie, C. Lilley, and I. (Eds.) Jacobs. ascading stylesheets, level 2, css2 specification. *Available via the World Wide Web*

at http ://www.w3.org/TR/1998/REC-CSS2-19980512, 1998.

[2] T. Bray, J. Paoli, and C.M et al. Sperberg-McQueen. Extensible markup language (xml) 1.0. *Available via the World Wide Web at http ://www.w3.org/TR/2004/REC-xml-20040204*, 2004.

[3] J.-P. Chancelier, F. Delebecque, C. Gomez, R. Goursat, M.and Nikoukah, and S. Steer. *Introduction à Scilab*. Springer, Paris, 2001.

[4] G. Collecutt, P. Drummond, J. Hope, and P. Cochrane. Extensible multi-dimensional simulator (xmds). *Available via the World Wide Web at http ://www.physics.uq.edu.au/xmds/index.html*, 2001.

[5] C. (Ed.) Gomez. *Engineering and scientific computing with scilab*. Birkauser, Boston, 1999.

[6] M. Hucka and A. Finney. Systems biology markup language : Level 2 and beyond. *Biochem. Soc. Trans.*, 31 :1472–1473, 2003.

[7] M. Hucka, A. Finney, H.M Sauro, H. Bolouri, J.C. Doyle, and al. The systems biology markup language (sbml) : a medium for representation and exchange of biochemical netwok models. *Bioinformatics*, 19 :524–531, 2003.

[8] P.S. Motta Pires and D.A. Rogers. Free/opensource software : an alternative of engineering students. *Procceding of the 32nd ASEE/IEEE Frontiers in Education Conference, November 6 - 9*, 2002.

[9] J.K. Ousterhout. *Tcl and the Tk toolkit*. Addison-Wesley, 1994.

[10] J.K. Ousterhout. Scripting : Higher-level programming for the 21st century. *IEEE Computer magazine*, March :23–30, 1998.

[11] H.S Thompson, D. Beech, M. Maloney, and N. Mendelsohn. Xml schema part 1 : Structures. *Available via the World Wide Web at http ://www.w3.org/TR/xmlschema-1*, 2000.