algoML : un format XML pour l'apprentissage de l'algorithmique et la programmation

Stéphane Rivière, Karine Zampieri, Béatrice Amerein-Stolner

algoML : un format XML pour l'apprentissage de l'algorithmique et la programmation

Stéphane Rivière[1], Karine Zampieri[1], Béatrice Amerein-Soltner[2]

stephane.riviere@uha.fr, karine.zampieri@uha.fr, beatrice.amerein-soltner@uha.fr

[1]Université de Haute-Alsace – Équipe Mage, LMIA
4, rue des Frères Lumière 68100 Mulhouse, France
[2]Université de Haute-Alsace – Chargée de Mission Innovations Pédagogiques
4, rue des Frères Lumière 68100 Mulhouse, France

Résumé. Nous présentons algoML, un format XML destiné à représenter des algorithmes et des programmes de façon indépendante d'une notation de pseudo-langage ou d'un langage de programmation particulier, format que nous avons développé dans le cadre de ressources en ligne dans le domaine de l'algorithmique pour l'UNT (Université Numérique Thématique) Unisciel.

Nous avons conçu ce format pour qu'il puisse être utilisé dans de nombreux contextes différents, que ce soit au niveau des langages utilisés par les applications (pseudo-langages de haut niveau, langages de programmation existants) ou au niveau des applications elles-mêmes (cours d'algorithmique avec sortie dans différents langages, environnement visuel de programmation avec un langage propre, …). Nous avons aussi conçu ce format pour qu'il puisse être traité facilement avec les outils XML standards, notamment le langage de transformation XSLT, sans avoir de connaissance particulière en compilation.

Mots-clés: informatique, algorithmique, programmation, enseignement, XML

Introduction

Un algorithme est une méthode qui en enchaînant l'utilisation d'un certain nombre d'outils de calculs de base (des variables pour stocker des valeurs, une instruction conditionnelle « si alors sinon » qui permet d'effectuer telles ou telles instructions en fonction du résultat d'un test, …), permet de calculer les résultats voulus à partir de données de départ et éventuellement de les afficher. Un programme est la mise en œuvre d'un algorithme sur ordinateur grâce à l'utilisation d'un langage de programmation particulier, ce qui permet ensuite d'exécuter le programme. L'algorithmique et la programmation, c'est-à-dire l'étude de la conception et de l'écriture d'algorithmes et de programmes, sont donc à la base de l'informatique.

Bien que les outils utilisés en algorithmique soient bien définis, il existe malheureusement pratiquement autant de notations différentes que d'auteurs (que

Baron, G.-L., Bruillard, E., & Komis, V. (2011). Sciences et technologies de l'information et de la communication (STIC) en milieu éducatif : Analyse de pratiques et enjeux didactiques, Posters du quatrième colloque international DIDAPRO 4 - Dida&STIC, 24-26 octobre 2011, Université de Patras. Athènes : New Technologies Editions, 2011. ISBN 978-960-6759-76-5

ce soit en français ou en anglais) pour décrire un algorithme. De plus, pour programmer un algorithme sur ordinateur, il faut utiliser un langage de programmation, chaque langage ayant une syntaxe différente et certains langages donnant même lieu parfois à des débats houleux sur l'indentation (c.-à-d. la mise en page) à adopter.

Lors de la réalisation d'un projet Unisciel (Université des Sciences en Ligne) de mise en ligne d'un cours sur les bases fondamentales de d'algorithmique et de la programmation (niveau L1), s'est alors posé à nous le problème du format d'écriture des algorithmes et des programmes : comment les écrire dans un format unique de façon indépendante d'un langage particulier et pouvoir ensuite à partir de ce format offrir une publication sous différentes formes (papier, HTML) et une génération automatique dans différents langages particuliers.

Nous nous sommes tournés naturellement vers un format XML, c'est-à-dire un format texte structuré grâce à des balises, car un tel format est facile à écrire et surtout bénéficie de nombreux outils permettant de l'exploiter. Malheureusement les formats XML existants créés pour représenter des programmes s'intéressent uniquement aux aspect techniques de représentation d'un programme.

Ainsi divers auteurs, (Badros, 2000), (Mamas & Kontogiannis, 2000), (Boshernitsan & Graham, 2000), ont développé des formats représentant la décomposition grammaticale générale d'un programme suivant la grammaire d'un langage particulier. En utilisant un arsenal d'analyseurs syntaxiques et de compilateurs, ils produisent automatiquement cette décomposition XML à partir d'un programme source écrit dans ce langage particulier. Un format plus récent (Collard, 05) se contente de rajouter des balises XML dans du code source Java. Dans tous ces cas le but est juste d'avoir la représentation grammaticale du programme sous une forme XML, forme plus facile à utiliser grâce aux outils XML existants. Aucun de ces auteurs ne s'intéresse aux aspects algorithmiques et encore moins aux aspects pédagogiques.

Le seul format s'intéressant à l'algorithmique que nous avons trouvé est Xalgo (Seidel, 2004), un format décrit dans un mémoire universitaire en allemand (Diplomarbeit) et apparemment resté sans suite. Ce format a été créé pour faire spécifiquement des sorties en C++ et Java, deux langages qui partagent beaucoup de similarités, notamment au niveau de la syntaxe. Ces deux langages ont fortement influencé ce format qui n'est alors pas assez général, complet et pas suffisamment structuré pour pouvoir envisager des sorties dans des langages différents.

Nous avons aussi examiné divers environnements de programmation visuelle à but pédagogique (Alice (Alice, 2000), AlgoBox (Algobox, 2010) par exemple) ou de développement de logiciels. Ces environnements proposent une interface permettant de créer de façon visuelle des programmes sans avoir à taper explicitement des lignes de code, l'environnement se chargeant ensuite de produire

automatiquement les lignes de code correspondantes, de les compiler et de les exécuter. Cependant ces environnements définissent en fait un langage de programmation propre spécifique à leurs besoins et les détails de stockage des programmes ne sont pas documentés ou sont des formats XML ad-hoc qui ne sont que la traduction en XML de la syntaxe du pseudo-code utilisé.

Notre but a été, à l'instar de MathML pour les mathématiques, de définir un format XML pivot de référence de représentation d'algorithmes et de programmes. Ce format, algoML, n'est pas un n-ème langage de programmation déguisé sous une syntaxe XML, mais un format de stockage qui permet de représenter avec une très grande souplesse d'utilisation un algorithme ou un programme (pour l'instant de programmation impérative). Il permet notamment aux enseignants et aux concepteurs de logiciels de pourvoir disposer d'une syntaxe unique à partir de laquelle il est possible de générer (facilement grâce aux nombreux outils XML standards) automatiquement la représentation concrète avec laquelle ils travaillent, syntaxe unique qui leur permet alors aussi de partager leurs travaux.

Conception de AlgoML

Nous expliquons dans cet article les lignes directrices qui ont guidé la conception de ce format, notamment : la volonté de représenter toutes les étapes d'un raisonnement algorithmique sans se restreindre à un langage particulier ; avoir un format général et polyvalent qui puisse par exemple représenter à la fois des algorithmes de haut niveau et des programmes précis avec des instructions très détaillées ; pouvoir représenter le raisonnement algorithmique d'analyse descendante. Pour finir nous détaillons l'utilisation concrète que nous en avons fait dans le projet Unisciel que nous avons développé et indiquons les outils que nous avons déjà développés ainsi que les possibilités offertes. Pour avoir une petite idée du format, nous présentons un exemple en annexe.

Structuration d'algoML suivant le raisonnement algorithmique

Nous avons structuré algoML en suivant la structuration du raisonnement algorithmique, c'est-à-dire nous avons fait attention à ce que chaque outil algorithmique, chaque étape du raisonnement algorithmique soit représentée explicitement dans AlgoML. Pour illustrer cette réflexion, nous donnons quelques exemples concrets.

Dans la résolution de certains problèmes, il faut recommencer plusieurs fois l'exécution d'un morceau de code au moyen d'une boucle ; suivant le problème, il faut ensuite choisir le bon type de boucle : boucle qui se répète un nombre connu de fois, boucle qui se répète tant qu'une condition est vraie, … Dans les langages de programmation courants on utilise directement les instructions correspondant à la boucle choisie, et dans certains langages on peut techniquement utiliser un type de boucle à la place d'un autre (notamment en C, C++ et java, la boucle for qui sert à répéter des instruction un nombre connu de fois est souvent utilisé à la place de la

boucle while qui elle se base sur une condition), par exemple parce que le code paraît plus court ou plus « joli » à écrire, au détriment alors de la sémantique et de la compréhension de ce qui est fait. Pour représenter une boucle dans algoML, nous utilisons l'élément <loop> qui contient les éléments correspondant à la boucle choisie. Cet élément qui n'existe pas dans les langages de programmation permet d'abord d'indiquer explicitement que l'on utilise une boucle (une instruction itérative), avant d'indiquer les détails de la boucle choisie. Cet élément permettrait par exemple à un environnement de programmation visuel de faire un affichage spécifique pour les instructions itératives (d'une certaine couleur et d'une certaine forme par exemple). Cet élément permet aussi de contenir les commentaires qui expliquent le but de la boucle. Nous avons ensuite conçu les différents éléments correspondant aux différentes boucles de manière à ce qu'ils reflètent bien leur utilisation logique. Par exemple la boucle <while>, boucle « tant que », est articulée autour d'une condition, alors que la boucle <for> , boucle « pour » est articulée autour des valeurs parcourues par un le compteur, et on ne peut pas détourner l'utilisation la boucle <for> pour en faire une boucle <while> cachée par exemple.

De la même façon, lorsqu'on écrit une procédure (un sous-programme) il faut d'abord réfléchir à ce qu'elle doit faire avant de réfléchir à comment elle va le faire, c'est-à-dire réfléchir à son utilisation : quels sont ses paramètres (les informations qu'elle va utiliser) et surtout comment sont-ils passés (pour faire des calculs, ou pour que la procédure y range des résultats en retour). Pour bien représenter toutes les étapes du raisonnement, nous avons défini l'élément <procedure> qui indique que l'on définit une procédure. Cet élément contient ensuite l'élément <header> qui définit l'en-tête de la procédure, c'est-à-dire son mode d'emploi : il contient son nom (dans un élément <name>) et la liste de ses paramètres (dans un élément <parameters>) ; chaque paramètre de la liste (chacun dans un élément <parameter>) indique son nom, éventuellement son type, et surtout son mode de passage qu'il est obligatoire d'indiquer. Ce mode de passage précise s'il s'agit d'une donnée ou d'un résultat de la procédure, ce qui entraîne un comportement différent. Un environnement qui utilise algoML forcerait alors l'utilisateur à réfléchir à cet aspect crucial dont les étudiants ont tendance à ne pas se préoccuper. Nous avons aussi distingué trois types de passage de paramètres qui correspondent aux trois utilisations logiques différentes (en entrée pour utiliser la valeur, en sortie pour y ranger une valeur, en entrée-sortie pour mettre à jour la valeur à partir de la valeur initiale), même si dans la plupart des langages de programmation ces passages se réduisent à deux types de passage : l'important est de représenter le raisonnement et non pas les détails techniques de tel ou tel langage.

Représentation des concepts algorithmique généraux

Comme déjà entrevu dans la section précédente au sujet des passages de paramètres, nous avons conçu algoML pour qu'il puisse représenter les raisonnements algorithmiques dans toute leur généralité sans être restreint par telle

ou telle particularité courante des langages de programmation. Cela signifie que nous avons gardé toutes les distinctions faites en algorithmique, même si les langages de programmation usuels ne les font pas.

Par exemple, on peut vouloir répéter un certain nombre de fois des instructions dans deux types de situations différentes : vouloir juste répéter n fois les instructions, ou faire varier un compteur entre des limites données et utiliser la valeur de ce compteur dans les instructions. Les langages de programmation ne définissent en général qu'une seule façon de réaliser ce genre de boucle en utilisant un compteur (en choisissant les bonnes limites on peut aussi faire exécuter les instructions le nombre de fois voulu). Nous avons cependant gardé la distinction entre ces deux utilisations différentes de cette boucle : l'élément <for> correspondant contient soit un élément <numberiter> indiquant le nombre d'exécutions de la boucle, soit un élément <iterator> qui contient les détails (nom et limites de variations) du compteur de la boucle.

Nous avons conçu algoML pour qu'il puisse aussi représenter des algorithmes généraux de haut niveau (par exemple : « pour tous les éléments du tableau »), qui ne peuvent pas être traduits directement dans un langage de programmation, mais qui peuvent être traduits dans un langage de pseudo-code algorithmique. À côté des éléments représentant des expressions ou des instructions précises, nous avons prévu par exemple les éléments <expression> et <instruction> qui représentent une expression et une instruction de haut niveau décrite de façon textuelle informelle. Cela permet d'utiliser algoML aussi dans des contextes purement algorithmiques (c'est-à-dire où on se contente d'indiquer une méthode générale), soit parce que l'on ne s'intéresse qu'au raisonnement algorithmique, soit parce que l'on établit d'abord une solution de haut niveau avant de la détailler ensuite pour aboutir à des instructions qui pourront être programmées.

Programmation hiérarchique descendante

Pour éviter d'avoir à maintenir séparément un programme et sa documentation, Donald Knuth (Knuth 1984) inventa en des outils de *literate programming* permettant d'écrire en parallèle dans un même document des fragments de code (dans le langage de programmation du projet) et du texte explicatif, l'auteur n'étant pas restreint à suivre l'ordre séquentiel du programme mais pouvant suivre un ordre qu'il juge plus logique (l'ordre du manuel d'utilisation par exemple). Des outils permettent alors d'extraire du même document d'une part la documentation, d'autre part le code qui sera compilé pour former le logiciel.

Nous avons fait en sorte qu'algoML puisse faire la même chose, non pas par rapport à l'écriture générale d'un manuel, mais de façon plus restreinte par rapport à la décomposition algorithmique procédurale descendante, le raisonnement de base enseigné en algorithmique. Dans ce raisonnement, on découpe le problème en une série d'étapes générales, puis chaque étape est alors examinée indépendamment et redécoupée en étapes plus précises et ainsi de suite jusqu'à arriver au niveau où

l'algorithme est exprimé avec les instructions de base et peut alors être traduit dans un langage de programmation. AlgoML permet de stocker toutes les étapes de cette décomposition procédurale descendante, l'utilisateur pouvant alors choisir le niveau de découpage qu'il souhaite utiliser à un moment donné.

Pour cela, algoML les éléments <expression> et <instruction> qui permettent de représenter une expression ou une instruction générale de haut niveau peuvent aussi contenir un élément <doc type="description"> où est maintenant précisée la description de haut niveau, suivie alors des éléments correspondant aux instructions détaillées de la décomposition algorithmique. Le contenu détaillé peut lui- même contenir de tels éléments <instruction> et <expression> permettant d'obtenir plusieurs niveaux de détail. Une application peut alors facilement utiliser le niveau de détail voulu (nous avons par exemple fait une feuille de style XSLT qui permet de produire le contenu algoML au niveau de détail voulu).

Utilisation d'algoML et réalisations

À l'heure actuelle, algoML permet de représenter les outils de base de la programmation impérative. Il possède cinq types simples (entier, réel, booléen, caractère, chaîne de caractères) et deux types structurés (tableaux et structures), avec le moyen de donner des valeurs littérales pour ces types. Pour faire les calculs, il possède les opérateurs habituels ainsi que les fonctions mathématiques usuelles, et la possibilité d'appeler des fonctions et des procédures en général. algoML possède les instructions simples (déclaration de variable, affectation, entrées/sorties) et les différentes instructions conditionnelles (si sinon et choix multiple) et itératives (boucles tant que faire, répéter tant que, répéter jusqu'à, pour). algoML permet de définir des fonctions et des procédures. Bien qu'il ne soit pas encore possible de déclarer des classes pour l'instant, il est déjà possible de les utiliser (créer des objets et appeler des méthodes dessus). L'utilisation entière de la programmation orientée objets est prévue pour la prochaine version du format. Enfin algoML permet l'utilisation des fichiers et la gestion des bibliothèques/paquetages.

Nous avons utilisé algoML pour représenter tous les algorithmes et programmes d'un module en ligne d'algorithmique et programmation de l'UNT Unisciel. Nous avons développé des feuilles de style XSLT qui permettent ensuite de produire automatiquement la traduction en C++ et en un pseudo-langage algorithmique. Nous avons pu ainsi, lors de l'écriture de ce module, vraiment nous concentrer uniquement sur les questions didactiques de l'algorithmique sans être restreints ou contraints par tel ou tel langage de programmation particulier. S'il y a besoin d'utiliser un autre langage de programmation (pour faire tourner les programmes sur ordinateurs), il suffit juste d'écrire par exemple la feuille de style XSLT qui fera la traduction de algoML vers ce langage. La grande structuration d'algoML guidée par la considération pédagogique de tout bien représenter explicitement (et visible

par exemple dans l'exemple de l'annexe), a aussi rendu plus facile l'écriture de ces feuilles de style XSLT.

Nous avons écrit un schéma de validation XML (écrit en RelaxNG) du format algoML, c'est-à-dire un document décrivant les éléments algoML autorisés et leur structuration. Ce schéma permet par exemple à un éditeur XML de vérifier que le document écrit est bien conforme au format algoML et de guider l'utilisateur dans l'édition du document en lui proposant les bonnes balises à utiliser aux bons endroits.

Un schéma de validation XML seul ne permet pas d'exprimer toutes les contraintes nécessaires à certains langages de programmation (par exemple devoir déclarer une variable avant de l'utiliser). Grâce à la forte structuration du format algoML, toutes les informations nécessaires pour faire ces vérifications sont présentes dans la structure XML même du document et facilement utilisables avec les outils XML habituels, sans devoir mettre en œuvre des techniques compliquées d'analyse syntaxique issues de la théorie de la compilation. Nous avons ainsi écrit quelques feuilles de styles XSLT qui font certaines de ces vérifications.

Conclusion

Nous avons présenté algoML, un format XML pivot destiné à représenter des algorithmes et des programmes indépendamment des spécificités d'un langage de programmation particulier. La conception de ce format a été guidée par la volonté de représenter tous les éléments du raisonnement algorithmique, notamment pour pouvoir les retrouver au moment de les enseigner. Il est ensuite possible, par application de feuilles de style XSLT par exemple, de traduire algoML dans le langage de programmation souhaité. De plus, de par sa forte structuration, algoML permet aussi l'utilisation des outils XML pour faire les vérifications usuelles faites par les compilateurs des langages de programmation.

Nous avons déjà utilisé ce format pour représenter les algorithmes d'un cours d'algorithmique et programmation et pour produire automatiquement les traductions dans différents langages (notamment C++). D'une manière générale, nous souhaitons que ce format puisse être librement utilisé et servir de base pour divers projets. Nous envisageons par exemple de produire un logiciel de développement visuel d'algorithmique basé sur ce format, c'est-à-dire un logiciel qui présenterait de façon graphique les divers éléments algorithmiques, ce qui éviterait à l'étudiant d'avoir à apprendre une syntaxe particulière.

Bibliographie

Badros, G. (2000). JavaML: a markup language for Java source code. Computer Networks, 33 (6), 159-177.

Boshernitsan, M., Graham, S., (2000). Designing an XML-based Exchange Format for Harmonia. In *Proceedings of WCRE'00'* (pp. 287-289). Brisbane.

Collard, M.L., (2005). Addressing Source Code Using srcXML. In *IEEE International Workshop on Program Comprehension Working Session IWPC'05* (3 pages). St. Louis.

Mamas, E., Kontogiannis, K., (2000). Towards Portable Source Code Representation Using XML. In *Proceedings of WCRE'00'* (pp. 172-182). Brisbane.

Knuth, D. (1984). Literate Programming. Computer Journal, 27 (2), 97-111.

Seidel, F. (2004). *Entwurf einer XML-Sprache zur programmiersprachenunabh¨angigen Formulierung von Algorithmen und einer Transformationskomponente für Java und C++*. Diplomarbeit, Université de Nantes.

Algobox (2010). Algobox, Logiciel pédagogique d'aide à la création et à l'exécution d'algorithmes. http://www.xm1math.net/algobox/index.html

Alice (2000). Alice, An Educational Software that teaches students computer programming in a 3D environment. http://alice.org

Annexe : un exemple de algoML

```
<algoML xmlns="http://www.algoml.org" version="1.0">
  <function>
    <header>
      <doc type="annotation">calcule le sinus cardinal en faisant
attention à x petit</doc>
      <name>sinc</name>
      <parameters>
        <parameter type="input">
          <doc type="annotation">valeur de calcul</doc>
          <name>x</name>
          <type><real/></type>
        </parameter>
        <parameter type="input">
          <doc type="annotation">limite de précision pour 0</doc>
          <name>epsilon</name>
          <type><real/></type>
        </parameter>
      </parameters>
      <return>
        <doc type="annotation">sinus cardinal</doc>
        <type><real/></type>
      </return>
    </header>
    <body>
      <test>
        <if>
          <doc type="annotation">|x|&lt;epsilon, x trop petit :
assimilé à 0</doc>
          <lt>
            <operand>
              <abs>
                <arguments>

<argument><parameter><name>x</name></parameter></argument>
                </arguments>
              </abs>
            </operand>
            <operand>
              <parameter><name>epsilon</name></parameter>
            </operand>
          </lt>
        </if>
        <then>
          <doc type="annotation">x = 0 : sinc(x)=1</doc>
          <return><real>1</real></return>
        </then>
        <else>
          <doc type="annotation">sinc(x)=sin x / x</doc>
          <return>
            <divide>
              <operand>
                <sin>
```

```
                    <arguments>

<argument><parameter><name>x</name></parameter></argument>
                    </arguments>
                  </sin>
                </operand>
                <operand>
                  <parameter><name>x</name></parameter>
                </operand>
              </divide>
            </return>
          </else>
        </test>
      </body>
    </function>
</algoML>
```