Propositions d'extensions à IMS-QTI 2.1 pour l'expression de contraintes sur les variables d'exercices mathématiques

Odette Auzende, Hélène Giroire, Françoise Le Calvez

Propositions d'extensions à IMS-QTI 2.1 pour l'expression de contraintes sur les variables d'exercices mathématiques

Extensions à QTI 2.1 pour l'expression de contraintes

Odette AUZENDE, Hélène GIROIRE, Françoise LE CALVEZ

Equipe MOCAH, LIP6, Université Paris 6,
104 avenue du président Kennedy, 75016 Paris
Odette.Auzende@lip6.fr, Helene.Giroire@lip6.fr, Francoise.Le-Calvez@lip6.fr

RÉSUMÉ. *La spécification IMS-QTI 2.1 permet, via la notion de variable et de contrainte, d'introduire une certaine variabilité dans les descriptions d'exercices destinés au Web. Mais cette possibilité ne suffit pas à traiter les cas où les variables d'un exercice sont interdépendantes. Nous montrons donc les limites de la spécification et en proposons des extensions afin de pouvoir définir des modèles d'exercices mathématiques faisant appel à des variables interdépendantes. Nous présentons ensuite l'éditeur de contraintes QTI_CE générant un fichier QTI 2.1 étendu et permettant à un enseignant–auteur d'entrer ses contraintes, de les visualiser et de les tester. Nous décrivons enfin le générateur de pages web dynamiques QTI_WPG permettant, à partir d'un modèle d'exercice QTI 2.1 étendu, de créer autant de « clones » d'un exercice que souhaité.*

MOTS-CLÉS : *IMS-QTI, modèles d'exercices, contraintes, exercices mathématiques, interopérabilité*

1 Introduction

Les nouvelles technologies permettent à chacun de se former où et quand il le souhaite. Le métier d'éditeur de contenu s'en trouve considérablement modifié, la plupart des éditeurs proposant maintenant en ligne des cours, des exercices et des tests pour toute matière et tout niveau. Rien qu'en Europe et dans le domaine des mathématiques, Caprotti [CAPROTTI et al. 05] estime que 16 à 24 millions de pages sont consultées par semaine, pour du travail à domicile.

Parmi les systèmes créant automatiquement des exercices et des tests dont les variables changent de valeurs à chaque exécution, citons Euler [EULER] et WIMS [WIMS] dans lequel des patrons d'exercices permettent aux enseignants de créer des exercices interactifs simples. La création automatique d'exercices pose cependant problème lorsque les variables sont liées par des contraintes complexes. L'enseignant spécifie les contraintes puis l'informaticien écrit le programme chargé de générer les valeurs des variables. WIMS [WIMS] propose ainsi un outil réservé aux seuls développeurs, permettant la création de modules d'exercices complets.

Le codage des exercices est souvent effectué en suivant une démarche propriétaire, adaptée à la plate-forme utilisée ; les fichiers ne peuvent donc pas être facilement déplacés d'une plate-forme vers une autre. L'expérience prouve en outre que le programme codé ne traduit pas toujours exactement les contraintes formulées par l'enseignant, générant alors des exercices d'un niveau différent de celui prévu par l'auteur.

Un projet RIAM (Recherche et Innovation en Audiovisuel et Multimedia), auquel participent l'éditeur Odile Jacob Multimedia et l'équipe MOCAH du LIP6, a pour but de développer une chaîne de production de contenus interactifs autorisant une génération de contenus dynamiques contrôlée par les auteurs et favorisant l'interopérabilité entre plates-formes.

Dans le domaine des ressources pédagogiques, des standards autorisent l'interopérabilité entre systèmes : SCORM [SCORM] et IMS-Learning Design [IMS-LD] concernent l'organisation des contenus pédagogiques, LOM [LOM] l'expression des méta-données pédagogiques. Les plates-formes d'e-learning permettent la création de ressources pédagogiques, suivant généralement des schémas XML propriétaires. Si certaines d'entre elles (telle que Moodle [MOODLE] ou Sakai [SAKAI]) sont capables d'importer des fichiers XML créés avec quelques autres outils, ce n'est ni une généralité, ni suffisant pour assurer une interopérabilité globale. Si aucun standard n'est encore unanimement reconnu actuellement pour l'expression d'exercices, la spécification QTI (Question and Test Interoperability) définie par le consortium IMS semble en bonne voie d'y parvenir [IMS-QTI 06] [VOGTEN et al. 06].

La génération automatique d'exercices dans le domaine mathématique pose des problèmes spécifiques, notamment l'expression et l'affichage d'expressions comportant des fractions ou des racines carrées, ainsi que le calcul d'expressions

symboliques pour évaluer les réponses. Pour y remédier, certains standards ont été définis [GOGUADZE et al. 06].

MathML [MATHML] et OpenMath [BUSWELL et al. 04] sont des schémas XML développés pour coder d'une part la *présentation* d'objets mathématiques, d'autre part leur *contenu* spécifiant leur sémantique, tandis qu'OMDoc (Open Mathematical Documents) [OMDOC] est une extension d'OpenMath et du contenu MathML destiné au Web. Ces standards ne permettent cependant pas de représenter l'ensemble du processus lié à l'exécution d'exercices, notamment les interactions, l'évaluation et le retour à l'apprenant. D'autres projets, tels que LeActiveMath en Allemagne [MANZOOR et al. 05], WebALT [WEBALT], "Serving Maths" au Royaume-Uni, n'appréhendent ni la déclaration de contraintes, ni la génération, en grand nombre, de clones d'exercices à partir d'un modèle. La spécification IMS-QTI (Question and Test Interoperability) permet de représenter à la fois le contenu d'exercices et leur traitement, en utilisant la partie présentation de MathML pour l'affichage des objets mathématiques. Le projet « Serving Maths" enrichit QTI en y intégrant OpenMath, créant ainsi MathQTI [GOGUADZE et al. 06] [MAVRIKIS 05]. Mais nous verrons que QTI a ses limites, puisqu'il ne permet pas de déclarer des contraintes complexes entre variables interdépendantes.

L'éditeur français Odile Jacob Multimedia (OJM) propose plusieurs centaines d'exercices à environ 300 collèges. La principale caractéristique des exercices scientifiques d'OJM est leur variabilité, chaque exercice étant codé comme un modèle associé à des contraintes sur les variables [OJM]. Mais le codage des contraintes spécifiées par l'enseignant est fait « à la main » selon un format lié à la plate-forme propriétaire BABEL d'OJM. Il n'y a donc ni interopérabilité, ni possibilité de contrôle, par l'enseignant, de la conformité du programme aux contraintes.

Pour remédier à cet état de chose, nous travaillons sur la représentation de modèles d'exercices dans le domaine scientifique. Notre but est de permettre aux enseignants-auteurs de créer des modèles d'exercices qui seront ensuite instanciés en ligne, les valeurs des variables étant choisies aléatoirement tout en respectant les contraintes que les enseignants-auteurs ont spécifiées. Pour que ces exercices soient exécutables sur différentes plates-formes, nous avons choisi d'utiliser la spécification IMS-QTI 2.1 que nous avons enrichie afin de pouvoir y définir l'ensemble des contraintes des exercices d'OJM.

Nous présentons tout d'abord la structure d'un modèle d'exercice suivant la spécification IMS-QTI 2.1 et les limites de cette spécification. Nous proposons ensuite d'étendre IMS-QTI 2.1 afin d'exprimer de nouvelles contraintes. Nous décrivons le logiciel QTI_CE qui permet aux enseignants–auteurs d'exprimer en IMS-QTI 2.1 étendu les contraintes relatives aux variables d'un exercice et de vérifier que les valeurs générées correspondent à leurs attentes. Nous présentons enfin le logiciel QTI_WPG, qui génère, à partir d'un modèle d'exercice au format QTI 2.1 étendu, des pages Web dynamiques, versions exécutables des exercices dans lesquelles les valeurs des variables sont générées à chaque appel.

2 La spécification IMS-QTI 2.1 et ses limites

Le consortium Instructional Management Systems (IMS) Global Learning définit des spécifications techniques pour l'interopérabilité d'applications de e-learning. Il a défini les spécifications QTI v1 puis QTI v2 pour spécifier l'édition et le jeu d'exercices. Actuellement, des plates-formes d'e-learning utilisent les versions 1 pour l'importation ou l'exportation de fichiers d'exercices. Mais dans les versions 2.0 et 2.1, spécifiées respectivement en 2005 et 2006, la définition des exercices a été réorganisée et de nouveaux concepts sont introduits, notamment la notion de **variable**. Chaque exercice est un *assessmentItem*, dont la structure est présentée dans la figure 1. La classe *assessmentItem* comporte des éléments concernant les variables de l'exercice (*responseDeclaration*, *outcomeDeclaration*, *template Declaration*, *templateProcessing),* des éléments concernant la présentation de l'exercice *(stylesheet* et *itemBody),* et d'autres spécifiant comment doivent être évaluées les réponses et le retour à l'apprenant (*responseProcessing*, *modalFeedback).*

assessmentItem	
responseDeclaration	déclare les variables réponses
outcomeDeclaration	déclare les variables de sortie
templateDeclaration	déclare les variables de l'exercice
templateProcessing	exprime les règles permettant d'initialiser les variables
stylesheet	fait le lien avec la feuille de style utilisée pour la présentation
itemBody	définit la présentation de l'exercice et l'interaction
responseProcessing	définit l'évaluation et les valeurs en sortie
modalFeedback	définit le retour à l'apprenant après évaluation

Figure 1. *Structure de l'élément assessmentItem (QTI 2.1)*

La classe *templateDeclaration* permet de déclarer les variables de l'exercice. La classe *templateProcessing* contient des éléments des classes *exitTemplate*, *setCorrectResponse*, *setDefaultValue*, *setTemplateValue* et *templateCondition* dérivées de la classe abstraite *templateRule* ; elle décrit l'initialisation des variables et des réponses. Des opérateurs mathématiques permettent de définir des éléments de type *expression*. A titre d'exemple, considérons deux variables entières a et b. La variable a est comprise entre 2 et 6. La valeur de b dépend de celle de a : si a est supérieur à 4, alors b est compris entre 2 et 6 ; sinon, b est compris entre 10 et 15. La variable REPONSE est définie comme le produit formel de a par b. La figure 2 montre l'ensemble de ces déclarations en QTI 2.1.

```
<responseDeclaration identifier="REPONSE" cardinality="single" baseType="integer" />
...
<templateDeclaration identifier="a" cardinality="single" baseType="integer" />
<templateDeclaration identifier="b" cardinality="single" baseType="integer" />
<templateProcessing>
        <setTemplateValue identifier="a"><randomInteger min="2" max="6" /></setTemplateValue>
        <templateCondition>
                <templateIf>
                        <gt><variable identifier="a" /><baseValue baseType="integer">4</baseValue></gt>
                        <setTemplateValue identifier="b"><randomInteger min="2" max="6" /></setTemplateValue>
                </templateIf>
                <templateElse>
                        <setTemplateValue identifier="b"><randomInteger min="10" max="15" /></setTemplateValue>
                </templateElse>
        </templateCondition>
        <setCorrectResponse identifier="REPONSE">
                <product><variable identifier="a" /><variable identifier="b" /></product>
        </setCorrectResponse>
</templateProcessing>
```

Figure 2. *Déclaration de variables dans un exercice QTI 2.1*

Dans cet exemple, a et b sont liés, mais b ne peut pas être initialisé avant que a ne le soit. Il y a donc une **dépendance séquentielle** entre a et b.

Mais il est indispensable de pouvoir exprimer **d'autres contraintes** dans les exercices mathématiques au niveau collège. En voici deux exemples, caractérisés par l'**interdépendance** entre les variables :

Exemple 1 : a et b sont des entiers tels que a est compris entre 2 et 10, b est compris entre 2 et 40, b est impair et a+b est multiple de 10.

Exemple 2 : n est un entier compris entre 10 et 12 ; t est un tableau de n entiers compris entre 4 et 10 dont la somme des éléments est égale à 65.

L'interdépendance ne peut pas être spécifiée en QTI 2.1. Nous proposons donc dans le paragraphe suivant des extensions à QTI permettant de pallier cette limitation.

Des tableaux peuvent être définis dans QTI 2.1, mais à condition d'**énumérer** tous leurs éléments. La figure 3 illustre ainsi la déclaration d'un tableau de 3 entiers. Il est par contre **impossible** de définir un tableau semblable à celui de l'exemple 2, dont la **taille** serait elle-même une **variable**. Nous lèverons également cette autre limitation au paragraphe suivant.

```
<templateDeclaration identifier="t" cardinality="multiple" baseType="integer" />
<setTemplateValue identifier="t">
        <multiple>
                <randomInteger min="1" max="20" />
                <randomInteger min="1" max="20" />
                <randomInteger min="1" max="20" />
        </multiple>
</setTemplateValue>
```

Figure 3. *Définition d'un tableau de 3 éléments en QTI 2.1*

3 Extensions à la spécification IMS-QTI 2.1

MathQTI [MATHQTI] permet d'exprimer une variable comme fonction d'autres variables, mais ne permet pas de définir des contraintes entre variables. Pour décrire ces contraintes, nous enrichissons *templateProcessing* d'une sous-classe *templateConstraint*, ce qu'illustre la figure 4.

Class: templateProcessing
Associated classes: assessmentItem
Contains: templateRule [1..*]
Contains: templateConstraint [0..*] *pour déclarer une contrainte liant différentes variables*

Class: templateConstraint
Contains: expression [1]

Figure 4. *Extension de la spécification relative à la classe templateProcessing*

Nous enrichissons également la classe *expression* en lui ajoutant les sous-classes *repeat, sumContainer* concernant des tableaux (container) et *gcd, lcm* des opérateurs mathématiques standard pgcd et ppcm (figure 5), rejoignant ainsi les conclusions des travaux didactiques de BOUHINEAU [BOUHINEAU et al. 05] pour l'obtention de patrons effectifs d'exercices.

Abstract class: expression
Derived class: ..., **repeat, sumContainer, gcd, lcm**

Class: repeat Attribute: **time [1]:**identifier Contains: expression [1]	*permet de définir des éléments d'une variable de type container pour définir le nombre d'éléments du container*
Class: sumContainer Contains: expression [1]	*renvoie la somme des éléments d'un container*
Class: gcd Contains: expression [2]	*renvoie le plus grand commun diviseur*
Class: lcm Contains: expression [2]	*renvoie le plus petit commun multiple*

Figure 5. *Extension de la spécification de la classe expression*

3.1 Utilisation de l'opérateur repeat

L'opérateur *repeat* permet de définir des tableaux dont la taille est elle-même une variable. Ainsi, si l'on souhaite faire calculer la moyenne de n entiers compris entre 4 et 10, n étant un entier précédemment déclaré, on déclare et initialise le tableau comme le montre la figure 6.

```
<templateDeclaration identifier="n" cardinality="single" baseType="integer" />
<templateDeclaration identifier="t" cardinality="multiple" baseType="integer" />

<setTemplateValue identifier="t">
        <multiple>
                <repeat time="n"><randomInteger min="4" max="10" /></repeat>
        </multiple>
</setTemplateValue>
```

Figure 6. *Déclaration et initialisation d'un tableau de taille n*

3.2 Utilisation de templateConstraint

Les valeurs affectées aux variables doivent satisfaire les conditions précisées dans les éléments *templateConstraint*. Les opérateurs utilisés pour exprimer les contraintes sont les mêmes que ceux permettant de définir les expressions. C'est l'**interprétation** des éléments *templateConstraint* dans le bloc *templateProcessing* qui fait **réitérer** le processus d'instanciation des variables jusqu'à ce qu'elles satisfassent les contraintes. Reprenons les exemples du paragraphe 2 :

Exemple 1 : la contrainte « b impair » s'exprime sous la forme « gcd(B,2)=1 » ; la contrainte « a+b est multiple de 10 » s'exprime en déclarant la variable s=a+b et en imposant la contrainte « s multiple de 10 ». La figure 7 montre les éléments *templateConstraint* permettant d'exprimer ces contraintes.

```
...
<templateProcessing>
        ...
        <setTemplateValue identifier="s"><sum><variable identifier="a" /><variable identifier="b" /></sum></setTemplateValue>
        <templateConstraint>
                <equal>
                        <gcd>
                                <variable identifier="b" />
                                <baseValue baseType="integer">2</baseValue>
                        </gcd>
                        <baseValue baseType="integer">1</baseValue>
                </equal>
        </templateConstraint>
        <templateConstraint>
                <equal>
                        <integerModulus>
                                <variable identifier="s" />
                                <baseValue baseType="integer">10</baseValue>
                        </integerModulus>
                        <baseValue baseType="integer">0</baseValue>
                </equal>
        </templateConstraint>
</templateProcessing>
```

Figure 7. *Expression des contraintes pour l'exemple 1*

Exemple 2 : la déclaration « t est un tableau de n entiers compris entre 4 et 10 » est matérialisée par l'opérateur repeat, comme le montre la figure 6. La contrainte « la somme des éléments est égale à 65 » s'exprime en utilisant l'opérateur *sumContainer*. La figure 8 montre l'élément *templateConstraint* permettant d'exprimer cette contrainte.

```
...
<templateProcessing>
...
        <templateConstraint>
                <equal>
                        <sumContainer><variable identifier="t" /></sumContainer>
                        <baseValue baseType="integer">65</baseValue>
                </equal>
        </templateConstraint>
</templateProcessing>
```

Figure 8. *Expression des contraintes pour l'exemple 2*

Pour permettre aux enseignants–auteurs de définir des modèles d'exercices mathématiques comportant des contraintes entre variables, nous avons construit un environnement qui génère du QTI « étendu » (QTI 2.1 augmenté de nos extensions) et propose d'autres fonctionnalités décrites ci-après.

4 Déclaration des contraintes d'un exercice : QTI_CE

Pour que les apprenants puissent s'entraîner sur des exercices constamment renouvelés, mais conservant un niveau de difficulté constant, un même modèle d'exercice doit pouvoir être décliné en un grand nombre de versions différentes. Un tirage aléatoire des valeurs de variables respectant des contraintes formulées par les enseignants–auteurs permet à la fois de générer des exercices variés et d'en contrôler la difficulté. Dans ce but, nous avons conçu et réalisé un premier logiciel, QTI_CE (QTI Constraint Editor) permettant à un enseignant de préciser les variables et les contraintes d'un exercice, puis un second logiciel, QTI_WPG (QTI Web Pages Generator), qui exploite les fichiers générés par QTI_CE et génère des pages web dynamiques JSP et PHP. Exécutables sur la plupart des plates-formes et des serveurs Web, ces pages dynamiques permettent aux apprenants de s'entraîner en leur offrant à chaque appel une nouvelle version de l'exercice.

4.1 Déclaration des variables

Avec QTI-CE, les enseignants déclarent les variables indépendantes (nom, type, éventuellement nombre de décimales, valeur minimale, valeur maximale), celles qui dépendent d'autres variables (nom, type, nombre de décimales, expression), les tableaux, soit composés de nombres aléatoires dont on précise le nombre, le type et les limites, soit définis en énumérant leurs éléments, soit choisis aléatoirement parmi d'autres tableaux déjà définis. On peut définir des contraintes sur les variables et sur les tableaux. Les contraintes simples utilisent les opérateurs égal, plus grand, plus petit, multiple de, premier avec… et sont définies entre une variable et un nombre ($c < 15$) ou entre variables (c premier avec d). Les contraintes complexes sont des conjonctions ou disjonctions de contraintes simples. La figure 9 illustre une déclaration de contrainte simple, la figure 10 montre l'élaboration d'une disjonction de contraintes simples, la figure 11 présente une déclaration de contrainte concernant la somme des éléments d'un tableau dont la taille est définie par une variable.

Figures 9, 10 et 11. *Déclarations de contraintes avec QTI_CE*

Pendant les déclarations des variables, tableaux et contraintes, QTI-CE génère et affiche, à la volée, les déclarations à la fois en contenu MathML et en QTI 2.1 étendu. En outre, à chaque nouvelle entrée, une interprétation en langage naturel de la déclaration est affichée, permettant à l'enseignant de vérifier, de supprimer et de modifier ses déclarations.

A la sortie du logiciel, QTI_CE sauvegarde quatre fichiers. Le premier contient l'ensemble des déclarations MathML ; il permet aux enseignants de visualiser les contraintes à l'aide d'un navigateur incluant un player MathML. Le deuxième contient l'ensemble des déclarations en QTI 2.1 étendu ; il doit être complété par l'énoncé, la description des interactions, les éléments d'analyse de réponse et de retour à l'apprenant pour obtenir un fichier d'exercice complet. Le troisième fichier est une application java qui attribue, aux variables de l'exercice, un ensemble de valeurs satisfaisant les contraintes ; à chaque exécution, un nouvel ensemble de valeurs est généré ; les enseignants peuvent ainsi visualiser de nombreux jeux de valeurs et vérifier qu'ils répondent à leurs attentes. Le dernier fichier est un fichier JSP contenant les mêmes instructions que l'application java ; exécuté en tant que page web par un serveur Web tel qu'Apache-Tomcat, il affiche à chaque appel, un nouveau jeu de valeurs pour les variables.

Le fichier complet de l'exercice au format QTI 2.1 étendu et les fichiers java sont les fichiers d'entrée du générateur de pages web dynamiques QTI_WPG décrit au paragraphe suivant.

5 Génération de pages web dynamiques : QTI_WPG

En choisissant QTI pour exprimer les contraintes, nous visions l'interopérabilité. Mais deux écueils se sont présentés : le premier est lié au fait qu'à notre connaissance, il n'existe pas encore de player QTI 2.1 capable de jouer des exercices au format QTI 2.1 standard ; le second découle du fait que nous avons dû étendre QTI 2.1 pour pouvoir prendre en compte toutes les contraintes relatives aux exercices constituant la base de notre étude.

Disposant de fichiers d'exercices au format QTI 2.1 étendu, nous avons alors décidé de réaliser non pas un player QTI, mais un générateur de pages dynamiques, QTI-WPG.

Pour chaque modèle d'exercice, à partir du fichier QTI 2.1 étendu spécifiant le corps de l'exercice, le processus d'analyse de la réponse et le retour à l'apprenant, d'une part, et des fichiers java générant les valeurs des variables satisfaisant les contraintes, d'autre part, QTI-WPG génère simultanément deux pages JSP et deux pages PHP. Pour chacune des deux technologies, la première page permet de présenter l'exercice à l'apprenant, d'attendre sa réponse puis de la transmettre à la seconde page, destinée à l'analyse de la réponse et à l'affichage du retour à l'apprenant. A chaque fois que la page de l'exercice est demandée, de nouvelles valeurs des variables sont générées et transmises à la page réponse. Les deux pages

web dynamiques, délivrant ainsi directement des clones du modèle d'exercice, jouent le rôle d'un "clone engine".

A titre d'exemple, considérons le modèle d'exercice : « Développer et réduire le produit suivant : $(\sqrt{a}+c\sqrt{b})(d\sqrt{a}+\sqrt{b})$ » où les contraintes liant les entiers a, b, c et d, précisées par l'enseignant, sont :

$2 \le a < 7$	$a \neq 4$
$2 \le b < 7$	$b \neq 4$
$a \neq b$	pgcd(a, b)=1
$2 \le c < 6$	$2 \le d < 6$
$(a = 2) \vee (a = 3) \supset (b \neq 6)$	$(a = 6) \supset (b \neq 2) \wedge (b \neq 3)$

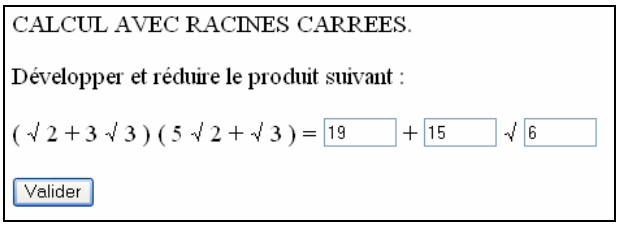
L'avant-dernière contrainte est traduite dans QTI_CE par deux contraintes déclarées séparément :

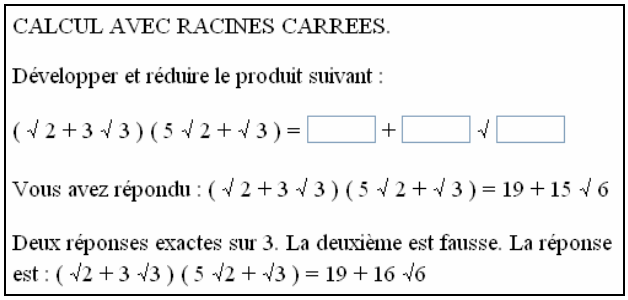
$(a \neq 2)$ ou $(b \neq 6)$ et $(a \neq 3)$ ou $(b \neq 6)$

De manière analogue, la dernière contrainte est traduite par :

$(a \neq 6)$ ou $(b \neq 2)$ et $(a \neq 6)$ ou $(b \neq 3)$

Le fichier des déclarations au format QTI 2.1 étendu généré par QTI_CE est ensuite complété par l'énoncé de l'exercice, la description de l'interaction, les éléments d'analyse de réponse et de retour à l'apprenant. QTI_WPG reçoit ensuite en entrée le fichier d'exercice complet et les fichiers java générés par QTI_CE. Il génère les pages JSP et les pages PHP permettant le jeu de l'exercice avec des valeurs différentes des variables à chaque appel. Les figures 12 et 13 montrent un exemple d'exécution des pages PHP générées par QTI-WPG pour cet exercice.





Figures 12 et 13. *Une exécution de l'exercice et le retour à l'apprenant*

Différents exemples, accompagnés des fichiers MathML, QTI, java, JSP et PHP générés, peuvent être consultés à l'adresse http://webia.lip6.fr/~lecalvez/QTI/

6 Conclusion

Nous avons défini des extensions à IMS-QTI 2.1 qui permettent aux enseignants de créer des modèles d'exercices mathématiques avec des contraintes complexes sur des variables interdépendantes. Ces extensions sont proposées au consortium IMS-QTI. Les contraintes sont exprimées en utilisant les opérateurs usuels (opérateurs arithmétiques, de comparaison, logiques) et peuvent concerner des tableaux de variables dont la taille est elle-même une variable.

Grâce à ces extensions, toutes les contraintes relatives aux variables des exercices mathématiques de la base d'OJM créés par des enseignants-auteurs peuvent être exprimées. L'éditeur de contraintes QTI_CE permet à ces enseignants de visualiser les contraintes, de vérifier la faisabilité de l'instanciation des variables et d'obtenir autant de jeux de valeurs qu'ils le souhaitent. Les fichiers générés par QTI_WPG proposent à l'apprenant, à chaque chargement, de nouveaux clones des exercices.

L'extension essentielle proposée est l'ajout d'une classe *templateConstraint* dans la classe *templateProcessing* permettant la déclaration de contraintes liant des variables interdépendantes dont l'instanciation ne peut pas être faite séquentiellement. QTI_CE pourrait être enrichi en prenant en compte l'ensemble des opérateurs de la classe *expression* de QTI 2.1, mais l'utilisation de QTI_CE et QTI_WPG montre la pertinence des extensions proposées.

Remerciements

Cette recherche a été menée dans le cadre d'un projet RIAM incluant l'éditeur Odile Jacob Multimedia et l'équipe MOCAH du LIP6. Nous remercions Lionel Leyrat qui a étudié IMS-QTI 2.1 durant son stage terminal de Master Recherche à l'université Paris 6, financé par le projet RIAM.

Bibliographie

[BOUHINEAU et al. 05] Bouhineau D., Bronner A., Chaachoua H., Nicaud J-F., (2005) Patrons d'exercices pour APLUSIX, *EIAH 2005*, pp 377-382.

[CAPROTTI et al. 05] Caprotti O., Carlson L., Seppälä M., Strotmann A., (2005) Web Advanced Learning Technologies for Assessment in Mathematics, *MICTE 2005* http://www.formatex.org/micte2005/391.pdf

[GOGUADZE et al. 06] Goguadze G., Mavrikis M., Gonzales Palomo A., (2006) Interoperability issues between markup formats for mathematical exercises, *WelAlt 2006 Proceedings*, Seppala M., Xambo S., Caprotti O., éditeurs, pg 69-80, consultable à http://www.activemaths.org/~george/work/pubs/webalt3.pdf (consulté en 2006)

[MANZOOR et al. 05] Manzoor S., Libbrecht P., Ullrich C., Melis E., Authoring presentation for OpenMath, *4th International Conference MKM 2005* Bremen Germany 2005, July 15-17 www.leactivemath.org/

[VOGTEN et al. 06] Vogten H., Martens H., Nadolski R; Tattersall C., van Rosmalen P., Koper R., CopperCore Service Integration – Integrating IMS Learnng Design and IMS Question and Test Interoperability, *ICALT 2006*, IEEE Computer Society pp. 378-382

Références sur le Web

[BUSWELL et al. 04] Buswell S., Caprotti O., Carlisle D., Dewar M., Gaëtano M., Kohlhase M., (2004) The OpenMath Standard, version 2.0, The OpenMath Society (Available at http://www.openmath.org/) (consulté en 2007)

[EULER] Euler http://euler.ac-versailles.fr/ (consulté en 2007)

[IMS-LD] IMS-LD http://www.imsglobal.org/learningdesign/ (consulté en 2007)

[IMS-QTI 06] IMS Question and Test Interoperability (2006), Version 2.1 public draft (revision 2) specification, available at http://www.imsglobal.org/question/ (consulté en 2007)

[LOM] LOM http://ltsc.ieee.org/wg12 (consulté en 2007)

[MATHML] MathML http://www.w3.org/TR/REC-MathML/toc.html (consulté en 2007)

[MAVRIKIS 05] Mavrikis, M. (2005) MathQTI draft specification http://www.maths.ed.ac.uk/mathqti/docs/v0p3/mathqti_v0p3.pdf (consulté en 2007)

[MOODLE] Moodle http://moodle.org (consulté en 2007)

[OJM] OJM exercises http://www.tdmaths.com/ (consulté en 2007)

[OMDOC] OMDoc http://www.mathweb.org/omdoc/ (consulté en 2006)

[SAKAI] Sakai http://sakaiproject.org/index.php (consulté en 2007)

[SCORM] SCORM http://www.adlnet.org (consulté en 2007)

[WEBALT] WebAlt http://webalt.math.helsinki.fi/ (consulté en 2007)

[WIMS] WWW Interactive Mathematics Server http://wims.unice.fr/wims/ (consulté en 2007)