Intégration des Langages pour la Gestion de Documents

Une Nouvelle Etape dans l'Evolution de XML

Catherine PUGIN, Rolf INGOLD

Département d'Informatique, Université de Fribourg

Mots-clés : XML, modélisation, transformation, intégration

Keywords: XML, modeling, transformation, integration

Résumé : Cet article présente une proposition d'intégration de trois langages principaux pour le traitement des documents semi-structurés : langages de balisage, de modélisation et de transformation. XSD (XML Schema) et XSLT sont les langages de modélisation et de transformation les plus connus. Ils enrichissent le langage de balisage XML car ce sont des applications XML pour XML lui-même. Toutefois, le traitement des documents manque de rigueur car ces langages manquent d'une intégration forte. Nous proposons donc trois nouveaux langages développés sur des fondements solides et des concepts clarifiés : YML (balisage), DML (modélisation) et DGL (transformation). Le développement de ces langages est basé sur l'intégration qui est le point de départ de ce projet. Cet article définit ce que signifie « intégration » et quelles sont les propriétés d'un système intégré. Finalement, nous introduisons une application concrète qui illustre quelques-uns des concepts présentés.

Abstract: This paper presents a study on the integration of three core languages dedicated to the processing of semi-structured documents. As core languages, we define markup, modeling and transformation languages. XSD and XSLT are well-known languages that enhance the XML language since they are XML applications for XML. However, the treatment of documents is not as rigorous as it could be because these languages are not well integrated. Therefore, we introduce three new languages built upon solid foundations: YML (markup), DML (modeling) and DGL (transformation). We apprehend them from the point of view of integration. The paper also defines precisely what is meant by integration in this proposal and what the requirements of an integrated system are. Finally, some of the concepts of integration are illustrated by a typical application.

1 Introduction

Publié pour la première fois en 1998 par le W3C, le langage de balisage XML [1], conçu initialement pour la représentation de documents semi-structurés, a rapidement conquis le monde scientifique et celui de la gestion. Il est aujourd'hui utilisé dans des contextes divers et par des équipes académiques et industrielles variées. Son succès est aussi dû au développement de nombreuses applications XML qui facilitent et uniformisent le traitement des documents, en particulier XSD [6] et XSLT [2] qui, en tant qu'applications XML pour le traitement du XML, enrichissent le langage.

Il a fallu une période de 40 ans aux langages de programmation traditionnels pour atteindre la maturité communément admise du paradigme orienté-objet. XML, bien que riche de ses 10 ans d'expérience et de ses cinq éditions, verra immanquablement sa spécification évoluer vers plus de maturité. Ainsi, XML n'est, à ce jour, qu'un sous-ensemble de SGML dépourvu de ses propriétés les plus obscures [5]. Les langages de modélisation et de transformation, qui, de notre point de vue, forment avec XML les langages centraux pour la gestion des documents semi-structurés, souffrent aussi d'un manque de maturité. Solutions pragmatiques et manque d'interaction entre les différents langages - dû à des intérêts commerciaux évidents - sont souvent justifiés par une rétrocompatibilité indispensable au vu de la large diffusion des documents au format XML.

Nous présentons une expérience qui vise à faire évoluer ces langages vers une nouvelle étape de maturité. La clé de voûte de ce travail est l'intégration des langages de balisage, de modélisation et de transformation. Nous souhaitons développer un système intégré qui soit auto-suffisant et qui permette de traiter des applications complexes de manière simple et naturelle, et ceci, grâce au concept d'intégration.

Les travaux existants dans le domaine de la modélisation et de la transformation de documents sont riches mais, même si certains traitent un certain type d'intégration entre les langages, aucun, à notre connaissance, n'est mené avec l'intégration comme véritable point de mire dès le début de la recherche.

Le système que nous proposons est basé sur trois langages : YML pour le balisage, DML pour la modélisation et DGL pour la génération et la transformation des documents. DML et DGL sont des applications YML et leurs instances respectives des documents YML.

Par intégration, nous entendons les propriétés suivantes pour le système : (1) chaque langage est conçu pour remplir idéalement sa tâche propre et est basé sur des concepts clairs, solides et uniquement dédiés à sa fonction ; (2) les langages sont développés en parallèle et poursuivent le même but, c'est-à-dire une gestion plus simple et plus naturelle des

documents ; (3) ils interagissent les uns avec les autres ; (4) ils bénéficient les uns des autres ; (5) finalement, ils sont soutenus par des spécifications formelles complètes qui assurent une implémentation rigoureuse et dépourvue d'erreurs.
Cet article est organisé de la manière suivante : la section 1 présente un bref état de l'art des langages de modélisation et de transformation pour les documents semi-structurés. La section 2 définit la notion d'intégration. YML et DML sont brièvement introduits dans la section 3 tandis que la section 4 développe plus spécifiquement le langage de transformation DGL. La section 5 s'étend sur l'intégration de ces langages à l'aide d'un exemple concret d'application intégrée. Finalement, la dernière section conclut l'article tout en présentant les perspectives futures de ce travail.

2 Bref état de l'art

Nous présentons dans cette partie les langages les plus significatifs qui permettent de modéliser ou de transformer des documents semi-structurés. Nous constatons ainsi que si l'intégration des concepts est introduite par certains travaux, elle n'est jamais considérée comme point de départ de la recherche.
DTD [1], XSD [6] et Relax NG [4] sont les langages de modélisation les plus utilisés et les plus populaires. DTD et XSD sont des propositions du consortium W3 tandis que Relax NG a été proposé par le consortium OASIS comme une alternative aux deux autres langages. Selon la taxonomie de Murata [10], l'expressivité de ces trois langages est différente : DTD décrit des grammaires locales (deux éléments partageant le même nom ont obligatoirement le même contenu), XSD des grammaires de type simple (deux éléments frères partageant le même nom ont obligatoirement le même contenu), tandis que Relax NG décrit des grammaires d'arbre régulières et est par conséquent le plus expressif des trois.
En plus de ces différences d'expressivité, la conception de chaque langage est dépendante de certains aspects particuliers. Ainsi, DTD est un héritage direct de la famille de langage SGML. Tandis que XML est une simplification de SGML, DTD est une simplification du langage de modélisation dédié à SGML (SGML DTD). DTD possède toujours une syntaxe non-XML qui empêche le langage d'être lui-même modélisé. De plus, il ne considère pas les espaces de nommage qui furent introduits après sa propre spécification. XSD est proposé en 2001 pour répondre aux critiques adressés à DTD. La syntaxe XML, la gestion des espaces de nommage et l'introduction de nombreux types de données prédéfinis en font une bonne alternative à DTD. Malheureusement, le langage, au vu de

sa lourde spécification, est complexe et difficile à appréhender pour des utilisateurs peu expérimentés. Ainsi, Relax NG se profile comme un bon compromis. Il est plus riche que DTD tout en restant plus simple que XSD. Facile à apprendre et à utiliser de manière efficace, il séduit largement, si bien que certains groupes d'intérêt du W3C l'utilisent aujourd'hui pour leurs travaux.
En 1999, le W3C propose le premier langage dédié à la transformation de documents XML : XSLT est de ce fait le langage le plus connu et le plus utilisé pour le traitement de données semi-structurées. Il est indissociable à XPath, un langage pour localiser et sélectionner les différents nœuds d'un document. Toutefois, la version 1.0 du langage souffre d'une spécification peu rigoureuse. Il n'existe ainsi aucune formalisation sémantique du langage et aucun modèle XSD complet n'y est associé, ce qui rend difficile la validation les transformations XSLT, qui sont elles-mêmes des documents XML. Le manque de spécifications claires avait jusqu'à récemment comme conséquence que les différentes implémentations de XSLT ne rendaient pas exactement les mêmes résultats. La deuxième version de XSLT [8], publiée en 2007, inclut une certaine notion d'intégration (« *schema-awareness* ») qui permet, d'une part, de valider les documents d'entrée et de sortie avant l'exécution de la transformation et, d'autre part, de définir des types pour des fonctions XSLT. Toutefois, cette intégration de XSD entraine de nombreuses constructions relativement obscures. De plus, ces concepts ne doivent pas obligatoirement être implémentés et restent ainsi peu usités.
La plus grande faiblesse communément admise de XSLT 1.0 est l'impossibilité d'effectuer un typage statique des transformations. Grâce à celui-ci, il est possible de vérifier avant l'exécution de la transformation que le document produit est valide par rapport au modèle attendu en sortie. Pour répondre à ce manque, différents projets ont vu le jour. Tozawa [15] intègre le typage statique dans un sous-ensemble de XSLT. XDuce [7] permet d'exprimer des contraintes structurelles sur les documents grâce à l'utilisation d'expressions régulières. Il a largement inspiré d'autres projets. Finalement, la librairie XACT [9] pour Java utilise XSD comme formalisme de typage, ce qui consiste également en une certaine forme d'intégration.

3 Propriétés d'intégration

Avant d'introduire les trois langages que nous proposons (YML pour le balisage, DML pour la modélisation et DGL pour la transformation, tous trois dans une syntaxe XML), nous définissons les propriétés du système intégré. Tout d'abord, pour favoriser l'intégration, DML et DGL sont des applications YML de telle sorte que des modèles DML puissent être

définis pour chaque langage et que les instances DML et DGL puissent être validées à l'aide du même outil que les instances YML en général. De la même manière, les instances DML et DGL peuvent être traités par des transformations DGL. De plus, celles-ci produisent tout type d'instances. Ainsi le système peut être qualifié d'auto-suffisant : toutes les instances dans le système peuvent être validées et peuvent être considérées comme le résultat d'une transformation. Aucun autre outil ou langage n'est donc nécessaire pour que le système intégré soit fonctionnel.

L'intégration des langages va encore plus loin et permet d'étendre l'expressivité des langages sans ajouter de nouvelles constructions mais en combinant les langages. Ainsi, si le processus de sélection des nœuds imaginé dans la version de base de DGL permet de sélectionner les nœuds seulement par rapport à leur genre (nœuds textes, éléments, etc.), l'intégration de constructions du langage DML dans DGL permet de sélectionner de manière naturelle les nœuds par rapport à leur type (structure interne). Parallèlement, l'inclusion de constructions du langage DGL dans DML permet de définir des structures dynamiques, c'est-à-dire des structures qui s'adaptent à un contexte donné.

Pour réaliser ce type d'intégration, il est indispensable que les concepts sur lesquels repose chaque langage soient des concepts simples et clairement définis. Pour ce faire, une importance toute particulière est accordée à la définition d'une sémantique rigoureuse. Cette sémantique, définie à l'aide de la sémantique dénotationnelle [14], ne doit pas être considérée comme une valeur ajoutée du projet, qui serait introduite après la spécification des langages. En effet, le développement de chaque langage repose sur la sémantique qui en est le fondement. Durant le développement, chaque concept difficile à formaliser est assimilé à un concept trop compliqué et est alors repensé et simplifié.

De plus, les relations entre les différents langages et les instances de chaque langage doivent être étudiées pour éviter des chevauchements de concepts comme cela arrive parfois dans le monde XML, où le problème des espaces blancs est géré par trois constructions différentes dans chacun des langages : l'attribut *xml:space* dans XML, l'élément *xsl:strip-space* dans XSL et l'élément *whiteSpace* dans XSD. Ainsi, les modèles DML sont organisés de manière hiérarchique avec, au sommet, un modèle universel qui décrit toute instance YML. Les autres modèles héritent de celui-ci. Au bas de la pyramide, chaque modèle permet de valider une seule instance YML. Des mécanismes de dérivation par restriction sont mis en place pour décrire les relations entre les modèles. Comme les modèles sont basés sur des expressions régulières, il suffit de tester leur inclusion pour déterminer si un modèle est plus restreint qu'un autre. De leur côté, les transformations DGL peuvent être typées statiquement. Avec la transformation DGL et le modèle d'entrée, un modèle canonique

est calculé. Si ce modèle canonique est égal ou inclus dans le modèle de sortie, alors la transformation est assurée de produire des documents valides, pour autant que les documents d'entrée soient valides par rapport au modèle d'entrée.
Finalement, il est important que chaque langage possède une syntaxe commune. Ainsi la syntaxe choisie est la syntaxe XML qui, grâce à un mécanisme d'espaces de nommage légèrement modifié, permet de concrétiser l'intégration des langages.
L'avantage de ce système est la possibilité de définir de manière simple et naturelle, des transformations génériques. Une transformation générique est une transformation qui peut être appliquée indépendamment à différentes entrées pour produire une transformation spécialisée qui permet de traiter un certain type de documents. L'utilisateur évite la définition d'un nombre conséquent de transformations, puisque celle-ci est automatisée.

4 Documents et modèles

Les langages de balisage (YML) et de modélisation (DML) ont déjà fait l'objet de diverses publications et sont de ce fait seulement brièvement introduit dans cette partie.
D'une part, YML, dont une première version, fortement remaniée, est présentée dans [11], est un langage de balisage, sous-ensemble de XML, développé après une étude critique minutieuse de XML et conçu dans le but de se départir de toutes les lourdeurs de l'héritage XML : gestion des espaces de nommage, traitement des espaces blancs, unicité de l'élément racine, etc. Après avoir songé à le détacher totalement du monde XML et avoir défini une nouvelle syntaxe pour ce langage, nous avons décidé de nous rapprocher du monde XML afin de profiter des nombreux outils déjà existants. Ainsi un document YML est un document XML soumis à aux restrictions syntaxiques suivantes : (1) la représentation du contenu textuel est différente. Des crochets carrés sont ajoutées avant et après le texte ainsi qu'un croisillon optionnel qui indique un retour à la ligne (*[texte], [texte]#*). Cette représentation permet de distinguer au niveau des instances directement quels espaces blancs sont significatifs (entre les délimiteurs) et lesquels ne le sont pas (à l'extérieur). (2) La déclaration XML est conservée tandis qu'un élément racine *yml* est rendu obligatoire pour se conformer aux règles de XML. Le document YML lui-même est composé des attributs de cet élément ainsi que des nœuds qui constituent son contenu. (3) La déclaration YML est composée de deux attributs inclus dans l'élément *yml*. Il indique la version et l'encodage du document.

D'autre part, DML [13] est une application YML et un langage de modélisation basé sur des grammaires d'expressions régulières. Il peut être analysé de manière totalement déterministe. Un méta-modèle DML normatif permet de valider n'importe quel modèle DML. Le langage est donc complètement auto-descriptif. Les modèles DML sont appréhendés dans une hiérarchie avec, au sommet, un modèle universel qui valide tout document YML. Un mécanisme d'héritage entre les modèles permet, entre autres, de simplifier des opérations de typage. Puisque les contenus sont décrits par des expressions régulières, il suffit de tester l'inclusion de celles-ci pour déterminer les relations d'héritage entre deux modèles.
Dans son approche, DML est proche de Relax NG. L'idée principale est de trouver un compromis entre DTD et XSD. Toutefois DML diffère sur les points suivants : Relax NG ne propose aucun mécanisme d'héritage qui compliquerait le langage. Dans son souci principal d'intégration, cette vision d'ensemble des modèles est importante à maintenir pour DML. Par ailleurs, si Relax NG traite les éléments et les attributs de manière équivalente, DML insiste sur le fait que les attributs représentent des métadonnées sur les éléments et de ce fait ne peuvent être assimilés à des éléments. Deux éléments partageant le même nom mais avec des attributs différents sont considérés comme différents. Finalement, la relation entre documents et modèles est sensiblement différente. Pour Relax NG, la validation est simplement une opération qui peut être appliquée à un document. Pour DML, la validité est une propriété intrinsèque du document, qui doit toujours être valide, au moins par rapport au modèle universel.

5 Génération et transformation de documents

Le langage DGL est une application pour la génération et la transformation de documents YML. Il permet de donc de générer de nouveaux documents YML sans qu'aucun document d'entrée ne soit défini ou de manière plus traditionnelle de transformer un document en un autre. Un programme DGL peut prendre de zéro à plusieurs documents en entrée. Si plusieurs documents sont nécessaires à la transformation, des identifiants sont associés à chaque document et conservés en mémoire. Le modèle de traitement de DGL [12] est proche de XSLT. Un programme DGL est composé de *templates* et de *patterns*. Un template principal initialise la transformation puis les patterns sont constitués de nouveaux templates réunis sous forme de règles. Trois opérations simples forment les templates : un nœud peut être copié, créé ou alors un pattern peut être appliqué sur une liste de nœuds sélectionnés.
Le processus de sélection des nœuds est fortement simplifié par rapport à XPath [3] mais est suffisant pour cette expérience. Il s'effectue étape par

étape à partir d'une première liste de nœuds. Celle-ci contient soit la racine qui est un composant abstrait représentant le document, soit le nœud courant, soit des nœuds définis par un mécanisme de variables qui permet de tenir compte de certains axes. A chaque étape, les nœuds sont sélectionnés selon leur genre jusqu'à la dernière étape. Entre autres spécificités, il est possible d'accéder non seulement aux nœuds mais également à leur contenu, par exemple à la chaîne de caractères qui constitue le nom d'un élément. Ceci permet de copier, par exemple, le nom d'un élément comme valeur d'un nœud texte.
DGL peut être typé de manière statique, comme expliqué précédemment, et de manière dynamique dans les cas où le typage statique est insuffisant. En associant des structures DML aux templates de la transformation, il devient possible de valider les parties d'un document par rapport aux parties de différents modèles.
De plus, grâce à l'intégration de DML, le mécanisme de sélection des nœuds peut être grandement enrichi. En effet, en associant, à chaque étape de la sélection, des structures DML décrivant le contenu d'un élément ou des types DML définissant les expressions régulières des nœuds textes, il devient possible de sélectionner les nœuds non seulement par rapport à leur genre mais également par rapport à leur contenu ou type. Ainsi le mécanisme de sélection devient très riche et permet de définir des transformations de plus en plus précises. Les structures DML associées à la sélection peuvent de plus être dynamiques, c'est-à-dire évolutives au fil de la transformation, suivant le contexte et les nœuds de l'input rencontrés.
Il faut encore souligner que le langage DGL est totalement et formellement spécifié, grâce la sémantique dénotationnelle. Ceci rend son implémentation très rigoureuse et permet de comprendre et d'appréhender le langage de manière plus saine.
Grâce à la simplicité de la version initiale de DGL ainsi qu'à l'intégration naturelle de DGL et de DML, il est possible de définir des transformations génériques qui permettent d'exécuter de manière automatique des tâches répétitives telles que celle décrite dans la prochaine partie.

6 Traduction de documents : une application intégrée

L'application que nous présentons est, parmi d'autres, un bon exemple de la puissance des transformations génériques réalisables grâce au trio YML-DML-DGL. Si, dans cet exemple, ce sont plutôt les concepts généraux de l'intégration qui sont mis en avant, d'autres applications comme la génération automatique de documents à partir de n'importe

quel modèle mettent en avant l'intégration concrète de DML et DGL. Elles ne sont toutefois pas présentées ici.
Cette application simple et automatique concerne donc la traduction de documents : si plusieurs modèles, qui possèdent d'une part une structure similaire et d'autre part des balises dans différentes langues sont disponibles dans le système, alors il est possible d'écrire une transformation DGL unique et générique qui prend en entrée deux modèles, le premier rédigé la langue source et le second dans la langue cible, et produit la transformation spécifique qui pour chaque document YML de la langue source produit le document traduit. La figure 1 illustre cette application pour un exemple simple de carnet d'adresses. Le modèle existe en anglais (source) et en français (cible). La traduction générique (*translation.dgl*) est écrite par l'utilisateur et la traduction spécialisée (*english-french.dgl*) est automatiquement produite par la traduction générique.

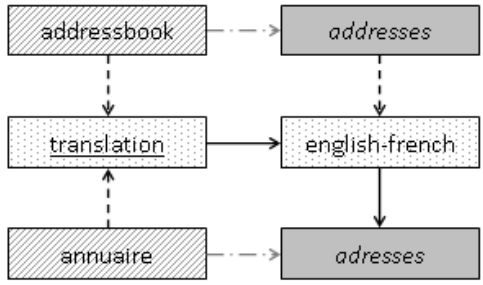

Figure 1. Traduction générique de documents

De manière plus concrète, la transformation générique produit pour chaque élément du modèle source une règle qui sélectionne ces éléments et crée de nouveaux éléments dont le nom est issu du modèle cible. Puisque la structure générale des modèles est la même, l'appariement entre les éléments est évident. L'information en tant que telle est contenue la plupart du temps, et c'est le cas dans cet exemple, dans les nœuds textes. Ceux-ci sont dès lors simplement copiés à des positions et niveaux équivalents. De plus, comme la transformation est générique, les modèles source et cible peuvent être interchangés sans que la transformation ne doive être modifiée. La transformation spécialisée ainsi générée peut être appliqué à n'importe quel document valide par rapport au modèle source pour être traduit.
Cette application montre la robustesse et les potentialités de l'intégration des langages. Appréhender ces langages sous la contrainte d'une intégration forte permet donc d'envisager une évolution positive de la gestion des documents semi-structurés.

7 Conclusion et perspectives

Dans cet article, nous avons présenté une étude qui vise à intégrer les trois langages principaux pour le traitement des documents semi-structurés. Afin d'assurer des fondations solides à ce projet, nous proposons trois nouveaux langages afin d'éviter les potentielles faiblesses des langages existants et de garantir que ces langages se focalisent avant tout sur une future intégration. YML pour le balisage, DML pour la modélisation et DGL pour la génération et la transformation de documents visent ce but. Chacun de ces trois langages a été développé de manière à s'intégrer avec les autres pour proposer un traitement rigoureux des documents semi-structurés.
Le système a été complètement développé en Java. Sur la base de la sémantique formelle complète développée pour soutenir les concepts de DGL mais également de certaines parties du DML telles que le mécanisme d'héritage, l'implémentation des outils s'est révélée rigoureuse et sûre.
Nous avons brièvement introduit une application intégrée qui montre comment les instances des différents langages sont liées les unes aux autres dans le langage. D'autres applications existent qui démontrent les potentialités de l'intégration notamment en matière de généricité. Parmi elles, une application de génération automatique de documents permet de produire soit le document minimal correspondant à tout modèle, soit une transformation capable de générer ce document minimal.
L'intégration peut encore être renforcée afin d'améliorer le traitement des documents semi-structurés, qui, au vu de l'évolution du Web et de la transmission de l'information, reste un défi majeur du monde informatique. Ainsi, la validation des documents YML pourrait être traitée par des transformations DGL qui produiraient un document YML certifiant la validité d'un autre document. Une telle transformation existe déjà, elle permet de produire pour un certain nombre de modèles répondant à des restrictions encore importantes des transformations validant les documents associés. L'intégration de langages si proches dans la gestion de documents est ainsi, et au vu des premiers exemples produits et analysée, tout à fait pertinente.

8 Références bibliographiques

[1] T. Bray, J. Paoli, C.M. Sperberg-Mcquenn, E. Maler et F. Yergeau. Extensible Markup Language (XML) 1.0 (Fifth Edition). W3C, 2008 http://www.w3.org/TR/2008/REC-xml-20081126

[2] J. Clark. XSL Transformation Version 1.0. W3C, 1999 http://www.w3.org/TR/XSLT

[3] J. Clark et S. DeRose. XML Path Language (XPath) Version 1.0. W3C, 1999
http://www.w3.org/TR/XPath

[4] J. Clark et M. Murata. RelaxNG Specification, 2001.
http://www.relaxng.org/spec-20011203.html

[5] D. Connolly. The Evolution of Web Documents. xml.com, 1997
http://www.xml.com/pub/a/w3j/s3.connolly.html

[6] D.C. Fallside, P. Walmsley. XML Schema Part 0: Primer. W3C, 2004
http://www.w3.org/TR/xml-schema-0

[7] H. Hosoya, B.C Pierce. XDuce: A statically typed XML processing language. ACM Transactions Internet Technologies, 2003.

[8] M. Kay. XSL Transformation Version 2.0. W3C, 2007.
http://www.w3.org/TR/xslt20

[9] C. Kirkegaard, A. Moeller. Type Checking with XML Schema in XACT. Proceedings of the Workshop on Programming Language Technologies for XML (PLAN-X), 2006.

[10] M. Murata et al. Taxonomy of XML schema languages using formal language theory. ACM Transactions Internet Technologies, 2005

[11] C. Pugin et R. Ingold. YML : une version épurée de XML pour faciliter une spécification rigoureuse des modèles de documents et des transformations. Actes du Colloque International sur le Document Electronique 9 (CIDE9), Fribourg (Suisse) 2006

[12] C. Pugin et R. Ingold. Combination of Schema and Transformation Language Described by a Complete Formal Semantics. Proceedings of ACM Symposium on Document Engineering (DocEng07), Winnipeg (Canada) 2007

[13] C. Pugin et R. Ingold. Instances, modèles et transformations: tout en un. Actes du Colloque International sur le Document Electronique 11 (CIDE11), Rouen (France) 2008

[14] R.D. Tennent. The Denotational Semantics of Programming Languages. Communication of the ACM, vol.19, n.8, 1976.

[15] A. Tozawa. Towards static type checking for XSLT. Proceedings of ACM Symposium on Document Engineering (DocEng01), Atlanta, GA, 2001.