TransM: A structured document transformation model

Nouhad Amaneddine * Jean-Paul Bahsoun Jean-Paul Bodeveix

Institut de Recherche en Informatique de Toulouse
Université de Paul Sabatier, Toulouse, France
{amaneddi, bahsoun, bodeveix}@irit.fr

Abstract: We present in this paper a transformation model for structured documents. TransM is a new model that deals with specified documents, where the structure conforms to a predefined data type. In particular TransM works on XML document conforming to a document type definition DTD. We present the different components of our model and we put the point on the kernel part of TransM which is the transformation rules. We show the strengths and the weaknesses of some recent transformation approaches and we prove that TransM can perform not only those simple and direct transformations but also complicated ones, especially, those who handle recursive appearance in the structure of the modeled documents, where elements may appear at any level of deepness in the hierarchical composition of the document structure, whilst the problem is not clearly resolved in the existing transformation models. We propose TransM as a general model for structured document transformation and we explore in more details its specification core that carries out document transformations.

1 Introduction

Nowadays, documents are produced in electronic forms and they are often displayed using different applications. Moreover, in the last recent years, the development of software engineering intended to separate the document structure from the document presentation [W303b][W303b].

Sharing documents between different systems is a very useful process in computer application. In some cases, a part of the document concerning particular information is needed by one application where the whole content is manipulated by another one [CI03][MR97]. Furthermore, document transformation shows an increasing attention of many research teems in the last few years [XF01][Pe03][Wi03]. Consequently, a dedicated language for specifying transformations should be defined. It may contain transformation rules that describe relationships between a source structure and a destination structure, and which can be used to generate a target structured document instance conforming to a predefined document type. The source and the destination model could be instances of the same document structure definition, or, they may conform to two different document structures.

*Lebanese university, faculty of Sciences

Structured document transformations relies on several modules. The core starts at the transformation language based on rules or expressions. Then strategies for the traversal of the input document must be defined. Advanced conditions and constraints may be applied in order to filter selected information. Finally the writing policy is used to place correspondent elements in the output structure. Some models utilize their own definition languages to carry out the transformations [OM02], some other prefer to generate code for specialized language rather than using a general one [Pe03]. Without forgetting the data manipulation and the application of particular functions on document content in case of needs. Consequently, many modules are used in order to accomplish all the steps of the transformation process.

The chosen strategy is built according to the selected document structure. As XML [W303b] became the standard markup language for structured document, a tree structure is preferred as the backbone structure for documents. Even if it has a tree form, an XML document can be considered as a graph. We have in XML the possibility to represent graphs. This can be performed by referring the destination node with the attribute of the source one. Eventually, a unique identity should be chosen for each node in the document thus it can be referred by its identity. This paper presents a new model for structured documents transformations, named TransM. We show the specification of the kernel part of TransM which is the transformation rules and we present the general algorithm that holds the transformation process. The paper is organized as follows: in section two we present an overview on structured document and their frameworks. We detailed TransM in section three. We present in section four some recent related works then we conclude in section five by a perspective for our future work.

2 Structured documents and transformation overview

A document is considered to be structured when it contains implicitly or explicitly additional information about its hierarchical composition [AB03][XF01]. Many types of documents are considered as structured ones, as those applying pre-defined grammars that can be a context free grammar or any other type of grammars [Mu98]. The structure of the document is the backbone of its internal representation. It is the hidden side of the external representation that is created conforming to the internal one and it is presented to the user of the application. The most important representation of a structured document is a representation that contains only the actual content of the document in the structured form. Structured document can be a document with a tree structure or a graph one, or it can be a document conforming to some constraints defined in another form than a grammar. XML documents conforming to some Document Type Definition are such examples.

When we specify the transformation, we indicate how the rules should be applied and what they do mean. Direct transformation are made when we have a simple transformation process and we suppose that we will not use the same transformation another time. Otherwise it is more wise to define the rules that we need in order to gain from the reusabil-

ity characteristic of the specifications. The recent research results are focusing on model transformation [Wi03][PR02]. We remark that they differ in the literature between document transformations and model transformations, while the two research areas can be placed under one single title which is the structured document transformation. The QVT approach [OM02] is one of those proposals that works on model transformation. For us, while we can represent any object to be transformed in a structural form, the resolution of the transformation will count on the general axis defined by the transformation of documents.

The main problem that we could mention in the used transformation method is that most all of those methods are efficient when the transformation process is not complicated. This case can be seen repeatedly in those transformations where both the input and output structures are so closed and when there are no manipulations to be performed on the transformed information. In this case, simple transformations are sufficient and we see useful rules and efficient schemas with a very easy and comprehensive way are used to accomplish the transformation. The problem starts when the structures of both input and output documents are quite different and when we need to make changes on the contents in order to produce the required document. In this case, we found either the used method can hold those transformations but they are too complicated to be understood and applied, and their use does not cover all the complicated situation we could face in such transformation, or, those methods are unable to support advanced transformations. In this case, the eventual question that could be asked is what is the interest of providing such a method and how relevant is its application, especially for a delicate transformation process. We present in the next section the TransM model that can support not only those direct and simple transformations but also more complicated ones.

3 Core of TransM transformation model

Before exploring our transformation rules which represent the kernel of the TransM, we present the eXtensible Style Sheet Translation language. We show the different steps of the transformation process then we introduce our model justifying the choice of generating XSLT code.

3.1 XSLT

The W3C proposed XSLT as a transformation language for XML documents [W303b]. This language has considerable computation power. A key element of XSLT is the sublanguage of patterns, which is used to match and select elements. The pattern language of XSLT has evolved into Xpath [W303b], a language for selecting nodes of a tree. It performs the core functions of XSLT. XSLT is a recursive XML transformation language

and an XSLT program can be thought of as an ordered collection of templates. Each template has an associated pattern and contains a nested set of construction rules. A template processes nodes that match the selected pattern and constructs output according to the construction rules. The transformation starts at the root of the input document and the construction rules specify, by means of construction patterns, where in the XML document the transformation process should continue. However, XSLT requires detailed and tedious programming to accomplish complex structure transformations. As alternative, we construct our transformation rules at the structure level. An XSLT program will be generated automatically in order to perform transformations on the instance level.

3.2 Transformation process

The process consists of transforming XML documents conforming to a predefined document type definition to another XML document conforming to a different document type definition. It starts by making the association relations between the input and the output structures as shown in figure 1. Those associations are chosen from a set of pre-defined possible associations. Then the transformations are specified by a set of rules. Those rules are used to hold the required transformations in the concrete level. The XSLT program generated after the transformation rules aimed to transform document conforming to the input structure to another document conforming to the output structure.

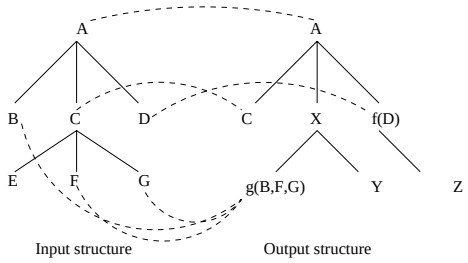

Figure 1: Elements Associations

In this figure we show the input structure and the output structure with a tree forms. Associations are represented by the dashed lines that link nodes from the input structure to nodes of the output one. The association may be simple, which means that it represents a direct copy of the node content, or it may be more complicated and expressed as a function of many variables depending on more than one node in the input structure.

We will not explore the details of all the parts of our model. We present the core part of the transformation system we are working on, which is the specification of the transformation rules that are made according to the chosen user associations. The TransM model and its different components are presented in figure 2.

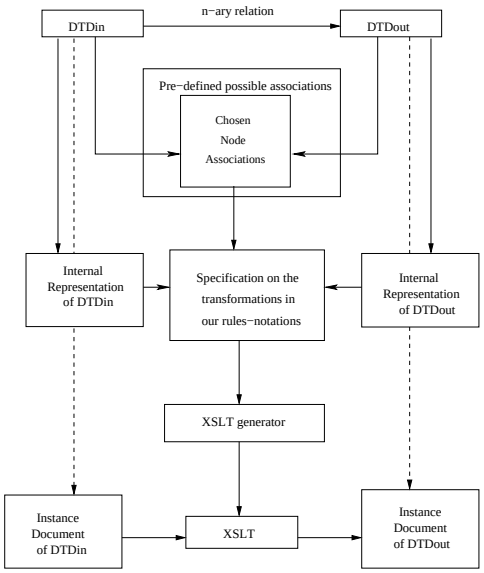

Figure 2: TransM, a structured document transformation model

3.3 Transformation rules

The main idea that our model relies on is the definition of transformation rules that may hold not only those simple transformations but also complicated ones. Thus the rules that hold the transformations should satisfy this constraint.

A rule of the rule database is selected if its left hand side matches the current node of the source document, and the associated constraint is satisfied. The new document is obtained by evaluating the right hand side. If no rule apply, the same document is traversed (in a top-bottom, left to right order) until a node can trigger of a rule. The subtree is then transformed according to the corresponding rules.

The right hand side defines the image of the current node. It may recursively call other transformation rules in order to build the image tree. In the same way, the right hand side is composed of two different parts: the first one consists in providing the position where the filtered data should be placed and the second part manipulates the content of the selected information.

3.3.1 Rules syntax and semantics

Our method consists of a compromise solution between relative-path based method and absolute-path based ones. We use an additional strategy that allows us to specify a set of rules for a particular type of document, depending on the type of the transformation. Also, we can control the selection and the writing of the element in any level of deepness.

The transformation is expressed as a set of : ***filter*** → ***expression*** production rules. We named the left hand side of the rule by ***filter*** and the right hand side by ***expression***. The filter may refer to any part of the whole document, so that, rules may be context dependant.

In order to simplify the understanding of the production rules definition we show by a list of cases how to express specific transformations. Each case represents a general transformation process. Because of the space limitations, we could not include in this paper the complete formal definition of the production rules neither the general transformation algorithm.

Selection constraints: This kind of constraints are usually applied to the left hand side of the transformation rule, here frequently-used sample cases:

1. A simple copy: Source_element → *Target_element*. All occurrences of the "Source_element" tag are replaced by "Target_element".
2. Element with a specific child: Source_element[specific_child] → *Target_element*. All occurrences of the "Source_element" tag having a child named "specific_child" are replaced by "Target_element".
   Here, we have applied the **filter** →**filter**"["**constraint**"]" production rule.
3. More constraints on children Element_A[(Element_B/Element_C)+, Element_D] → *Target_element*. This rule selects Element_A elements that have a non empty sequence of Element_B children with Element_C child, followed by Element_D child.
4. Constraint on a sequence of elements and on element's occurrence:
   Element_D[A+, B*, C] → *Target_element*. This rule identifies Element_D elements that have a list of one or more A child followed by zero or more B child that are followed by a single C child.
5. Constraint on attributes Element_A[@**attr**] → *Target_element*. Here we select Element_A elements which have an attribute called **attr**
6. Selecting elements with specific attribute value: Element_A(@attr="v1")→ *Target_element*. This rule identifies Element_A elements having an attribute named attr with v1 value.
7. More conditions on attributes: Element_A[!@attr]→ *Target_element*. Here we identify Element_A elements that do not have an attribute named attr.
8. Condition on elements' content:Element_A[B="text1" & C/D="text2"]→ *Target_element*. This rule selects Element_A elements having the string "text1" as a text content of its element B child and the string "text2" as a text content of the child D of the Element_A's child C.

9. Constraint with absolute path: Element_A[/C/D@attr]→ *Target_element*. This rule selects Element_A elements if the root of the structure that contain A has a grand child named D. Moreover, this grand child possesses an attribute called attr.
10. Constraint with relative path: Element_A[C/D@attr]→ *Target_element*. This rule selects Element_A element if its grand child D has an attribute named attr.

Production constraints: Assigning elements of the input document to variables In the selection constraint sample cases, we have considered the selection criteria in the left hand side of the production rule. This part of the rule may wrap a filtering conditions or constraints. We represented the right hand side as a simple *Target_element*. The following examples reveal how to manipulate output data in the right hand side of the production rule. The result of the production will eventually depend on the selected information. The left hand side will write down data selected with respect of the left hand side filtering.

1. Matching attribute values: Element_A(@att1=x', @att2=y', @att3=z') → Target_element(@at1=x'+y', @at2= z'-x'). This rule selects Element_A elements which have three attributes named att1, att2 and att3. It recuperates the values of those attributes in x', y' and z' variables then it uses them when producing output element.
2. Filtering by Using end-values: Element_A(@att1="value1", @att2="value2") → Target_element(@at1="value3", at2="value4"). This rule matches the Element_A elements that have the first attribute value equal to "value1" and the second attribute value equal to "value2" then write the output node with different static values for its attributes.
3. Filters composition:
   Element_A(@att1="value1", @att2=x')(Element_B(@att1=y')) →
   Element_C(@at1="value2", @at2=x'+y'). This rule selects Elements_A elements having two attributes, att1 with value "value1" and att2, and one child, Element_B with attribue att1. The values of att1 and att2 are assigned to the variables x' and y'. The right hand side creates the Element_C element with two attribute, the first one is called at1 and values "value2" the second one is called at2 and values x'+y'.

Also, filters with variable assignment can be made on the tag name, the attribute list and the children list. As same, additional constraints can be applied on the presence of a particular attribute with a given value, a presence of a child, of a descendant one, a value of a child, a sibling and on ancestor. The selection we use enables us to navigate over all the structure of the document. Form current node in the hierarchical document, we can select a node using criteria depending on any node of the document. The context dependent filtering mechanism ca access ancestors, children sibling or the root of the document. The algorithm processes in an descendent way from the root down to the leaves of the document [YS99]. Linking the two neighbor elements in the selection part is not enforced, but it can be done according to the needs imposed by the type of the transformation we want. In

other words, we can refer to a node directly by the node name or by presenting the parent followed by a slash then the required node, this if we need to add constraints concerning this particular element. Filters and expression start always by the main pattern which is the element identifier. The other parts of both left and right hand side of the production rule consist of condition criteria and selection tools as same as the writing paradigm in the right hand side.

3.4 Example: Image transformation

Consider the example of transforming an image stored as an XML file that conforms to a predefined document type definition DTD to another image that conforms to the structure imposed by another DTD. We assume that the basic elements that compose the image are rectangles, triangles and circles. Each element has its own properties that are represented as attributes in the corresponding DTD.

The choice of this particular example is not to show how to model graphics in XML. In fact, XML possesses the Scalable Vector Graphics language (SVG) [W303a]. This is a modularized language for describing two dimensional vector and mixed vector/raster graphics in XML. We chose the DTD of an image because of its recursive nature. Our goal is to show how to use our rules to transform documents where recursive elements may appear.

In this example we intend to change the color of red rectangles into green ones and the blue circles in green ones, furthermore, we want to apply translation on the circles following the horizontal coordinate. The X coordinate of the center of the target circle equals the sum of the coordinates of the center of the source circle . Vertical positions of the source center and the target one are the same and the radius of both circles are equal.

The DTD we chose to define the structure of the XML document that represents the image is as follows:

```
<!ELEMENT image (rectangle|circle|image)* >
<!ELEMENT rectangle (EMPTY)>
<!ELEMENT circle (EMPTY)>
<!ATTLIST rectangle
            color CDATA #IMPLIED
            length CDATA #IMPLIED
            width CDATA #IMPLIED
           >
<!ATTLIST circle
            color CDATA #IMPLIED
            radius CDATA #IMPLIED
            x_center CDATA #IMPLIED
            y_center CDATA #IMPLIED
            >
```

This DTD shows that a recursive appearance of a graphical element is possible, a rectangle can include a circle that can include in its turn another node of the same type rectangle. We suppose that the target structure has the same DTD, with different node names.

To apply the above-mentioned transformation, we use the following rule that conforms to the general syntax of rule definitions :

Rule TransImage{
image → target_image;
rectangle (@color = "red", @length=x', @width=y') → target_rectangle (@color = "green", @length=x', @width=y');
circle (@color = "blue" , @radius = r', @x_center = x', @y_center = y') →
target_circle(@color = "green", @radius = r', @x_center=x'+y', @y_center = y');
}

Consider the following XML document that conforms to the DTD mentioned above:

Input XML:

```
<image>
        <rectangle color = "red" length = "34" width = "56" />
        <image>
                <circle color = "blue" radius = "9" x_center="8" y_center = "6" />
                <circle color = "green" radius = "5" x_center ="82" y_center = "16" />
        </image>
        <circle/>
</image>
```

The application of the rule TransImage on the instance document produces the following XML document:

Output XML

```
<target_image>
        <target_rectangle color = "green" length = "34" width = "56" />
        <targe_image>
            <target_circle color = "green" radius = "9" x_center="14" y_center = "6"/>
        </target_image>
</targe_image>
```

It is shown that the lines corresponding to the green circle and the line corresponding to the empty circle are omitted in the output document. This is because there is no implicit rules for circles that are not blue in the input documents. If we want to copy all the circles we should add an explicit rule with conditions that differ from those used to select other

types of circles. The existence of two different rules for the same element does not cause any problem, since conditions imposed in those two rules and for the same element are different. In fact, we do not need many rules since we can compose constraints in the rule by using logical operators. Thus complicated filtering could be made by using logical expression in the constraint part of the rule.

Because of the possible recursive appearance of the elements in the input and the output structures, it is difficult to express the associations only graphically. We need textual forms to convey such associations. Graphical associations are convenient to be utilized in the instance level. In this case, rules could not be used as specifications and they should be applied locally. This is what happens in add-hoc transformation application where transformations are local ones. This is not the case in our model where we define rules on the abstract level in order to be reused everywhere we need the transformations and for any document instance.

4 Recent approaches

Recently, different methods have been defined for carrying out document transformations. Some of them work directly on the basic level of the document which is its content [Jo02]. Therefore the transformation is performed on the data level. Some others specify the transformation rules and apply the transformation process with respect to the specifications [BBG01]. Other research groups are working on the same problem but with a different form, like introducing the problem as model transformation. In this case the power of UML to represent and manipulate models is used [Wi03]. On the other hand, we can differ from those textual method and graphical ones, some approaches consider all the needed steps concerning the transformation process can be performed by graphical method, in other words, by using geometric notations to express transformation procedures. Thus, they use a full graphical way to hold the transformation between different applications or different models [Wi03].

We present in a brief way some of the recent transformation approaches. We explain for each approach its relevance and its behavior, as well as the framework of its application.

4.1 OMG' QVT

Models are the primary artifacts in the OMG's model driven architecture software development approach [OM03]. MDA made a significant difference from earlier uses of modeling languages such as OMG's UML, in which the primary purpose of models was to aid understanding and communication.

In MDA, transformations play a key role, a standard syntax and execution semantics for transformation is an important enabler for an open MDA tools chain. In [OM02] the OMG

issued a revised Request for Proposal for MOF 2.0 Query, views and Transformations to address a technology part of the OMG Meta Object Facility entering to the main issues in the manipulation of the MOF models. The object management group has issued a Request for Proposal for Query/views/Transformation (QVT) language to exploit the Meta-Object Facility that share common core concepts with the Unified Modeling Language (UML).

The proposed pattern language of QVT is not always the best way for expressing aspects of a particular transformation. The differentiation between relation and mapping cause a wondering confuse. The relations are not clearly the specifications of the mappings. This is obviously shown when some of the mapping rules are kept as they are in their definition in the relations part.

The second problem appears in those complicated transformations, especially when inheritance sub-models appear in the hierarchical architecture of the manipulated model. This problem is difficult to be seen when a graphical representation of the model or the structured document is shown.

4.2 MTrans

MTrans [PR02] is a transformation-specific language developed at France Telecom. Dedicated languages or domain-specific languages (DSL) are programming languages or executable specification languages that offer, through appropriate notations and abstractions, expressive power focused on, and usually restricted to a particular problem domain.

The use of a domain-specific language may be a solution of some problems related to the software development (reusability, productivity, maintainability). [Wi03] demonstrates that dedicated languages can reduce the cost of software maintenance. Other studies [Kr92] present those languages as one of the main solutions to satisfy software reusability.

MTrans aims to supply a general framework for expressing model transformation. It is based on a meta-modeling approach and contains a language and an environment to write model transformation. It uses a finite set of instructions, and it is used to transform only MOF compliant models.

Before the last version of the MTrans framework, MTrans rules were translated into XSLT code [Pe03]. Then an XSLT processor was used to transform an XMI document that represents the source model into another XMI document representing the destination model. Because of some difficulties faced in manipulating and using the extensible style sheet translations, the research group has changed the generated language that carries out the transformation, and they use Python instead of XSLT.

One of the problems that faces MTrans is the fact that it transforms only those entities that can appear in a flat meta-model. Flat meta-model is a meta model that contains only concepts, which means that it is a meta model that defines the set of terminal instances [PZB00]. By example, if we have in the source meta-model an inheritance tree structure, only leafs are interesting for transformations. Mtrans does not support transformation of

recursive elements. Rules are not named so it is not possible to call a rule in order to apply it from other rules. Another disadvantage of this framework is that the used language seems to be able to do many manipulations on transformations but it is not clear how it does them.

4.3 UMLX

UMLX provides an open source tool to support the OMG's Model Driven Architecture initiative [OM03]. It describes a primarily graphical transformation language that extends UML through the use of a transformation diagram to define how an input model is to be transformed to an output model.

UMLX uses standard UML class diagrams to define information schema and their instances. It extends the class diagram to define inter schema transformations. Four main extensions have been added to the class diagram to support model transformation [Wi03]. The first one is a graphical representation of an invocation which is used when transferring a schema syntax to another schema syntax. Two other graphical design were added to distinguish the input model from the output one. We think that even if those additional graphical tools may intend to enrich the used graphical language, it is obvious that some of them are irrelevant representations. Since the input model and the output one are clearly known especially when arrows are used in the schema that represent the two models and the transformation. Additional geometrical objects are used to clearly reveal input and output models.

A problem appears in using the UMLX transformation which is the incapability of modeling recursive instances of both source and destination models especially when the depth of the modeled entity is not known. For known depth relations we need to pass an outer context down as an inner context is explored. This can be seen in the resolution of primary key in the UMLX2RDBMS example shown in [Wi03].

Graphical notations can get cumbersome for strange complicated relationships, but seem simpler for practical simple graphs. The graphics is a less explored area, so it takes a while to learn the new idioms, and possibly to provide the correct family of helper transformations/syntax extensions.

4.4 Other transformation approaches

Many approaches attempted to perform structured document transformation. The difficulties appear each time the transformation become complicated and advanced manipulation on the selected information are needed to be applied. Some of those approaches are based on syntax directed translation (SDT), others have added extensions to the SDT method to resolve more critical contexts (ESDT), some others are based on pattern matching [Ho04][IN04]. On the other hand, some transformation models consider as essential

the backbone structure of the modeled document: they build formal definitions and they apply the transformation through a syntax that conforms to the predefined specifications. The tree automaton based approach [Mu98] and the filter based approach are such examples. The efficiency of the used methods that carry out the transformation remains a serious problem, as same as the complexity of the adopted algorithms. Some research context explored transformation process approaches in more details [AB03]. They explained for each one its relevance and its behavior as well as the framework of its application. The problem remains always with non-direct transformations and when the structures of the input model and the output one are not alike. In this case, additional manipulation should be performed in the transformation process and it is not clear how to solve such transformation in those existing approaches.

5 Conclusion and future work

Prior to constructing our model, we examined several approaches for specifying transformation for structured documents. None of them seemed suitable as a specification model for complex transformations. Some are too operational in natures and other can describe local transformations only. We presented in this paper our model for transforming structured document .We have shown its core part which is the transformation specifications in term of rules in more details. The XSLT language is suitable to transform an XML document into another one, but it is hard and error prone to manually write XSLT programs. Our formalism allows expressing transformation rules in a more succinct and readable way. This model can be used not only for transforming structured document, but also for model transformation. The main requirement will be the representation of the input and output meta model as DTD and their model instances as XML documents conforming to the defined DTD. The difference with the model transformation is that we are working on two levels: the abstract level and the concrete level. In the model transformation, the OMG community has proposed the UML modeling language as model representation. It intends to affront the incompatibility of having a variety of meta models the OMG has projected a general structure for meta model integration. This organization conducted to a four level architecture: the meta-meta model level, the meta model level, the model level and the data level. In this architecture, each level preserves an instantiation relation with its superior model. We aim to study possible extensions on our rule definition. We intend to study particular transformation in case we find in our tests that the rule specification does not fit to handle a particular type of transformation. Therefore, and as the architecture of our model permits, our ongoing research will continue to study various potential optimizations to incorporate into our rule specification and the template generating algorithm. We intend to develop the core part of the XSLT generator. This part will be transparent for the end users. It may relies on transducer, or we may use a parsing algorithm that transfers directly our code into XSLT. We may use regular grammars for code transformation. We are searching a practical and efficient method to generate extensible style sheet translation without any lost of the semantic handled by our transformation rules.

References

[AB03] Amaneddine, N. and Badr, Y. A taxonomy of transformation methods for structured document. Proceeding of the second International Conference on Computer Systems and Applications, ACS/IEEE, Tunisia, ISBN 07803779837. Library of Congress: 2003106612. July 2003.

[BBG01] Banerji, A., Bartolini, C., and Ger, D. Web services conversation language wscl 1.0. `http://www.e-speak.hp.com/media/wscl_5_16_01.pdf`. 2001.

[CI03] CIDX. The chemical industry data exchange. `http://www.cidx.org`. September 2003.

[Ho04] Hosoya, H. Regular expression filters for xml. In Programming Languages Technologies for XML (PLAN-X), Venice, Italy. January 2004.

[IN04] INRIA, F. The cduce language. `http://www.cduce.org/`. 2004.

[Jo02] Jones, D. Translating c to ada. `http://www.knosof.co.uk/ctoa.html` Knowledge Software Ltd, Hants, UK. 2002.

[Kr92] Krueger, C. Software reuse. in ACM computing survey. 1992.

[MR97] Murray-Rust, P. Chemical markup language. World Wide Web journal, 135-147,`http://xml-cml.org`. 1997.

[Mu98] Murata, M. Data model for document transformation and assembly (extended abstract). In Principle on Digital Document Processing, pages 140-152. 1998.

[OM02] OMG. Qvt. Initial submission for the MOF Query/views/Transformation, DSTC International Business Machines. 2002.

[OM03] OMG. Omg. The Object Management Group, `www.omg.org`. 2003.

[Pe03] Peltier, M. Techniques de transformation de modèles basé sur la méta-modélisation. Ph.D dissertation, University of Nantes. September 2003.

[PR02] Peltier, M. and R&D, F. T. ransformation entre un profil uml et un méta-modèle mof. In Langage et modèles à objets LMO. ISBN: 2-7261-1131-9. 2002.

[PZB00] Peltier, M., Ziserman, F., and Bezivin, J. On levels of model transformation. In XML Europe conference, Paris. 2000.

[W303a] W3C. Scalable vector graphics (svg) 1.1 specification. `http://www.w3.org/TR/SVG/`. 2003.

[W303b] W3C. The world wide web consortium. `http://www.w3.org/XML/`. 2003.

[Wi03] Willink, E. Umlx: A graphical transformation language for mda. In MDAFA03, Workshop on model driven architecture foundation and application, Enschede, The Netherlands. 2003.

[XF01] X.Tang and F.W.Tompa. Specifying transformations for structured documents. Proceeding of the 4th International Workshop on the Web and Databases (WebDB'2001). May 2001.

[YS99] Yamasaki, K. and Sodeshima, Y. A comparison of bottom-up pushdown tree transducers and top-down pushdown tree transducers. Department of Information Sciences, Faculty of Science and Technology, University of Tokyo 2641 Japan. 1999.