A Model for Complex Tree Integration Tasks

Marcin Maleszka and Ngoc Thanh Nguyen

Institute of Informatics, Wroclaw University of Technology, Wybrzeze Wyspianskiego 27, 50370 Wroclaw, Poland
{marcin.maleszka,ngoc-thanh.nguyen}@pwr.wroc.pl

Abstract. The common approach to integrating XML documents is based on existing formal structures, not originally designed to integration tasks. In this paper we propose a Complex Tree model designed from the beginning to integration tasks, capable of representing most tree structures. The Complex Tree model is defined on both Schema and Instance level, to better work in practical situations. The integration task for Complex Trees is also defined on both levels. A set of explicitly stated criteria for integration is given, to better design future integration algorithms, in respect of the desired aim of integration process. Finally a simple integration algorithm is presented, based on selected criteria.

Keywords: XML integration, complex tree, integration task, integration criteria.

1 Introduction

The XML file format and it derivatives are today the *de facto* standard of data and knowledge storage [14]. The XML itself has evolved, moving from its first schema – DTD – to more modern approaches, like XSD, DSD and more [5]. Each of those solutions was designed to make the XML (both on schema and instance level) more robust and more practical. Those structures are therefore the basis of many enterprises [3].

The same enterprises occasionally need to integrate those specifically designed structures, i.e. in e-business [3], thus making it necessary to develop tools to automate the process. Various tools for this task exist, each develop for a specific practical, or theoretical purpose [9][13]. Not many of those however are based on the integration theory, most arising from the practical need and only using excerpts of the theory.

This paper is based on an alternate approach. First, a structure allowing for defining various hierarchical structures is defined, with accordance to the integration theory and an integration task is defined. Then criteria are laid down for the integration task, based on the practical approaches (where the criteria are not explicitly stated) and on the theory (where the criteria are too general). The paper concludes with a simple integration algorithm, with more to follow in the future work of the authors.

This paper is organized as follows: Section 2 contains a short survey of existing integration approaches, both for practically used XML and other hierarchical structures; Section 3 provides a model of a hierarchical structure (tree) used in this

N.T. Nguyen, C.-G. Kim, and A. Janiak (Eds.): ACIIDS 2011, LNAI 6591, pp. 36–46, 2011.

paper, as well as a definition of an integration task for this model; Section 4 gives an overview of various simple integration criteria; Section 5 provides a simple integration algorithm based on the definitions of previous sections; the paper concludes with a discussion on future work to be done in this area.

2 Related Works

Some of the chronologically oldest approaches to hierarchical structures (trees) integration may be derived from the works of evolutionary biologists seeking methods to integrate differing results from multiple data sources (integrating different evolutionary trees). The structures they operated on in the 1980s and later are *n*-trees, trees with labeled only leaves. Different approaches created basing on general integration theory used some additional structures for integration, like clusters [7] and their variations [4][15], so-called Maximum Agreement SubTrees [10] or triads [1]. Each of those methods, while well developed, was useful only for the narrow field it was designed for.

For fully labeled trees, an area much closer to practical XML documents, some work was done on integrating multilevel classification trees [6]. There some problems known from integration theory, like knowledge inconsistency and the need to determine a consensus, became more visible.

The main area of work in integrating hierarchical structures was done in the most practical area, that is the integration of XML documents and their schemas (general hierarchical schemas, DTD schemas, XSD schemas and more). These approaches vary from using a specific tree grammar [2] to using graph representation [11]. The latter may be based on other sub-representations, like path-based approaches used in XML search engines [8] (there the aim of integration is for the final structure to be able to provide the same answer for a specific path query, thus all paths must remain unchanged during the process).

In their survey on automatic schema matching Rahm and Bernstein [13] detail solutions used also when integrating hierarchical schemas (XML and DTD). The classification they provide shows examples of using single elements or whole sub-structures for matching, matching according to cardinality or some auxiliary information about the element. These are in fact implicitly stated criteria for integration – as defined in section 4.

3 Hierarchical Structures Integration

In this section we will introduce the hierarchical structure model designed specifically for integration purposes, which is capable of representing different tree structures (XML, its schemas, n-trees, some ontologies, etc.) – the Complex Tree. First, the general description of the structure will be presented. In 3.2, the distinction between Complex Tree Schema and Instance will be drawn. In the rest of this section integration tasks will be defined for both structures.

3.1 Complex Tree

For purposes of this work, a model of hierarchical structure (called Complex Tree) has been defined. This model provides the ability to represent most practically used structures (XML, its schemas, n-trees, some ontologies), while retaining relatively simple for integration purposes. It also allows representation of both schema and instance level structures with virtually no changes in structure.

Definition 1. Complex Tree (CT) is a tuple

$$CT = (T, S, V, E)\ , \tag{1}$$

where:

T – a set of node types (i.e.: *root, leaf*)

$$T = \{t_1, t_2, t_3, \dots\}\ , \tag{2}$$

S – a function defining the number and type of attributes for each type of nodes (i.e. nodes of the type *root* always have only 1 attribute *date_created*)

$$S(t_i) = (a_{i1}, a_{i2}, \dots, a_{in_i})\ , \tag{3}$$

V – a set of nodes (vertices), where each node is a triple

$$V = (l, t, A)\ , \tag{4}$$

in which the consecutive elements represent the label of the node, the type of the node ($t \epsilon T$) and the set of attributes and their values for this node (as defined by the S function; note that in some cases the attribute values will be null, i.e. if the hierarchical structure represents a schema, not the actual document)

E – is the set of edges in the structure

$$E = \{e = (v, w), v \in V, w \in V\}\ . \tag{5}$$

3.2 Complex Tree Schema and Instance

The Complex Tree definition given above is a general one, when practical uses differentiate between Complex Tree Schema and Complex Tree Instance (i.e. as generalizations of XSD and XML). While the rest of this paper is based on this general definition, this chapter will explain the specific versions of the Complex Tree.

Complex Tree Schema (CTS), like database schema or XML Schema, is a structure that describes the possible outlook of multiple structure it is representing (Complex Tree Instances (CTI). As such, the CTS has less actual nodes and no attribute values – no actual data. On the other hand the CTS may have some types that will not occur in the instance of this schema.

Complex Tree Instance, like the actual database table or XML document, is a structure that stores the data or knowledge. As such, it must not have all the types from the schema, but all the attributes must have determined values (even if it represents unknown) and the structure of the nodes may become very complex.

We represent the *i*-th Complex Tree Schema as $CT^{(i)}$, where:

$$CT^{(i)} = (T^{(i)}, S^{(i)}, V^{(i)}, E^{(i)})\ . \tag{6}$$

Each CTS may have multiple Complex Tree Instances, we represent the *j*-th CTI of *i*-th CTS as as $CT^{(i)}_{(j)}$, where:

$$CT^{(i)}_{(j)} = (T^{(i)}_{(j)}, S^{(i)}_{(j)}, V^{(i)}_{(j)}, E^{(i)}_{(j)}) \ . \tag{7}$$

Two distinct definitions of the same structure require different, although similar, approaches to defining the Integration Task. These are presented in the following two sections.

3.3 Integration Task for CTS

For a structure defined as in section 3.1, with respect to the notation in 3.2, the integration Task of Complex Tree Schema is given as follows:

The input of the integration process is *n* Complex Trees $CT^{(1)}$, $CT^{(2)}$, …, $CT^{(n)}$:

$$\begin{gathered} CT^{(1)} = (T^{(1)}, S^{(1)}, V^{(1)}, E^{(1)}) \\ \vdots \\ CT^{(n)} = (T^{(n)}, S^{(n)}, V^{(n)}, E^{(n)}) \ . \end{gathered} \tag{8}$$

The output of the integration process is one Complex Tree Schema *CT**, connected with input structures by a group of criteria.

$$CT^* = (T^*, S^*, V^*, E^*) \ . \tag{9}$$

The parameters of the integration task, are the integration criteria $K=\{K_1,K_2,\ldots K_n\}$ tying *CT** with CT_1 and CT_2, each at least at a given level $\alpha_1,\ldots,\alpha_n$

$$K_i\left(CT^* \middle| CT^{(1)}, CT^{(2)}, \ldots, CT^{(n)}\right) \geq \alpha_i \ . \tag{10}$$

Alternatively the integration process may be defined as a function *I*:

$$I: CTS^n \rightarrow CTS \ , \tag{11}$$

where *CTS* is the space of all possible Complex Tree Schemas.

Introduction of explicit integration criteria to the definition of the integration task is crucial, as without those the result of the process would not hold any relation to the input. The criteria also help define the aim of the integration – whether keeping all the information or the precise structure is most important (for more details see section 4).

Note also that integration defined as above automatically has one criterion met, without explicitly stating so, that is keeping the same type of structure as the output, as the input structures. Thus this criterion is not necessary to be defined.

3.4 Integration Task for CTI

For the same Complex Tree Schema multiple Instances are allowed, with different actual data (i.e. it is possible to have multiple XML documents with the same XSD). In this section we simplify the notation (only one CTS is used), that is instead of notation:

$$CT^{(i)}{}_{(j)} = (T^{(i)}{}_{(j)}, S^{(i)}{}_{(j)}, V^{(i)}{}_{(j)}, E^{(i)}{}_{(j)}) \,, \tag{12}$$

we will use:

$$CT_j = (T_j, S_j, V_j, E_j) \,. \tag{13}$$

The Integration Task for CTI may be now defined as follows:

The input of the integration process is *n* Complex Trees CT_1, CT_2, …, CT_n from the same schema.

$$\begin{gathered} CT_1 = (T_1, S_1, V_1, E_1) \\ \vdots \\ CT_n = (T_n, S_n, V_n, E_n) \,. \end{gathered} \tag{14}$$

The output of the integration process is one Complex Tree *CT**, connected with input structures by a group of criteria.

$$CT^* = (T^*, S^*, V^*, E^*) \,. \tag{15}$$

The parameters of the integration task, are the integration criteria $K=\{K_1,K_2,\ldots K_m\}$ tying *CT** with $CT_1, CT_2,\ldots,CT_n$, each at least at a given level $\alpha_1,\ldots,\alpha_n$

$$K_i(CT^*|CT_1, CT_2, \ldots, CT_n) \geq \alpha_i \,. \tag{16}$$

Alternatively the integration process may be defined as a function *I*:

$$I: CTI^n \rightarrow CTI \,, \tag{17}$$

where *CTI* is the space of all possible Complex Tree Instances.

The final note to the previous section holds true for Integration of CTI.

4 Integration Criteria

The integration task defined in section 3.3. requires explicitly given criteria, that have to be met at given levels. In this section some basic criteria will be defined, based both on criteria used for XML integration and theoretically defined criteria used for general integration.

4.1 Completeness

The definition of the Complex Tree structure presented in section 3.1 allows for multiple definitions of the completeness criterion. Those would be the *definitions completeness*, *structure completeness* and *data completeness*. It is also possible to merge all those criteria into a *general completeness*, based on some parameters (given by the expert for each problem).

Definition 2. Definitions Completeness is a criterion measuring if all the types and their descriptions (T and S) from the input structures remain after the integration.

$$C_d(CT^*|CT_1, CT_2) = \frac{1}{card\{t:t\epsilon T_1 \cup T_2\}} \sum_{t\epsilon T_1 \cup T_2} \left[\frac{1}{2} m_T(t, T^*) + \frac{1}{2 \cdot card\{a:a\epsilon S_1(t) \vee S_2(t)\}} \sum_{a\epsilon S_1(t) \vee S_2(t)} m_S(a, S^*(t))\right] \tag{18}$$

where $m_T(t,T)$ determines whether element t exists in T and $m_S(a,S(t))$ determines whether a is an attribute of t according to S.

Definition 3. Structure completeness is a criterion measuring if all the nodes (identified by types and labels) from the input structures remain after the integration.

$$C_s(CT^*|CT_1, CT_2) = \frac{1}{card\{v:v\epsilon V_1 \cup V_2\}} \sum_{v\epsilon V_1 \cup V_2} m_V(v, V^*) , \tag{19}$$

where $m_V(v,V)$ determines whether element v exists in V.

Definition 4. Data completeness is a criterion measuring if all the data (attribute values) from the input structures remain after the integration.

$$C_a(CT^*|CT_1, CT_2) = \frac{1}{card\{v:v\epsilon V_1 \cup V_2\}} \sum_{v\epsilon V_1 \cup V_2} \left[\frac{1}{card\{a:a\epsilon A(v)\}} \sum_{a\epsilon A(v)} m_A(a(v), a^*(v))\right] , \tag{20}$$

where $m_A(a(v),a^*(v))$ determines the percentage of attributes of v from input trees occurring in the output trees.

Definition 5. General completeness is a criterion measuring if all the other completes measures are met, each with a given weight to the final result.

When considering CTS the first element is the most important and the parameter γ is 0 (there are no attribute values), while when considering CTI the two last elements are the most important.

$$C(CT^*|CT_1, CT_2) = \alpha C_d(CT^*|CT_1, CT_2) + \beta C_s(CT^*|CT_1, CT_2) + \gamma C_a(CT^*|CT_1, CT_2) . \tag{21}$$

4.2 Precision

Definition 6. Precision is a criterion measuring if no new elements were introduced during the integration and if no duplicate information is in the output tree.

In case of CTS the first element is more important, while in case of the CTI the second element should have higher weight.

$$P(CT^*|CT_1, CT_2) = \alpha \frac{card\{t:t\epsilon T_1 \cup T_2\}}{card\{t:t\epsilon T^*\}} + \beta \frac{card\{v:v\epsilon V_1 \cup V_2\}}{card\{v:v\epsilon V^*\}}. \quad (22)$$

The literature presents some similar criteria called optimality and understandability, which represent that each element after integration represents corresponding elements from both input trees and that each element after integration represents at most one element from each input tree. Here, these criteria are redefined to better fit the definition of the Hierarchical Structure proposed in section 3.1.

4.3 Optimality

The definition of optimality criterion in this section bears little resemblance to the optimality known in literature. Instead it is based on a criterion known as 1-optimality, that has not been used for tree structures often.

Definition 7. Optimality is a criterion measuring how close the output tree of the integration process is to the input trees, in terms of a given tree distance.

$$M(CT^*|CT_1, CT_2) = \frac{\min_H (d(CT,CT_1)+d(CT,CT_2))}{d(CT^*,CT_1)+d(CT^*,CT_2)}, \quad (23)$$

where $d(H_1,H_2)$ is a distance measure for Complex Trees. A proposition of such distance is given below.

Definition 8. Complex Tree Distance is a weighted average of distances between the types of both CTs, the type attribute functions of both CTs and tree structures of both CTs.

In case of CTS the first two elements of the sum are more important, while for the CTI the tree distance is the most important.

$$d(CT_1, CT_2) = \alpha \cdot d_T(T_1, T_2) + \beta \cdot \frac{1}{card\{T_1 \cup T_2\}} \sum_{i=1}^{card\{T_1 \cup T_2\}} d_S(S_1(t_{1i}), S_2(t_{2i})) + \gamma \cdot d_{VE}((V_1, E_1), (V_2, E_2)), \quad (24)$$

where
distance between sets of types is measured as the relation of number of corresponding types in both sets to the overall number of types:

$$d_T(T_1, T_2) = 1 - \frac{card\{t:t\epsilon T_1 \wedge t\epsilon T_2\}}{card\{t:t\epsilon T_1 \vee t\epsilon T_2\}}, \quad (25)$$

distance between type attribute functions S is measured as the number of attributes attributed to the same types by both:

$$d_S(S_1(t_1), S_2(t_2)) = \begin{cases} 1 - \frac{card\{a:a\in S_1(t_1) \wedge a\in S_2(t_2)\}}{card\{a:a\in S_1(t_1) \vee a\in S_2(t_2)\}} & \text{if } t_1 = t_2 \\ 0 \text{ otherwise} \end{cases}, \quad (26)$$

distance between trees (directed graphs) is measured using a fast tree distance measure; in this case the tree edit measure introduced in [12], due to its linear complexity:

$$d_{VE}\big((V_1,E_2),(V_1,E_2)\big) = \frac{I\cdot c_i + D\cdot c_d + \sum_{r\in R} c_r(v_{1r},v_{2r})}{N\cdot c_i + M\cdot c_d}, \tag{27}$$

in which:

N and *M* are the numbers of nodes in first and second Complex Tree, respectively,

I and *D* are the numbers of insert and delete operations required to transform first tree to the second, respectively,

c_i and c_d are the costs of insertion and deletion operation, respectively (here assumed to be 1),

R is the set of all pairs of nodes to be replaced,

$c_r(v_1,v_2)$ is a function determining the cost of replace operation between two nodes, normally a complex operation due to character of this meta-relationship, here for simplicity we assume it is equal to 0 if nodes are identical, equal to 1/3 if the nodes have the same type, equal to 2/3 if the nodes have the same type and label, and if they share identical attribute values it is equal to:

$$c_r(v_{1r},v_{2r}) = 1 - \frac{card\{a: a\in A_1 \wedge a\in A_2\}}{3card\{a: a\in A_1 \vee a\in A_2\}}. \tag{28}$$

5 Simple Integration Algorithm

The aim of defining explicit integration criteria for the Integration Task is to make the decision about selecting an actual algorithm for a specific practical task simpler for the user. Even the few criteria presented in section 4 allow for multiple approaches to integration, for example completeness equal to 1 will almost never occur at the same tame as optimality equal to 1. In some tasks the first criterion would be more important, in others – the second. In this section we will present a simple algorithm that matches a given set of criteria at desired levels, in order to show that it is possible to design algorithms based on given requirements.

The simple integration algorithm was designed to fit the following criteria:

- Integration is done on schema level;
- General completeness must be equal to 1, with parameters α=2/3, β=1/3 and γ=0, respectively (see note in section 4.1);
- Precision equal to 1, with parameters α=½ and β=½, respectively;
- Non-zero optimality (note: this is always assured).

The description of the integration task is already provided in section 3.3. Below, only the algorithm will be presented, in a version for two input CTS's (n = 2).

Input: $CTS^{(1)}$, $CTS^{(2)}$
Output: *CTS**
BEGIN
$T^*:=T^{(1)}$
For each type *t* in $T^{(2)}$ do

If there is no matching (similar) type *t* present in *T**, add *t* to *T**
otherwise go to next *t*

For each type *t* in *CTS** do

1. For each *a* in $S^{(1)}(t)$ do
   Add *a* to *S*(t)*
2. For each a in $S^{(2)}$(t) do
   If *a* is not present in *S*(t)*, add *a* to *S*(t)*
   otherwise go to next *a*

$V^*=V^{(1)}$
$E^*=E^{(1)}$
For each node *v* in $V^{(2)}$ in post-order:

1. If *v* is the root element of $CTS^{(2)}$, and the root of *CTS** is different than *v*, and if *v* is not a child of the root of *CTS**, add *v* as a child of the root of *CTS**; go to next node
2. If *v* is the root element of $CTS^{(2)}$ matching the root of *CTS**, go to next node
3. If v is present in *CTS**
   If *v* is a child of node *w* in $CTS^{(2)}$ and is not a child of the same node in *CTS**, add *v* as a child of node *w* in *CTS**
   otherwise go to next node
4. If *v* is not present in *CTS**
   Locate node *w* in *CTS** that has a matching (similar) node *w* as the parent of node *v* in $CTS^{(2)}$
   Add node *v* as a child of node *w* in *CTS**

END

The algorithm was designed specifically for low computational complexity. The fist loop section takes no more than $O\left(card\{T^{(2)}\}\right)$, which is linear. The same holds for the second loop section (which may be restricted from top by $\max\{a: a \in S^{(1)}(t) \vee a \in S^{(2)}(t), t \in T^{(1)} \cup T^{(2)}\}$). While the last loop section may appear complex, only case selection takes place and the complexity is $O\left(card\{V^{(2)}\}\right)$. The overall computational complexity of the algorithm is then linear and restricted by the number of elements in the structures, specifically $O\left(card\{T^{(2)}\} \cdot \left(1 + \max\{a: a \in S^{(1)}(t) \vee a \in S^{(2)}(t), t \in T^{(1)} \cup T^{(2)}\}\right) + card\{V^{(2)}\}\right)$.

5.1 Example

For an example of this simple algorithm, we will use the following structures:

$CT_1 = (T_1, S_1, V_1, E_1)$, $CT_2 = (T_2, S_2, V_2, E_2)$
T_1={root, leaf}, T_2={root, branch, leaf}, $\forall_t: S_1(t) = S_2(t) = \emptyset$
V_1={ (root,root,-), (a,leaf,-), (b,leaf,-) }
V_2 ={ (root,root,-), (a,leaf,-), (b,leaf,-), (c,leaf,-), (x,branch,-) }
E_1 = { (root,a), (root,b) }, E_2 = { (root, a), (root, x), (x,b), (x,c) }

The integration algorithm first joins the sets of node types, this results in:
$T^* = \{root, branch, leaf\}$

The function S determining the attributes in both Complex Trees is null, as there are no attributes present. The S^* thus remains null.

The process of integrating both tree structures takes place for V and E simultaneously. The result of this process is in this example:

$V^* = \{ (root,root,-), (a,leaf,-), (b,leaf,-), (b,leaf,-), (c,leaf,-), (x,branch,-) \}$
$E^* = \{ (root,a), (root, b), (root, x), (x, b), (x,c) \}$

The node labeled as b appear twice in the output structure, otherwise it is similar to T_2. The general completeness is equal to 1, the precision is also equal to 1 and the optimality is greater than 0.

6 Future Work

This paper presented only a general idea and basic concepts of the integration task with clearly defined criteria and with a structure designed just for the integration purposes.

The criteria described in section 4 are the most basic ones, deriving both from the study of XML integration in practical enterprises and from the general theory of integration, independent of the data structure. It is possible to develop more criteria, both theoretically and based on the existing solutions and integration requirements.

The simple algorithm presented in section 5 is the obvious solution for the criteria requirements stated for its creation. The same criteria may be the base for better (i.e. faster) algorithms or algorithm compatible with other, more precise criteria. With different set of base criteria, different algorithms may also be proposed. Specifically, the existing algorithms, once the criteria for them are explicitly stated, may have the possibility of improvement.

Acknowledgements. This paper was partially supported by Polish Ministry of Science and Higher Education under grant no. N N519 407437.

References

1. Adams, E.N.: N-Trees as Nestrings: Complexity, Similarity, and Consenss. Journal of Classification 3, 299–317 (1986)
2. Arenas, M., Libkin, L.: A Normal Form for XML Documents. ACM Transactions on Database Systems 29(1), 195–232 (2004)
3. Bae, J.K., Kim, J.: Integration of heterogeneous models to predict consumer behavior. Expert Systems with Applications 37, 1821–1826 (2010)
4. Barthelemy, J.P., McMorris, F.R.: The Median Procedure for n-Trees. Journal of Classification 3, 329–334 (1986)
5. Bonifati, A., Ceri, S.: Comparative Analysis of Five XML Query Languages. ACM SIGMOD Record 29(1) (2000)

6. Danilowicz, Cz., Nguyen, N.T.: Methods for choice of representation of ordered partitions and coverings, Wroclaw (1992)
7. Day, W.H.E.: Optimal Algorithms for Comparing Trees with Labled Leaves. Journal of Classification 2, 7–28 (1985)
8. Delobel, C., Reynaud, C., Rousset, M.C., Sirot, J.P., Vodislav, D.: Semantic integration in Xyleme: a uniform tree-based approach. Data & Knowledge Engineering 44, 267–298 (2003)
9. Do, H.-H., Melnik, S., Rahm, E.: Comparison of Schema Matching Evaluations. In: Chaudhri, A.B., Jeckle, M., Rahm, E., Unland, R. (eds.) NODe-WS 2002. LNCS, vol. 2593, pp. 221–237. Springer, Heidelberg (2003)
10. Farach, M., Przytycka, T.M., Thorup, M.: On the agreement of many trees. Information Processing Letters 55, 297–301 (1995)
11. Lian, W., Cheung, D.W., Mamoulis, N., Yiu, S.M.: An Efficient and Scalable Algorithm for Clustering XML Documents by Structure. IEEE Transactions on Knowledge and Data Engineeing 16(1) (January 2004)
12. Maleszka, M., Mianowska, B., Prusiewicz, A.: On some approaches to reduce the computational costs of similarity measures between XML trees. In: Nguyen, N.T., Kolaczek, G., Gabrys, B. (eds.) Knowledge Processing and Reasoning for Information Society, pp. 165–180. Exit, Warszawa (2008)
13. Rahm, E., Bernstein, P.A.: A survey of approaches to automatic schema matching. The VLDB Journal 10, 334–350 (2001)
14. Routledge, N., Bird, L., Goodchild, A.: UML and XML Schema. In: Zhou, X. (ed.) Conferences in Research and Practice in Information Technology, vol. 5 (2002)
15. Stinebrickner, R.: s-Consensus Trees and Indices. Bulletin of Mathematical Biology 46, 923–935 (1984)