Describing Semistructured Data*

Luca Cardelli
Microsoft Research

Abstract

We introduce a rich language of *descriptions* for semistructured tree-like data, and we explain how such descriptions relate to the data they describe. Various query languages and data schemas can be based on such descriptions.

1 Introduction

1.1 Trees and their Descriptions

We consider data that is represented as labeled trees, and we ask: how can we *describe* the structure of such data? We use *descriptions* (or, more precisely, *formulas* in a special logic) to talk about properties of labeled trees. A description denotes the collection of trees that, by a precise definition, match the description.

A description can be used as a yes/no query against labeled trees: "Does the tree under consideration match the description?". With some extensions, a description can be used as a query returning a complex result. Hence, description languages can be seen as kernels of query languages. Some special classes of descriptions can be used as path queries, or as flexible type systems (schemas) for the data.

We aim to find a very general class of descriptions, so we can accommodate a large class of actual or potential schema languages and query languages. Most of all, though, we aim to communicate an approach to formalizing descriptions that can be adapted to different contexts. The presentation here is introductory and not completely formal; we refer to other work for full details [11].

We consider only labeled trees, not labeled graphs. Labeled trees are closer to common practice in XML, while labeled graphs are the more general model used for semistructured data. While graphs are natural generalizations of trees, descriptions of graphs are *much* more complex than descriptions of trees. So, for the moment at least, we just restrict ourselves to trees.

We want both our data and our descriptions to be compositional: if A is a description of a tree, and B is a description of another tree, then a simple composition (e.g., root-merge) of the trees should correspond to a simple composition of A and B. Note that this means that we are not just interested in describing paths through a tree, but also in describing how trees branch out.

Our syntax for labeled trees, and a small but important fragment of our description language, are summarized below:

Syntax for Trees	
P, Q ::=	
$\mathbf{0}$	root
$n[P]$	edge
$P \mid Q$	composition
Basic Descriptions	
A, B ::=	
$\mathbf{T}$	there is anything
$\mathbf{0}$	there is only a root
$n[A]$	there is one edge n to a subtree
$A \mid B$	there are two joined trees

The description $\mathbf{T}$ describes any tree. The description $\mathbf{0}$ describes the empty tree (consisting of just a root node). The description $n[A]$ describes a tree consisting of a single edge labeled n off the root, leading to a subtree described by A. The description $A \mid B$ describes any tree that can be seen as the root-merge of two trees that are described by A and B.

1.2 Historical Remarks

This work arose originally from the observation that the areas of *semistructured databases* [4] and *mobile computation* [9] have some surprising similarities at the technical level. These areas are inspired by the need to find better ways to describe, respectively, data and computation on the Internet. The technical similarities permit the transfer of some techniques between the two areas. More interestingly, if we can take advantage of the similarities and generalize them, we may obtain a broader model of data and computation on the Internet.

The ultimate source of similarities is the fact that both areas have to deal with extreme dynamicity of data and behavior. In semistructured databases, one cannot rely on uniformity of structure, because data may come from heterogeneous and uncoordinated sources. Still, it is necessary to perform searches based on whatever uniformity one can find in the data. In mobile computation, one cannot rely on uniformi-

*Database Principles Column
Column editor: Leonid Libkin, Department of Computer Science, University of Toronto, Toronto, Ontario M5S 3H5, Canada. E-mail: libkin@cs.toronto.edu.

ty of structure because agents, devices, and networks can dynamically connect, move around, become inaccessible, or crash. Still, it is necessary to perform computations based on whatever resources and connections are available on the network.

As examples of the potential convergence of these two areas, consider the following arguments. First, one can regard data structures stored inside network nodes as a natural extension of network structures, since on a large time/space scale both networks and data are semistructured and dynamic. Therefore, one can think of applying the same navigational and code mobility techniques uniformly to networks and data. Second, since networks and their resources are semistructured, one can think of applying semistructured database searches to the network structure. This is a well-known major problem in distributed computation, going under the name of resource discovery.

2 Labeled Trees

We begin with a simple syntax for semistructured data.

Syntax for Trees

P, Q ::=

0	root
$n[P]$	edge
$P \mid Q$	composition

- **0** represents the tree consisting of a single root node.
- $n[P]$ represents a tree consisting of a single edge labeled n off the root, leading to a subtree represented by P.
- $P \mid Q$ represents the tree obtained by taking the trees represented by P and by Q, and by merging their roots.

For example, the following piece of data:

$$Cambridge[Eagle[chair[\mathbf{0}] \mid chair[\mathbf{0}]]]$$

represents: "in Cambridge there is (nothing but) a pub called the Eagle that contains (nothing but) two empty chairs".

We consider here a commutative composition operation $P \mid Q$, for unordered trees. However, it is easy to consider a non commutative operation, say $P ; Q$, for ordered trees, that can replace or be added to $P \mid Q$. This may be necessary, for example, to model certain XML trees more precisely and conveniently.

The description of trees in the syntax given above is not unique. For example the expressions $P \mid Q$ and $Q \mid P$ represent the same (unordered) tree; similarly, the expressions $\mathbf{0} \mid P$ and P represent the same tree. We consider two expressions P and Q equivalent when they represent the same tree, and we write P 7 Q. The relation P 7 Q is an equivalence and a congruence (i.e., equals can be replaced by equals in any syntactic context). Moreover, the following simple properties hold:

$P \mid Q$ 7 $Q \mid P$
$(P \mid Q) \mid R$ 7 $P \mid (Q \mid R)$
$P \mid \mathbf{0}$ 7 P

3 Descriptions

As an example, here is a description asserting that there is exactly one edge labeled *Cambridge*, leading to at least one edge labeled *Eagle*, leading to least one edge labeled *chair*, leading to nothing:

$$Cambridge[Eagle[chair[\mathbf{0}] \mid \mathbf{T}] \mid \mathbf{T}]$$

This assertion happens to be true of the tree shown earlier. In general, our descriptions include both assertions about trees, such as the one above, and standard logical connectives for composing assertions.

The exact meaning of descriptions is given by a *satisfaction relation* relating a tree with a description. The term *satisfaction* comes from logic; for reasons that will become apparent shortly, we will also call this concept *matching*. The basic question we consider is: does this tree match this description?

The satisfaction/matching relation between a tree P (actually, an expression P representing a tree) and a description A is written, for the purposes of this paper:

$$P \text{ matches } A$$

Informally, the matching relation can be described as follows, where at the same time we introduce the syntax of descriptions and their meaning. It is important to realize that a descriptions states a property that holds at a certain place in the tree: a top-level description talks about a tree from its root, and a sub-description may talk about a part of the whole tree.

- Invariance

  if P matches A and P 7 Q
  then Q matches A

- **T**: anything

  any P matches **T**

- $\neg A$: negation

  if P does not match A
  then P matches $\neg A$

- $A \wedge B$: conjunction

  if P matches A and P matches B
  then P matches $A \wedge B$

- **0**: root

  0 (the tree expression) matches **0** (the description)

- $n[A]$: edge

  if P matches A
  then $n[P]$ matches $n[A]$

- $A \mid B$: composition

  if P matches A and Q matches B
  then $P \mid Q$ matches $A \mid B$

- Ó$x.A$: universal quantification

  if, for all labels n, P matches $A\{x \leftarrow n\}$
  (i.e., A where x is replaced by n)
  then P matches Ó$x.A$

- $\mu X.A$: least fixpoint (with X occurring positively in A)

  if P is contained in the least fixpoint of the
  function $\lambda X.A$, taken over the collection
  of sets of labeled trees ordered by inclusion,
  then P matches $\mu X.A$

Many useful derived connectives can be defined from the ones above. For example:

Derived Connectives

F	@ $\neg$**T**	false
$A \vee B$	@ $\neg(\neg A \wedge \neg B)$	disjunction
$A \Rightarrow B$	@ $\neg A \vee B$	implication
$A \Leftrightarrow B$	@ $(A \Rightarrow B)$ $\wedge (B \Rightarrow A)$	logical equivalence
ô$x.A$	@ $\neg$Ó$x.\neg A$	existential quantification
$A \parallel B$	@ $\neg(\neg A \mid \neg B)$	decomposition
$A^{Ó}$	@ $A \parallel$ **F**	every part matches A
$A^{ô}$	@ $A \mid$ **T**	some part matches A
✧A	@ $\mu X.\ A \vee$ ô$x.\ x[X] \mid$ **T**	somewhere A holds
¤A	@ $\neg$✧$\neg A$	everywhere A holds
$A \mid\Rightarrow B$	@ $\neg(A \mid \neg B)$	parallel implication
$n[\Rightarrow A]$	@ $\neg n[\neg A]$	nested implication
$\nu X.A$	@ $\neg(\mu X.\neg A\{X \leftarrow \neg X\})$	greatest fixpoint

- Many operators are derived as standard DeMorgan duals: disjunction, existential quantification, and the everywhere modality.
- *Decomposition*, $A \parallel B$, is the DeMorgan dual of composition. A decomposition description $A \parallel B$ is satisfied if for every parallel decomposition of the tree in question, either one component satisfies A or the other satisfies B.
- Then, $A^{Ó}$ means that in every decomposition either one component satisfies A or the other satisfies **F (@ ¬T);** since the latter is impossible, in every possible decomposition one component must satisfy A. For example: $(n[\mathbf{T}] \Rightarrow n[m[\mathbf{T}]])^{Ó}$ means that every edge n that can be found off the root leads to a single edge m. The DeMorgan dual of $A^{Ó}$ is $A^{ô}$, which means that it is possible to find a decomposition where one component satisfies A. For example, $n[m[\mathbf{T}]^{ô}]^{ô}$ means that there is at least one edge n that leads to at least one edge m.
- *Normal Implication:* $A \Rightarrow B$ @ $\neg A \vee B$. This is the standard definition of implication. Note that this means that P matches $A \Rightarrow B$ if whenever P matches A then the same P matches B. As examples, consider *Borders*[**T**] $\Rightarrow$ *Borders*[*Starbucks*[**T**] | **T**], stating that a *Borders* bookstore must contain a *Starbucks* shop, and (*NonSmoker*[**T**] | **T**) $\Rightarrow$ (*Smoker*[**T**] | **T**), stating that if there is a non-smoker, there is also a smoker nearby (the tree P must be composed of both a smoker and a non-smoker).
- *Parallel Implication:* $A \mid\Rightarrow B$ @ $\neg(A \mid \neg B)$. This means, by definition, that it is not possible to split the root of the current tree in such a way that one part satisfies A and the other does not satisfy B. In other words, every way we split the root of the current tree, if one part satisfies A, then the other part must satisfy B. For example, *NonSmoker*[**T**] $\mid\Rightarrow$ (*Smoker*[**T**] | **T**) is a slightly more compact formulation of the property of nonsmokers given above.
- *Nested Implication*: $n[\Rightarrow A]$ @ $\neg n[\neg A]$. This means, by definition, that it is not possible that an edge n leads to a tree that does not satisfy A. In other words, if there is an edge n, it leads to a tree that satisfies A. For example: *Borders*[$\Rightarrow$*Starbucks*[**T**] | **T**] is, again, a slightly more compact formulation of the property of Borders given above.
- *Greatest Fixpoint*: The dual of the least fixpoint operator $\mu X.A$ is the greatest fixpoint operator $\nu X.A$. For example $\mu X.X$ is equivalent to **F**, while $\nu X.X$ is equivalent to **T**. More interestingly, $\mu X.\ \mathbf{0} \vee m[X]$ describes every tree of the form $m[m[\ldots\ m[\mathbf{0}]]]$, and, on finite trees, it is equivalent to $\nu X.\ \mathbf{0} \vee m[X]$. However, if we consider infinite trees, the distinction between least and greatest fixpoint becomes more important. For example, the infinite tree $m[m[\ldots]]$ satisfies $\nu X.\ \mathbf{0} \vee m[X]$, but does not satisfy $\mu X.\ \mathbf{0} \vee m[X]$. When we consider only finite trees, as we do here, the μ and ν operators are quite similar in practice, since most interesting descriptions have a single fixpoint.
- *Somewhere*. A tree P satisfies ✧A if there is a subtree Q of P that satisfies A. This is defined by a recursive description.
- *Everywhere*: ¤A @ $\neg$✧$\neg A$. What is true everywhere? Not much, unless we qualify a property by negation or implication. For example, ¤$\neg(n[\mathbf{T}]^{ô})$ means that there is no edge called n anywhere. Moreover, we can write ¤$(A \Rightarrow B)$ to mean that everywhere A is true, B is true as well. For example, ¤(*NonSmoker*[**T**] $\mid\Rightarrow$ (*Smoker*[**T**] | **T**)): everywhere there is a non-smoker there is also a smoker.

4 Equivalent Descriptions

A precise semantics of descriptions helps in deriving equivalences between descriptions (and, further, between queries) [11]. Many such equivalences can be derived; we list some of them here, just to give an idea of the rich collection of properties one can rely on. Equivalences can be used by a query optimizer; in particular, they can be used to push negation to the leaves of a description, by dualizing operators.

Equivalent Descriptions

$n[A]$	$\Leftrightarrow$	$n[\mathbf{T}] \wedge n[\Rightarrow A]$
$n[\Rightarrow A]$	$\Leftrightarrow$	$n[\mathbf{T}] \Rightarrow n[A]$
$n[\mathbf{F}]$	$\Leftrightarrow$	$\mathbf{F}$
$n[\Rightarrow \mathbf{T}]$	$\Leftrightarrow$	$\mathbf{T}$
$n[A \wedge B]$	$\Leftrightarrow$	$n[A] \wedge n[B]$
$n[\Rightarrow A \vee B]$	$\Leftrightarrow$	$n[\Rightarrow A] \vee n[\Rightarrow B]$
$n[A \vee B]$	$\Leftrightarrow$	$n[A] \vee n[B]$
$n[\Rightarrow A \wedge B]$	$\Leftrightarrow$	$n[\Rightarrow A] \wedge n[\Rightarrow B]$
$n[\hat{0}x.A]$	$\Leftrightarrow$	$\hat{0}x.n[A] \quad (x \neq n)$
$n[\Rightarrow \acute{O}x.A]$	$\Leftrightarrow$	$\acute{O}x.n[\Rightarrow A] \quad (x \neq n)$
$A \mid \mathbf{F}$	$\Leftrightarrow$	$\mathbf{F}$
$A \parallel \mathbf{T}$	$\Leftrightarrow$	$\mathbf{T}$
$\mathbf{T} \mid \mathbf{T}$	$\Leftrightarrow$	$\mathbf{T}$
$\mathbf{F} \parallel \mathbf{F}$	$\Leftrightarrow$	$\mathbf{F}$
$A \mid (B \vee C)$	$\Leftrightarrow$	$(A \mid B) \vee (A \mid C)$
$A \parallel (B \wedge C)$	$\Leftrightarrow$	$(A \parallel B) \wedge (A \parallel C)$

5 From Descriptions to Queries

A satisfaction relation, such as the one defined in the previous section, is not always decidable. However, in some interesting cases, the problem of whether P matches A becomes decidable [14]; some complexity results are also known [16]. A decision procedure for such a matching problem is also called a *modelchecking* algorithm. Such an algorithm implements a matching procedure between a tree and a description, where the result of the match is just success of failure.

For example, the following match succeeds. The description can be read as stating that there is an empty chair at the *Eagle* pub; the matching process verifies that this fact holds starting from the root of the tree:

$$\begin{array}{l} Eagle[chair[John[\mathbf{0}]] \mid chair[Mary[\mathbf{0}]] \mid chair[\mathbf{0}]] \\ \text{matches} \\ Eagle[chair[\mathbf{0}] \mid \mathbf{T}] \end{array}$$

More generally, we can imagine collecting information, during the matching process, about which parts of the tree match which parts of the description. Further, we can enrich descriptions with markers that are meant to be bound to parts of the tree during matching; the result of the matching algorithm is then either failure or an association of markers to the trees that match them.

We can thus extend descriptions with *matching variables*, X. For example by running the matching computation for:

$$\begin{array}{l} Eagle[chair[John[\mathbf{0}]] \mid chair[Mary[\mathbf{0}]] \mid chair[\mathbf{0}]] \\ \text{matches} \\ Eagle[chair[X] \mid \mathbf{T}] \end{array}$$

we obtain, bound to X, either somebody sitting at the *Eagle*, or the indication that there is an empty chair. Moreover, by matching:

$$\begin{array}{l} Eagle[chair[John[\mathbf{0}]] \mid chair[Mary[\mathbf{0}]] \mid chair[\mathbf{0}]] \\ \text{matches} \\ Eagle[chair[(\neg\mathbf{0})\wedge X] \mid \mathbf{T}] \end{array}$$

we obtain, bound to X, somebody (not $\mathbf{0}$) sitting at the *Eagle*. Here the answer could be either $John[\mathbf{0}]$ or $Mary[\mathbf{0}]$, since both bindings lead to a successful global match. Moreover, by using the same variable more than once we can express constraints: the description

$$Eagle[chair[(\neg\mathbf{0})\wedge X] \mid chair[X] \mid \mathbf{T}]$$

is successfully matched if there are two people with the same name (or any two equal structures) sitting at the *Eagle*.

These generalized descriptions that include matching variables can thus be seen as *queries*. The result of a successful matching can be seen as a possible answer to a query, and the collection of all possible successful matches as the collection of all answers.

For serious semistructured database applications, we need also sophisticated ways of matching labels (e.g. with wildcards and lexicographic orders) and of matching paths of labels. For the latter, though, we already have considerable flexibility within the existing logic; consider the following examples:

- *Exact path*. The description $n[m[p[X]] \mid \mathbf{T}]$ means: match a path consisting of the labels n, m, p, and bind X to what the path leads to. Note that, in this example, other paths may lead out of n, but there must be a unique path out of m and p.
- *Dislocated path*. The description $n[✧(m[X] \mid \mathbf{T})]$ means: match a path consisting of a label n, followed by an arbitrary path, followed by a label m; bind X to what the path leads to.
- *Disjunctive path*. The description $n[p[X]] \vee m[p[X]]$ means: bind X to the result of following either a path n,p, or a path m,p.
- *Negative path*. The description $✧m[\neg(p[\mathbf{T}] \mid \mathbf{T}) \mid q[X]]$ means: bind X to anything found somewhere under m, inside a q but not next to a p.
- *Wildcard and restricted wildcard*. $m[\hat{0}y.y\neq n \wedge y[X]]$ means: match a path consisting of m and any label different from n, and bind X to what the path leads to. (Inequality of labels can be easily added to the descriptions [11]).
- *Kleene Star for paths.* $\mu X.\ A \vee (m[X] \mid \mathbf{T})$ means: match a path consisting of any number of m edges leading to a subtree that matches A.

Although we have a lot of power and flexibility in defining descriptions for paths, we may want to have a convenient syntax for such common situations; a syntax for paths that easily translates into our descriptions is defined in [11].

In related work [11], we use a rather traditional SQL-style *select-from* construct for constructing answers to queries, after the matching phase described above. The resulting query language TQL [3], is fairly similar to XML-QL [4], perhaps indicating a natural convergence of query mechanisms.

We should emphasize, though, that our composition operator is very powerful, and not very common in the query literature. It can be used, for example, for the following purposes:

- Composition makes it easy to describe record-like structures both partially ($(b[\mathbf{T}] \mid c[\mathbf{T}] \mid \mathbf{T})$ means: contains b, c, and possibly more fields) and completely ($(b[\mathbf{T}] \mid c[\mathbf{T}])$ means: contains only b and c fields); complete descriptions are difficult in path-based approaches.
- Composition makes it possible to bind a variable to 'the rest of the record', as in "X is everything but the paper title": $paper[title[\mathbf{T}] \mid X]$.
- Composition makes it possible to describe schemas, as shown next.

6 Schemas

Path-like description explore the vertical structure of trees. Our descriptions can also easily explore horizontal structure, as is common in schemas for semistructured data. (E.g. in XML DTDs, XDuce [19] and XMLSchema [1]. However, our present formulation deals directly only with unordered structures.)

For example, we can extract from our description language the following regular-expression-like sublanguage, inspired by XDuce types. Every expression of this language denotes a set of trees:

$\mathbf{0}$	the empty tree
$A \mid B$	an A next to a B
$A \vee B$	either an A or a B
$n[A]$	an edge n leading to an A
$A*$ @ $\mu X.\ \mathbf{0} \vee (A \mid X)$	
	finite composition of zero or more A 's
$A+$ @ $A \mid A*$	finite composition of one or more A 's
$A?$ @ $\mathbf{0} \vee A$	optionally an A

In general, we believe that a number of proposals for describing the shape of semistructured data can be embedded in our description language, or in something closely related. Each such proposal usually comes with an efficient algorithm for checking membership or other properties. These efficient algorithms, of course, do not fall out automatically from a general framework. Still, a general frameworks such as ours can be used to compare different proposals.

7 Conclusions

Semistructured databases have developed flexible ways of querying data, even when the data is not rigidly structured according to schemas [4]. In relational database theory, query languages are nicely related to query algebras and to query logics. However, query algebras and query logics for semistructured database are not yet well understood.

We believe we have provided at least an example of a query logic that is suitable for semistructured data. Moreover, in related work [11,12] we describe a *table algebra* for our query logic; this has the same function as relational algebra for relational databases, and can take advantage of a rich set of algebraic properties, such as the ones listed in section 4.

An implementation of a query language, TQL [3], based on these ideas is being carried out in Pisa by Giorgio Ghelli and co-workers. The current prototype can be used to query XML documents accessible through files or through web servers.

Acknowledgments

Giorgio Ghelli restarted my interest in databases, and in particular in semistructured data, and we have since coauthored works that are partially reflected in this paper.

References

[1] **XML schema**. Available from http://www.w3c.org, 2000.

[2] **XML query**. Available from http://www.w3c.org, 2001.

[3] **TQL**. Available from http://macbeth.di.unipi.it/TQL. 2001.

[4] S. Abiteboul, P. Buneman, D. Suciu.: **Data on the Web**. Morgan Kaufmann Publishers, San Francisco, 2000.

[5] S. Abiteboul, R. Hull, and V. Vianu. **Foundations of Databases**. Addison-Wesley, Reading, MA, 1995.

[6] S. Abiteboul, Dallan Quass, Jason McHugh, Jennifer Widom, and Janet L. Wiener. **The Lorel query language for semistructured data**. International Journal on Digital Libraries, 1(1):68-88, 1997.

[7] P. Buneman, S. B. Davidson, G. G. Hillebrand, and D. Suciu. **A query language and optimization techniques for unstructured data**. In Proc. of the 1996 ACM SIGMOD International Conference on Management of Data (SIGMOD), Montreal, Quebec, Canada, pages 505-516, 4-6 June 1996. SIGMOD Record 25(2), June 1996.

[8] P. Buneman, B. Pierce.: **Union Types for Semistructured Data**. Proceedings of the International Database Programming Languages Workshop, 1999. Also available as University of Pennsylvania Dept. of CIS technical report MS-CIS-99-09.

[9] L. Cardelli.: **Abstractions for Mobile Computation.** Jan Vitek and Christian Jensen, Editors. Secure Internet Programming: Security Issues for Mobile and Distributed Objects. LNCS. 1603, 51-94, Springer, 1999.

[10] L. Cardelli. **Semistructured computation**. In Proc. of the Seventh Intl. Workshop on Data Base Programming Languages (DBPL), 1999.

[11] L. Cardelli, G. Ghelli.: **A Query Language Based on the Ambient Logic.** In Proceedings ESOP'01, volume 2028 of LNCS, pages 1-22. Springer, 2001.

[12] L. Cardelli and G. Ghelli. **Evaluation of TQL queries**. Available from http://www.di.unipi.it/~ghelli/papers.html, 2001.

[13] L. Cardelli, A.D. Gordon.: **Mobile ambients**. In Proceedings FoSSaCS'98, volume 1378 of LNCS, pages 140-155. Springer-Verlag, 1998. To appear in Theoretical Computer Science.

[14] L. Cardelli, A.D. Gordon: **Anytime, Anywhere. Modal Logics for Mobile Ambients.** Proceedings POPL'00, 365-377, 2000.

[15] D. Chamberlin, J. Robie, and D. Florescu. **Quilt: An XML query language for heterogeneous data sources.** In Proc. of Workshop on the Web and Data Bases (WebDB), 2000.

[16] W. Charatonik and J.-M. Talbot: **The Decidability of Model Checking Mobile Ambients**. Proceedings of the 15th Annual Conference of the European Association for Computer Science Logic. Springer LNCS, 2001 (to appear).

[17] A. Deutsch, D. Florescu M. Fernandez, A. Levy, and D. Suciu. **A query language for XML**. In Proc. of the Eighth International World Wide Web Conference, 1999.

[18] M. Fernandez, D. Florescu, A. Levy, and D. Suciu. **A query language and processor for a web-site management system.** In Proc. of Workshop on Management of Semistructured Data, Tucson, 1997.

[19] M. Fernandez, Daniela Florescu, Jaewoo Kang, Alon Levy, and Dan Suciu. **Catching the boat with Strudel: experiences with a web-site management system**. In Proc. of ACM SIGMOD International Conference on Management of Data (SIGMOD), pages 414-425, 1998.

[20] G. Ghelli. **TQL as an XML query language**. Available from http://www.di.unipi.it/_ghelli/papers.html, 2001.

[21] R. Goldman, J. McHugh, and J. Widom. **From semistructured data to XML: Migrating the lore data model and query language**. In Proc. of Workshop on the Web and Data Bases (WebDB), pages 25-30, 1999.

[22] B.C. Pierce H. Hosoya. XDuce: **A typed XML processing language** (preliminary report). In Proc. of Workshop on the Web and Data Bases (WebDB), 2000.

[23] F. Neven and T. Schwentick. **Expressive and efficient pattern languages for tree-structured data.** In Proc. of the 19th Symposium on Principles of Database Systems (PODS), 2000.

[24] Y. Papakonstantinou, H.G. Molina, and J. Widom. **Object exchange across heterogeneous information sources.** Proc. of the eleventh IEEE Int. Conference on Data Engineering, Birmingham, England, pages 251-260, 1996.