Designing Semistructured Databases: A Conceptual Approach

Mong Li Lee[1], Sin Yeung Lee[1], Tok Wang Ling[1],
Gillian Dobbie[2], Leonid A. Kalinichenko[3]

[1] School of Computing, National University of Singapore, Singapore
{leeml, jlee, lingtw}@comp.nus.edu.sg
[2] Dept of Computer Science, University of Auckland, New Zealand
gill@cs.auckland.ac.nz
[3] Institute for Problems of Informatics, Russian Academy of Sciences, Russia
leonidk@synth.ipi.ac.ru

Abstract. Semistructured data has become prevalent with the growth of the Internet. The data is usually stored in a database system or in a specialized repository. Many information providers have presented their databases on the web as semistructured data, while others are developing repositories for new applications. Designing a "good" semistructured database is important to prevent data redundancy and updating anomalies. In this paper, we propose a conceptual approach to design semistructured databases. A conceptual layer based on the Entity-Relationship model is used to remove redundancies at the semantic level. An algorithm to map an ER diagram involving composite attributes weak entity types, recursive, n-ary and ISA relationship sets, and aggregations to a semistructured schema graph (S3-Graph) is also given.

1 Introduction

It is increasing important to design good semistructured databases. While many information providers have presented their databases on the web as semistructured data, others are developing repositories for new applications. Consider the building of an e-commerce application. This involves designing and maintaining repositories of data including product catalogs, customer and vendor information and business-to-business transactions. Currently application builders must define and create objects and relationships in the underlying databases, as the details of the schema are not expressed in the data. As with traditional databases, poorly designed databases contain redundant data, leading to undesirable anomalies. A design methodology that seamlessly maps from the objects and relations in the database, to the hierarchical elements and attributes in semistructured data is also required.

In this paper, we describe the research that forms the basis of a methodology to build applications with semistructured data. This comprises three steps:

1. model the underlying database using ER diagrams [1],
2. normalize the ER diagrams [4],
3. model the views on the data using S3-graphs [3].

Introducing a normalized layer has the following advantages. First, anomalies and redundancies can be removed at the semantic level. Second, customized XML views can be generated from the normalized model. Third, data can be stored in relational databases with controlled redundancy. The contribution of this paper is an algorithm that maps ER diagrams to semistructured schema graphs (S3-Graphs), forming a seamless mapping between how the stored data is modeled and the semistructured views of the data. Our study also reveals similarites between the S3-Graph and the hierarchical model and nested relations in that all have limitations in modeling situations with nonhierarchical relationships given their tree-like structures.

The rest of the paper is organized as follows. Section 2 reviews background concepts such as the S3-graph and the ER approach. Section 3 shows how the ER approach can be used to design a semistructured database and gives a comprehensive algorithm to translate an ER diagram into a normal form semistructured schema graph. Finally, we conclude in Section 4.

2 Preliminaries

Data modeling people have acknowledged the fact that if the database attributes are fixed in structure, the modeling power of the data model is greatly reduced. In this section, we review some of the concepts of a semistructured data graph and a semistructured schema graph (S3-Graph) defined in [3]. We also review the ER approach and discuss how it can be used to model semistructured data.

2.1 SemiStructured Graph

With the introduction of XML, semistructured data becomes widespread. XML has characteristics which are very similar to semistructured data: self-describing, deeply nested or even cyclic, and irregular. Graph-based complex object models such as Object Exchange Model (OEM) [7], Document Object Model (DOM) [2], Araneus Data Model (ADM) [6] and semistructured graph [3] provide a natural framework for describing semistructured XML repositories and their DTDs. Figure 1 shows a semistructured data graph. The data graph is basically a labeled graph in which vertices correspond to objects and edges represent the object-subobject relationship. Each edge has a label describing the precise nature of the relationship. Each object has an object identifier (oid) and a value. The value is either atomic or complex, that is, a set of object references denoted as a set of (label, oid) pairs.

Figure 2 shows the schema of the semistructured data graph in Figure 1. A *SemiStructured Schema Graph (S3-Graph)* is a directed graph where each node is either a *root node* or a *component node*. We differentiate nodes in a semistructured data graph from the S3-graph by using the & symbol instead of #. Node #1 is a root node representing the entity Student. Node #2 is both a component node as well as a leaf node. Node #3' is a component node which references the root node Course (Node #3). Each directed edge in the

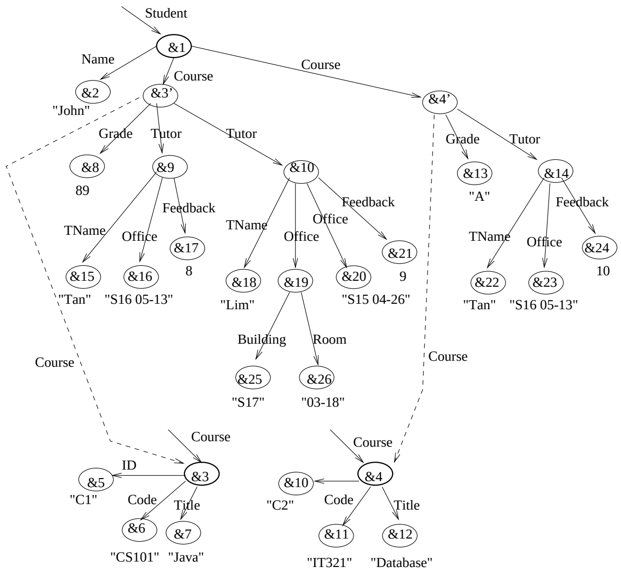

Fig. 1. A semistructured data graph.

S3-graph is associated with a *tag* indicating the relationship between the source and destination nodes. The tag may be suffixed with a "*" indicating multiple occurences. A node V_1 is connected to node V_2 by a *component edge* (a solid arrow line) with a tag T if V_2 is a component of V_1. If T is suffixed with a "*", the relationship is interpreted as "Entity type represented by V_1 has many T". Otherwise, it is interpreted as "Entity type represented by V_1 has at most one T". A node V_1 is connected to node V_2 via a *referencing edge* (a dashed arrow line) if V_1 references the entity represented by node V_2. A node V_1 is pointed by a *root edge* (a solid arrow line with no source node) tagged T if the entity type represented by V_1 is owned by the database.

The design of our example database is not a good one due to data redundancy. The two instances "$\ll$ &9 $\gg$" and "$\ll$ &14 $\gg$" in Figure 1 actually represent the same tutor "Tan" and his office "S16 05-13". If a tutor changes his office, then all occurences of this information must be changed to maintain consistency. [3] employs a decomposition method to restructure a S3-graph to remove anomalies. As in relational databases, the decomposition approach does not ensure a good solution.

2.2 The Entity-Relationship Approach

The ER model [1] incorporates the concepts of entity types, relationship sets and attributes which correspond to structures naturally occuring in information systems and database applications. An *entity type* is a collection of similar objects which have the same set of predefined common properties or attributes.

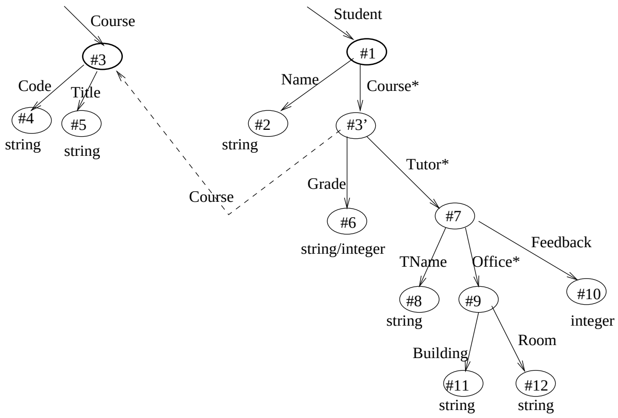

Fig. 2. Schema of the semistructured graph in Figure 1.

Attributes can be single-valued, multivalued or composite. A minimal set of attributes whose values uniquely identify an entity in an entity type is called a *key*. A *relationship* is an association among two or more entities. A *relationship set* is a collection of similar relationships with a set of predefined common attributes. If the existence of an entity in one entity type depends on the existence of a specific entity in another entity type, such a relationship set and entity type are called *existence dependent relationship set* and *weak entity type*. A relationship set which involves weak entity types is a weak relationship set. An entity type which is not weak is a regular entity type. If an entity in one entity type $E1$ is also in another entity type $E2$, we say that $E1$ ISA $E2$. The ER model also supports *recursive* relationships involving entities in the same entity type. Relationship sets can be viewed as a high level entity type known as *aggregation*.

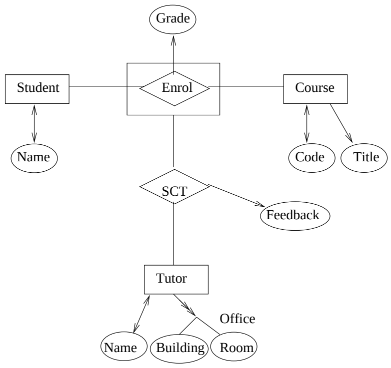

Fig. 3. Entity-Relationship diagram for Student-Course-Tutor example.

Figure 3 shows the ER diagram for the Student-Course-Tutor database. The relationship set Enrol captures the association that a student is enrolled in a course and has a single-valued attribute Grade. Since a student taking a course is taught by some tutors, we need to associate the relationship set Enrol with entity type Tutor. This association is captured in SCT where Enrol is viewed as an entity type participating in other relationship sets. The ER diagram is also in normal form.

3 ER to S3-Graph Translation

The task of designing a "good" semistructured database can be made easier if we have more semantics. For example, [4] proposes a normal form for the ER model and [5]uses the normal form ER to design normal form nested relations. This top-down approach has two advantages. First, normalizing an ER diagram removes ambiguities, anomalies and redundancies at the semantic level. Second, converting a normalized ER diagram into a set of nested relations results in a database schema with clean semantics, in a good normal form. An S3-graph is similar to the nested relations and hierarchical model in that they have a tree-like structure and allow repeating groups or multiple occurences of objects. All these models represent hierarchical organizations in a direct and natural way, but they are problematic when representing nonhierarchical relationships. Duplication of data is necessary when representing many-to-many relationships or relationships that involve more than two participating entity types. The problem of handling symmetric queries with data duplication also exists in semistructured data. Keys and foreign keys are used in nested relations while virtual pairing by logical parent pointers to represent many-to-many relationships are used in the hierarchical model to allow symmetric queries. In S3-graphs, we will show how referencing edges can be used to remove redundancies.

We will now describe the translation of an ER diagram to an S3-Graph. If the ER diagram is not in normal form, then the S3-Graph obtained may contain redundancy. However, if the ER diagram is in normal form, then the S3-Graph obtained will be in normal form (S3-NF). Anomalies in the semistructured database are removed and any redundancy due to many-to-many relationships and n-ary relationships are controlled.

Algorithm: Translation of an ER diagram to a S3-Graph.
Input: an ER diagram; Output: the equivalent S3-Graph
Step 1. Transform the ER diagram to a normal form ER diagram.
Step 2. Map regular entity types.
Step 3. Map weak entity types.
Step 4. Map relationship sets with no aggregations[1] .
Step 5. Map relationship sets with aggregations.
Step 6. Map ISA relationship sets.

[1] The participating entity types of a relationship set R may be aggregations or simply entity types.

Step 1. *Transform the ER diagram to a normal form ER diagram.*
Detailed steps with examples are described in [4].

Step 2. *Map regular entity types.*
Each regular entity type E becomes a root node N of an S3-Graph.

(a) Each single-valued and multivalued attribute A of E is mapped to a component node N_A connected to E by a component edge tagged A. If A is multivalued, then the tag is suffixed by an "*". N_A is also a leaf node labeled with the data type of A.

(b) Each composite attribute A of E is mapped to a component node N_A connected to E by a component edge tagged A. Each component attribute C of A is mapped to a component node N_C connected to N_A by a component edge with tag C.

Step 3. *Map weak entity types.*
Each weak entity type W becomes a component node N_W connected to the node N corresponding to its owner entity type E by a component edge tagged W suffixed by an "*". Attributes of W are mapped in the same way as the attributes of a regular entity type.

Step 4. *Map regular relationship sets with no aggregations.*

Case (1) *R is a binary relationship set.*
Let R be a binary relationship set with participating entity types E_A and E_B. E_A and E_B are mapped to root nodes N_A and N_B respectively. Depending on the application, there are several ways to map R:

(a) R is mapped to a component node N_R.
N_A is connected to N_R by a component edge tagged R_A and N_R connected to N_B by a referencing edge tagged R_B.

(b) R is mapped to a component node N_R.
N_B is connected to N_R by a component edge tagged R_B and N_R connected to N_A by a referencing edge tagged R_A.

(c) R is mapped to component nodes N_R and $N_{R'}$.
N_A is connected to N_R by a component edge tagged R_A and N_R connected to N_B by a referencing edge tagged R_B, while N_B is connected to $N_{R'}$ by a component edge tagged $R'_B$and $N_{R'}$ connected to N_A by a referencing edge tagged R'_A.

(d) R is mapped to a root node N_R.
N_R is connected to N_A and N_B by referencing edges tagged R_A and R_B respectively.

If R is a one-to-one relationship set, then all the tags of the component and referencing edges are not suffixed by an "*". If R is a many-to-many relationship set, then the tags are suffixed by an "*". If R is a one-to-many relationship set, then without loss of generality, let the cardinalities of E_A and E_B in R be 1 and m respectively. If we have N_A connected to N_R by a component edge, then the tag of the component edge is suffixed by an "*". Otherwise, if we have N_B connected to N_R by a component edge, then the tag is suffixed by an "*". Tags of component edges for $N_{R'}$ are similarly suffixed.

Attributes of R are mapped to component nodes in the same way as attributes of regular entity types. Figure 4 summarizes the mapping of binary relationship sets. Note that mappings (a) and (b) do not allow symmetric queries to be answered efficiently, while mapping (c) has controlled data redundancy. The relationship set R is "flattened" in mapping (d) to allow symmetric queries without duplication.
Note that if R is a recursive relationship set, then only one entity type, say E_A is involved in R with roles r1 and r2. The mapping for a recursive relationship set is similar to binary relationship sets except that the edges are tagged with the rolenames of E_A.

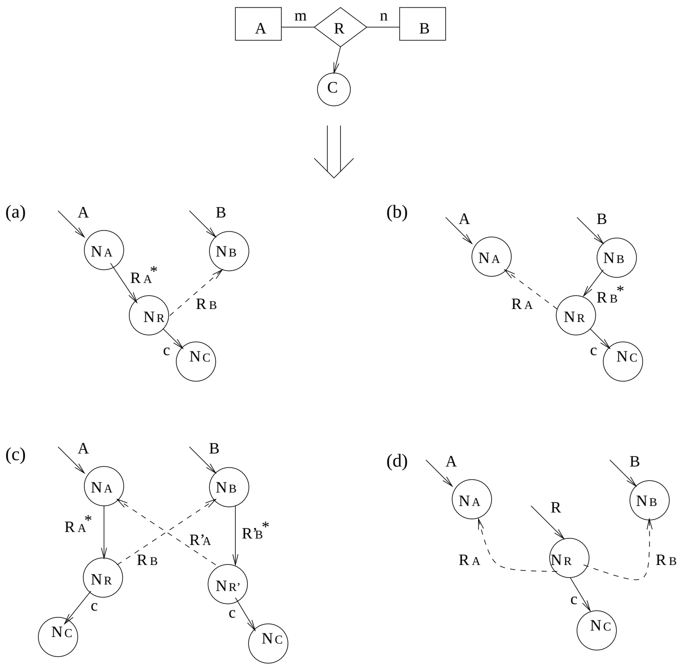

Fig. 4. Different ways to map a binary relationship set in an ER diagram to S3-graph

Case (2) *R is a n-ary relationship set where* $n > 2$.
Let the participating entity types of R be E_1, E_2, ..., E_m, where $m > 2$. E_1, E_2, ..., E_m are mapped to root nodes N_1, N_2, ..., N_m respectively. There are several ways to map R as shown in Figure 5.

(a) Map R to a component node N_R. Without loss of generality, connect N_1 to N_R by a component edge tagged R_{E1}. Then N_R has a referencing edge tagged $R_{E1}, R_{E2}, ..., R_{Em}$ to each of the root nodes N_2, N_3, ..., N_m respectively.

(b) First choose a path to link the participating entity types of R. Let $\prec V_1, V_2, V_3, \cdots, V_k \succ$ be the path, vertex V_1 corresponds to some participating entity type of R which is associated with some root node N_1, and vertex V_i, $2 \leq i \leq k$, corresponds to either a participating entity type of R or a combination of participating entity types of R. Next, create component nodes $N_{R2}, N_{R3}, ..., N_{Rk}$ that is associated with $V_2, V_3, ..., V_k$ respectively. Root node N_1 has a component edge tagged $R_{2_{E1}}$ to node N_{R_2}, while each node N_{R_i}, where $2 \leq i \leq k-1$, has a component edge tagged $R_{i+1_{E1}}$ to $N_{R_{i+1}}$, and a referencing edge(s) tagged R_{Ei} to the root node(s) that is associated with the participating entity type(s) of R corresponding to V_i.

(c) "Flatten" the relationship set by mapping R to a root node N_R. N_R is connected to N_1, N_2, ..., N_m by referencing edges tagged $R_{E1}, R_{E2}, ..., R_{Em}$ respectively.

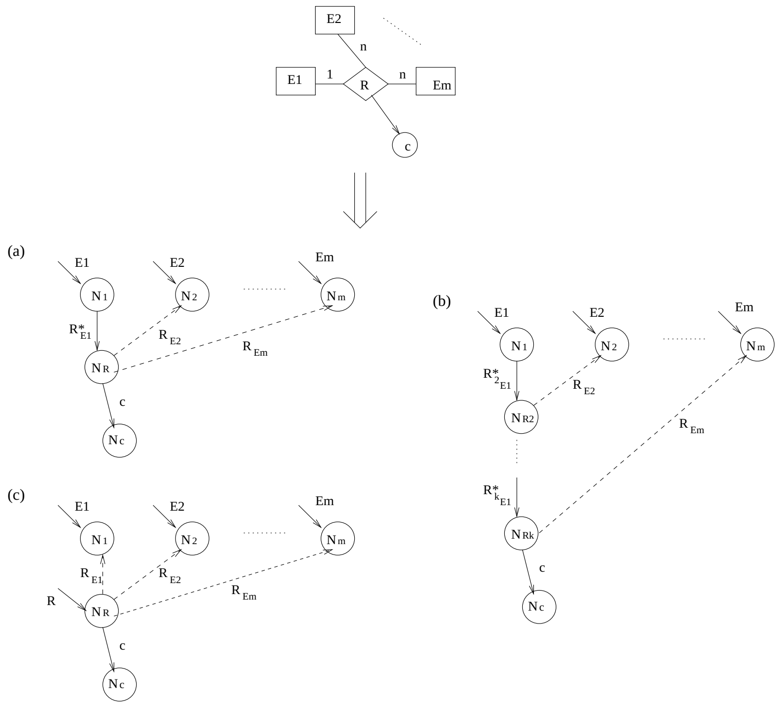

Fig. 5. Different ways to map a n-ary relationship set in an ER diagram to S3-graph

Step 5. *Map regular relationship sets with aggregations.*

Aggregations are a means of enforcing inclusion dependencies in a database. Let R be a regular relationship set and E_1, E_2, ..., E_m and E_{A_1}, E_{A_2}, ..., E_{A_n} be the participating entity types of R. Entity types E_1, E_2, ..., E_m have been mapped to root nodes N_1, N_2, ..., N_m respectively. Relationships in the aggregations E_{A_1}, E_{A_2}, ..., E_{A_n} have been mapped to nodes N_{A_1}, N_{A_2}, ...,

N_{A_n} respectively. Depending on the application, use the various alternatives in Step 4 to map R to a node N_R and link N_R to the nodes N_i and N_{A_j}, where $1 \leq i \leq m$, $1 \leq j \leq n$.

Step 6. *Map special relationship set ISA.*

Given A ISA B, map A and B to nodes N_A and N_B respectively and the ISA relationship set to a referencing edge tagged ISA connecting N_A to N_B.

Example. The ER diagram in Figure 3 can be translated to the semi-structured schema graph in Figure 6 as follows. The entity types Student, Course and Tutor become entity nodes #1, #3, #7 respectively. The attributes also become nodes and are connected to their owner entity type by component edges. We need to process the relationship Enrol before SCT because Enrol is involved in an aggregation. Enrol is mapped to an entity node #13 with component edges to entity nodes Course and Student. The attribute Grade is a component of the entity node #13. Next, we map the relationship set SCT to an entity node #15 which has component edges to entity nodes Enrol and Tutor. The attribute Feedback is a component of node #15. The S3-Graph obtained does not contain data redundancy. Note that the relationship sets Enrol and SCT have been flattened in the S3-Graph in Figure 6 because we want to answer symmetric queries with no redundant data. However, if an application only process queries which retrieve the courses taken by a student, and do not need to find students who take a given course, then Figure 7 shows an alternative way to map the ER diagram. Note that this schema cannot answer symmetric queries effectively.

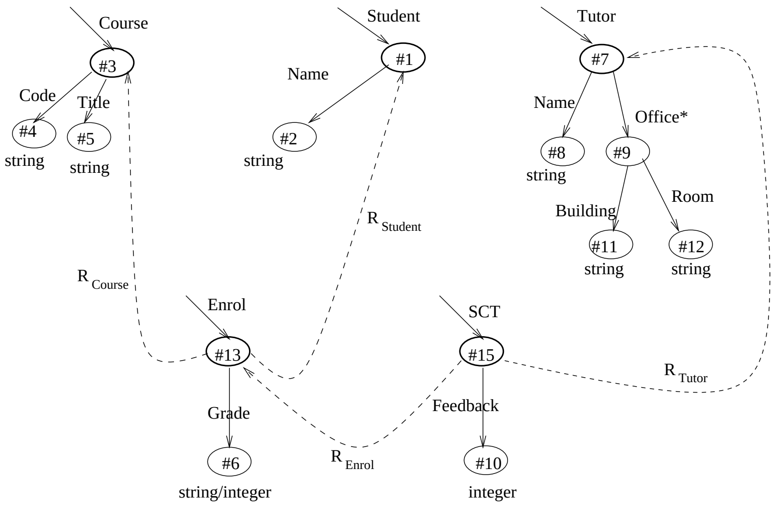

Fig. 6. An S3-Graph for the ER diagram in Figure 3

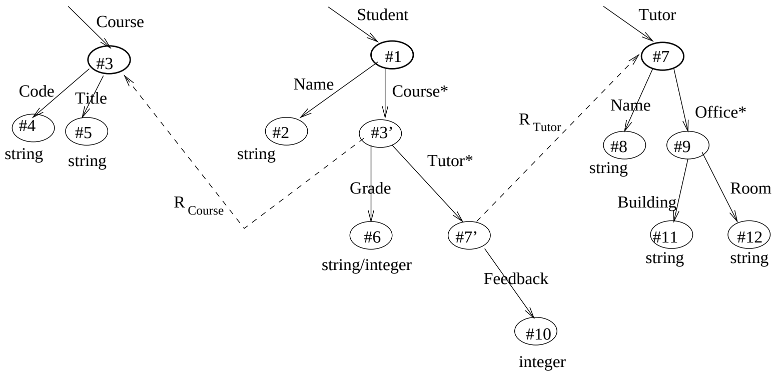

Fig. 7. An alternative S3-Graph for the ER diagram in Figure 3

4 Conclusion

To the best of our knowledge, this is the first paper that presents a conceptual approach for designing semistructured databases which can be associated with some schema. We envisage the growing importance of well designed semistructured databases with the development of new e-commerce applications that require the efficient design and maintenance of large amounts of data. The introduction of an ER-based conceptual layer allows us to remove anomalies and data redundancies at the semantic level. We have developed an algorithm to map an ER diagram involving weak entity types, recursive, n-ary and ISA relationship sets, and aggregations to a normal form S3-Graph. Using the mappings proposed, XML DTDs and customised XML views can be generated from the normal form ER diagrams. Relational tables can also be created from the normalized ER diagram to store the XML data with controlled or no redundancy.

References

1. P.P. Chen. The ER model: Toward a unified view of data. *ACM Transactions on Database Systems*, Vol 1, No 1, 1976.
2. Document Object Model(DOM). *http://www.w3.org/TR/REC-DOM-Level-1*.
3. S.Y. Lee, M.L. Lee, T.W. Ling, and L. Kalinichenko. Designing good semistructured databases. In *Proc. of 18th Int. Conference on ER Approach*, 1999.
4. T.W. Ling. A normal form for entity-relationship diagrams. In *Proc. of 4th Int. Conference on Entity-Relationship Approach*, pages 24–35, 1985.
5. T.W. Ling. A normal form for sets of not-necessarily normalized relations. In *Proc. of 22nd Hawaii Int. Conference on Systems Science*, pages 578–586, 1989.
6. G. Mecca, P. Merialdo and P. Atzeni. Araneus in the Era of XML. *IEEE Bulletin on Data Engineering*, 1999.
7. Y. Papakonstantinou, H. Garcia-Molina, and J. Widom. Object exchange across heterogeneous information sources. In *IEEE Int. Conference on Data Engineering*, 1995.