Survey of Graph Database Models

Renzo Angles , Claudio Gutierrez
{rangles,cgutierr}@dcc.uchile.cl

Technical Report Number TR/DCC-2005-10
Computer Science Department
Universidad de Chile

"A la memoria de Alberto Mendelzon"

Abstract

Graph database models can be characterized as those where data structures for the schema and instances are modeled as graphs or generalizations of them, and data manipulation is expressed by graph-oriented operations and type constructors. These models flourished in the eighties and early nineties in parallel to object oriented models and their influence gradually faded with the emergence of other database models, particularly the geographical, spatial, semistructured and XML.

Recently, the need to manage information with inherent graph-like nature has brought back the relevance of the area. In fact, a whole new wave of applications for graph databases emerged with the development of huge networks (*e.g.* Web, geographical systems, transportation, telephones), and families of networks generated thanks to the automation of the process of data gathering (*e.g.* social and biological networks).

The main objective of this survey is to present in a single place the work that has been done in the area of graph database modeling, concentrating in data structures, query languages and integrity constraints.

Contents

1 Introduction

The term data model has been used in the information management community with different meanings and in diverse contexts. In its most general sense, a data[base] model (*db-model*)[1] is a concept that describes a collection of conceptual tools for representing real-world entities to be modeled and the relationships among these entities [116]. Often this term denotes simply a collection of data structure types, or even a mathematical framework to represent knowledge [90].

From a database point of view, the conceptual tools defining a db-model should address at least the structuring and description of the data, its maintainability and the form to retrieve or query the data. According to these criteria, a db-model is defined as a combination of three components, first a collection of data structure types, second a collection of operators or inference rules and third a collection of general integrity rules [31]. Note that several proposals of db-models define only the data structures, omitting sometimes operators and/or integrity rules.

Due to the importance of modeling conceptually, philosophically and in practice, db-models have become essential abstraction tools. Among the purposes of a db-model are: Tool for specifying the kinds of data permissible; general design methodology for databases; coping with evolution of databases; development of families of high level languages for query and data manipulation; focus in DBMS architecture; vehicle for research into the behavioral properties of alternative organizations of data [31].

Since the emergence of database management systems, there has been an ongoing debate about what the db-model for such a system should be. The evolution and diversity of existent db-models show that there is no silver bullet for data modeling. The parameters influencing their development are manifold, and among the most important we can mention the characteristics or structure of the domain to be modeled, the type of intellectual tools that appeals the user, and of course, the hardware and software constraints imposed. Additionally, each db-model proposal is grounded on certain theoretical tools, and serves as base for the development of related models. Figure 1 sketches these influences.

[1] In the database literature the terms "data model" and "database model" (and sometimes even "model") usually denote the same concept. In the scope of this survey we will consider them as synonyms and use the abbreviated expression db-model.

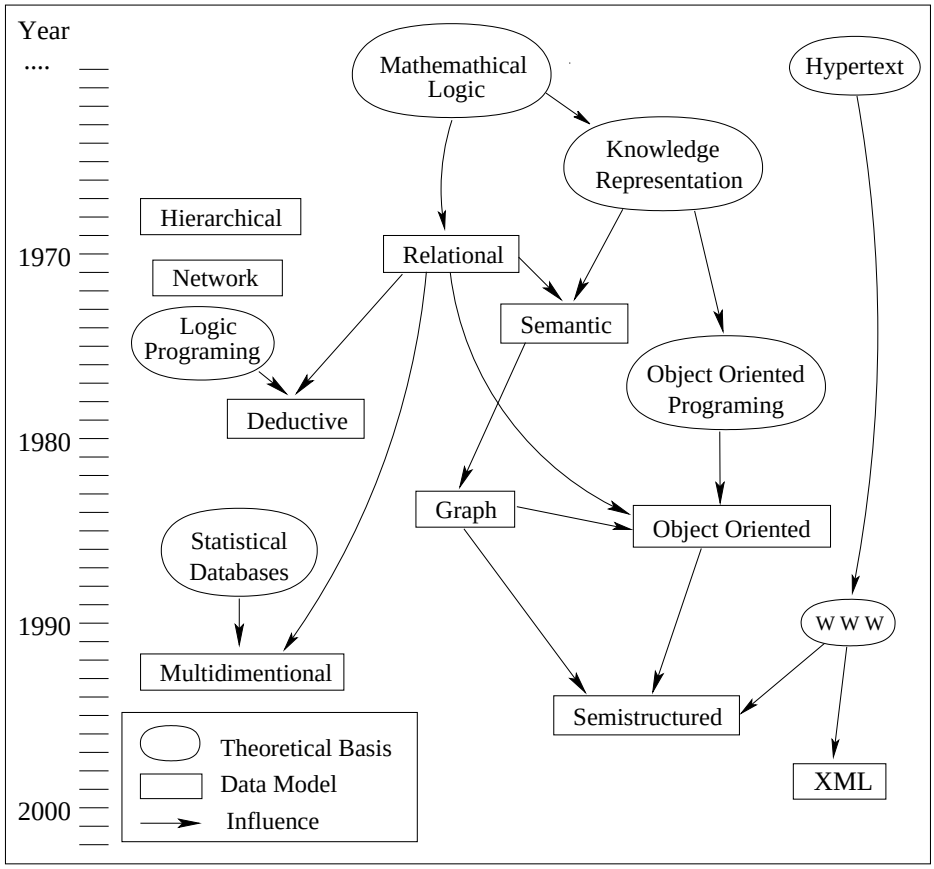

Figure 1: Evolution of database models. Rectangles denote models, arrows indicate influences, and circles denote theoretical developments. On the left hand side a time-line in years. (After a diagram by A. O. Mendelzon.)

Surveys and taxonomies of db-models are as manifold as db-models themselves (*e.g.* [116, 93, 17, 72]). Following we briefly describe the most representative and widely accepted and/or used db-models which are of general purpose, *i.e.*, although best suited for particular kinds of data, do not have any particular application in mind.

1.1 Database Models Evolution – Brief Historical Overview

In the beginnings of the design of db-models, physical (hardware) constraints were one of the fundamental parameters to be considered. Before the advent of the relational model, most db-model focused essentially in the specification of the structure of data in actual file systems. Kerschberg et al.c̃ite50130 developed a taxonomy of db-models prior to 1976, comparing essentially their mathematical structures and foundation, and the levels of abstraction used.

Two representative db-models are the *hierarchical* [126] and *network* [122] models, which emphasize the physical level, and offer the user the means to navigate the database at the record level, thus providing low level operations to derive more abstract structures.

The relational db-model was introduced by Codd [30, 32] and highlights the concept of level of abstraction by introducing the idea of separation between physical and logical levels. It is based on the notions of sets and relations. Due to its simplicity of modeling, it gained a wide popularity among business applications.

Semantic db-models [100] allow database designers to represent objects and their relations in a natural and clear manner to the user (as opposed to previous models). They intended to provide the user with tools that could capture faithfully the semantics of the information to be modeled. A well-known example is the entity-relationship model [28].

Object oriented db-models [75] appeared in the eighties, when most of the research was concerned with so called "advanced systems for new types of applications [17]. These db-models are based on the object-oriented paradigm and their goal is representing data as a collection of objects that are organized in classes and have complex values associated with them.

Semistructured db-models [23] are designed to model data with a flexible structure, *e.g.*, documents and Web pages. Semistructured data (also called unstructured data) is neither raw nor strictly typed as in conventional database systems. Additionally, data is mixed with the schema, a feature which allows extensible exchange of data. These db-models appeared in the nineties and are currently in evolution.

The *XML (eXtended Markup Language)* [21] model did not originate in the database community. Although originally introduced as a standard to exchange and model documents, soon it became a general purpose model, with focus on information with tree-like structure. Similar to semistructured model, scheme and data are mixed. See Section 2.3 for a more in depth comparison among these models.

Other Models and Frameworks. There are other important db-models designed for particular applications, as well as modeling frameworks not directly focusing in database issues, which indirectly concern graph database modeling. Among the db-models are Spatial databases [43, 98, 109], Geographical Information Systems (GIS) [112, 15], Temporal db-models [121, 29], Multidimensional db-models [130]. Frameworks related to our topic, but not directly focusing in database issues are Semantic Networks [119, 107, 56], Conceptual Graphs [117, 118], and Knowledge Representation Systems: G-Net Model [40], Topics Maps [101, 102, 1, 89], Hypertext [33]. Due to the size limitations of this survey they are not covered here.

1.2 Graph Database Models – Brief Historical Overview

The notion of *graph db-model* made its appearance almost in parallel with the object oriented db-models, as an alternative to the limitations of traditional db-models for capturing the inherent graph structure of data appearing in applications such as hypertext or geographic database systems, where the interconnectivity of data is an important aspect.

Activity around graph databases flourished in the first half of the nineties and then the topic almost disappeared. The reasons for this decline are manifold: the database community moved toward semistructured data (a research topic which did not have links to the graph database work in the nineties); the emergence of XML captured all the attention of the work on hypertext; people working on graph databases moved to particular applications like spatial data, web, documents; the tree-like structure is enough for most applications. Figure 2 reflects this evolution by means of papers published in main conferences and journals.

Graph db-models emerged with the objective of modeling information whose structure is a graph. In an early approach, Roussopoulos and Mylopoulos [107] facing the failure of current (at the time) systems to take into account the semantics of the database, proposed a semantic network to store data about the database. An implicit structure of graphs for the data itself was presented in the Functional Data Model [115], whose goal was to provide a "conceptually natural" database interface. A different approach proposed the Logical Data Model [79], where an explicit graph db-model intended to generalize the relational, hierarchical and network models. Years later Kunii [78] proposed a graph db-model for representing complex structures of knowledge called G-BASE.

In the late eighties, Lécluse et al. [82] introduced O_2, an object oriented db-model based on a graph structure. On the same lines, GOOD [59] is an

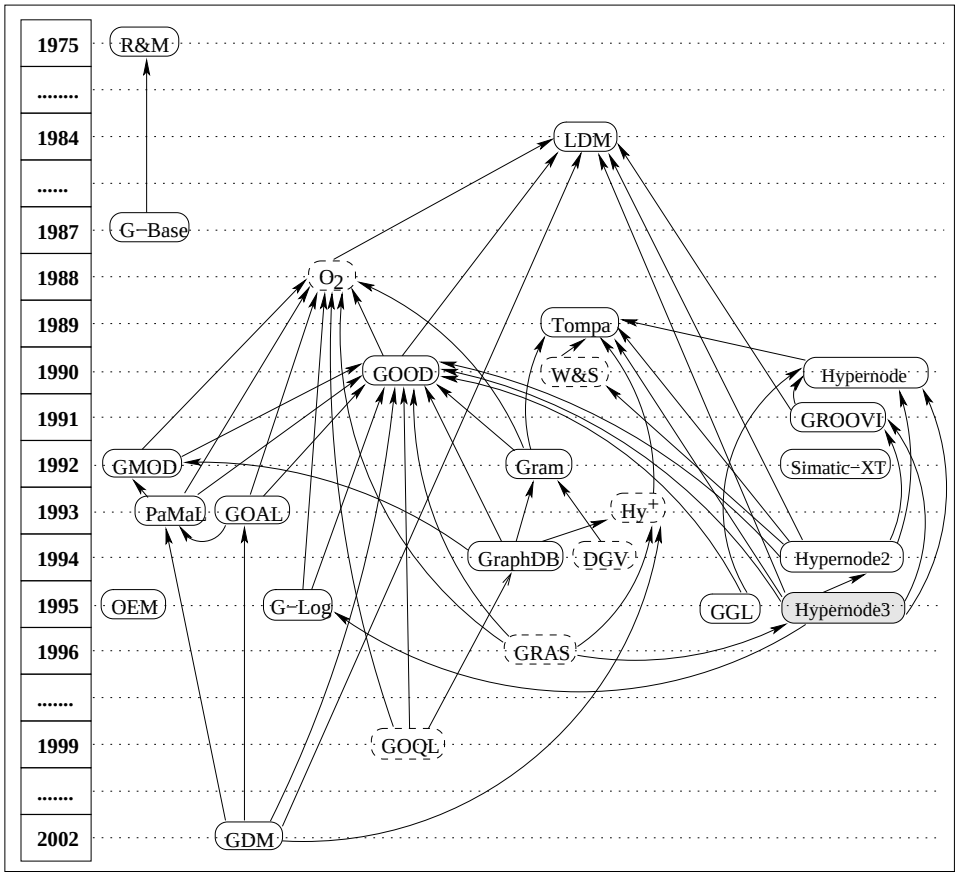

Figure 2: Graph db-models development. Nodes indicate models and arrows citations. Dashed nodes represent related works in graph db-models. DGV [57], GDM [68], GGL [54], GMOD [13], GOAL [69], GOOD [59, 60], GOQL [113], Gram [11], GRAS [73], GraphDB [58], GROOVY [85], G−Base [78], G−Log [99], Hypernode [84], Hypernode2 [104], Hypernode3 [83], Hy+ [34], LDM [79, 80], OEM [97], O_2 [82], PaMaL [48], R&M [107], Simatic-XT [86], Tompa [125], W&S [132].

influential graph-oriented object model, intended to be a theoretical basis for a system in which manipulation as well as representation are transparently graph-based. Among the subsequent developments based on GOOD are: GMOD [13] that proposes a number of concepts for graph-oriented database user interfaces; Gram [11] which is an explicit graph db-model for hypertext data; PaMaL [48] which extends GOOD with explicit representation of tuples and sets; GOAL [69] that introduces the notion of association nodes; G-Log [99] which facilitates working with end user interfaces; and GDM [68] that incorporates features from object-oriented, Entity-Relationship and Semistructured models.

There were proposals that used generalization of graphs with data modeling purposes. Levene and Poulovassilis [84] introduced a db-model based on nested graphs, called the Hypernode Model, on which subsequent work was developed [104, 83]. The same idea was used for modeling multi-scaled networks [86] and genome data [54]. GROOVY [85] is an object oriented db-model which is formalized using hypergraphs. This generalization was used in other contexts: query and visualization in the Hy+ system [34]; modeling of data instances and access to them [132]; representation of user state and browsing [125];

There are several other proposals that deal with graph data models. With a motivation coming from managing information in transport networks, Güting proposed the model GraphDB [58] intended for modeling and querying graphs in object-oriented databases. Another general purpose project is Database Graph Views [57], that proposes an abstraction mechanism to define and manipulate graphs stored in either relational object oriented or file systems. The project GRAS [73] uses attributed graphs for modeling complex information from software engineering projects. Finally, the well known OEM [97] model aims at providing integrated access to heterogeneous information sources, focusing in information exchange.

1.3 Scope and Organization of this Survey

The objective of this survey is to present in a single place the work that has been done in the area of graph database modeling. We stress the fact that the goal of the survey, rather than making a balance of the area, is to present in a comprehensive way the different developments and relevant pointers to facilitate a researchers to go to the sources. This obvious goal of a survey is highlighted in our case by the fact that the area has been overlooked and lately is quickly gaining relevance.

We concentrate in presenting the main aspects of modeling, that is, data structures, query languages and integrity constraints. Considering that the area has not yet an identity by itself, we spend Section 2 surveying different existing views on graph db-models, and the motivations and application that drive this developments. Section 3 surveys the most relevant graph db-models and describe in detail each of them.

There is a substantial amount of work dealing with query languages and graph interfaces. In fact, we think that transformation and query languages for graphs are topics that deserves a thorough survey by itself. On the other hand, not all the db-models treated in Section 3 consider the topic of constraints, and several of them do not define a proper query language. Section 4 gives a brief overview of these topics with the sole purpose of giving a flavor of the area while a survey comes up.

We would like to warn the reader that there are several related areas that fall out of the scope of this survey. Among the most important we can mention graph visualization, graph data structures and algorithms for secondary memory, graph methods for databases, and in general graph database system implementation. Table 19 indicates which models covered in this survey were implemented.

2 Graph Data Modeling

2.1 What is a Graph Data Model?

Although almost all papers on db-models cited in the previous section use the term "graph data[base] model", few of them define the notion explicitly. Nevertheless their views on what a graph db-model is do not differ substantially. Usually the implicit definition is given by comparing the model against other models where graphs are involved, like the semantic, object-oriented and semi-structured models.

In what follows we will conceptualize the notion of graph db-model according to the three basic components of a db-model, namely data structures, transformation language, and integrity constraints. A graph db-model is characterized by:

- *The data and/or the schema are represented by graphs*, or by data structures generalizing the notion of graph (hypergraphs, hypernodes, hygraphs, etc.). Almost everybody coincide on this point modulo slight variations.

  Let us review different wordings of authors on this issue. The approach is to model the database directly and entirely as a graph [58]. A graph db-model is one whose single underlying data structure is a labeled directed graph; the database consists of a single digraph [83]. A database schema in this model is a directed graph, where leaves represent data and internal nodes represent connections between the data [79]. Directed labeled graphs are used as the formalism to specify and represent database schemes, instances, and rules [99]. The model is basically defined as a labeled directed graph. In this model, a database is described in terms of a labeled directed graph called schema graph [78]. A graph db-model formalizes the representation of the data structures stored in the databases as a graph [54]. The schema as well as the instance of an object database is represented by a graph. The nodes of the instance graph represent the objects of the database [59]. Database instances and databases schemes are described by certain types of labeled graphs [68]. The model for data is organized as graphs [11]. Labeled graphs are used to represent schemes and instances [69].

On top of these descriptions, one could add the fact that sometimes the schema and the data (instances) are difficult to differentiate in these models, a fact that resembles closely semi-structured models. But in most cases the schema and the instances are separated.

- *Data manipulation is expressed by graph transformations* [59], or by operations whose main primitives address directly typical features of graphs, like paths, neighborhoods, subgraphs, graph patterns, connectivity, and statistics about graphs (diameter, centrality, etc.). The db-model defines a flexible collection of type constructors and operators which create and access the graph data structures [54], or in other terms, the approach is to express all queries in terms of a few powerful graph manipulation primitives [58]. The operators of the language can be based on pattern matching, *i.e.* finding of all occurrences of a prototypical piece of an instance graph [69].

- The existence of *integrity constraints* enforcing the consistency of the data, which are directly related to the graph data structure. For example, labels with unique names [55], typing constraints on nodes [80], functional dependencies [85], domain and range of properties [76].

Summarizing, a graph db-model is a model where the data structures for the schema and/or instances are modeled as a (labeled)(directed) graph, or generalizations of the graph data structure, where data manipulation is expressed by graph-oriented operations and type constructors, and has integrity constraints appropriate for the graph structure.

2.2 Why a Graph Data Model?

The application areas of graph db-model models are those were information about the interconnectivity or the topology of the data is more important, or as important as, the data itself. This is usually accompanied by the fact that data and relations among data are at the same level. In fact, introducing graphs as a modeling tool has several advantages for this type of data.

First, it leads to a more natural modeling: graph structures are visible to the user. They allow a natural way of handling data appearing in applications (*e.g.* hypertext or geographic databases). Graphs have an important advantage: they can keep all the information about an entity in a single node and show related information by arcs connected to it [99]. Graph objects (like paths, neighborhoods) may have first order citizenship; a user

Type of Model	Abstract. level	Base data structure	Main Focus	Data complex. homogeneity.
Network	physical	point + rec.	records	simple/hom.
Relational	logical	relations	data/attributes	simple/hom.
Semantic	user	graphs	schema/relations	medium/hom.
Object-O	logical/physical	objects	object/methods	high/het.
Semistruct.	logical	tree	data/components.	medium/het.
Graph	logical	graph	data/relations	medium/het

Table 1: A coarse-granularity comparative view among different general-purpose database models. The parameters are: abstraction level, base data structure used, what are the types of information objects the db-model focus in, complexity and homogeneity of the data items modeled.

can define some part of the database explicitly as a graph structure [58], allowing encapsulation and context definition [84].

Second, queries can refer directly to this graph structure. Associated with graphs are specific graph operations in the query language algebra, such as finding shortest paths, determining certain subgraphs, and so forth. Explicit graphs and graph operations allow a user to express a query at a very high level. To some extent, this is in contrast to graph manipulation in deductive databases, where often fairly complex rule programs need to be written [58]. Last but not least, for purposes of browsing it may be convenient to forget the schema [24].

Third, as far as implementation is concerned, graph databases may provide special storage graph structures for the representation of graphs and the most efficient graph algorithms available for realizing specific operations [58]. Although the data may have some structure, the structure is not as rigid, regular or complete as traditional DBMS. It is not important to require full knowledge of the structure to express meaningful queries [4]. The system can use efficient graph algorithms designed to utilize the special graph data structures [58].

2.3 Comparison with other Database Models

In this section we compare the most influential db-models with graph db-models. Table 1 presents a coarse granularity overview of the most influential models. Below we present the details.

Physical db-models. They were the first ones to offer the possibility to organize large collections of data. Among the most important ones are the hierarchical [126] and network [122] models. These models lack good abstraction level and are very close to physical implementations. The data-structuring is not flexible and not apt to model non-traditional applications. For our discussion they do not have much relevance.

Relational db-model [30, 32] was introduced by Codd to highlight the concept of level of abstraction by introducing a clean separation between physical and logical levels. Gradually the focus shifted to modeling data as seen by applications and users [93]. This is the emphasis and the achievement of the relational model, in a time where the domain of application were basically simple data (banks, payments, commercial and administrative applications).

The relational model was a landmark development because it provided a mathematical basis to the discipline of data modeling. It is based on the simple notion of relation, which together with its associated algebra and logic, made the relational model a primary model for database research. In particular, its standard query and transformation language, SQL, became a paradigmatic language for querying.

The differences between graph db-models and the relational db-model are manifold. Among the most relevant ones are: the relational model was directed to simple record-type data with a structure known in advance (airline reservations, accounting, inventories, etc.). The schema is fixed and extensibility is a difficult task. Integration of different schemes is not easy nor automatizable. The query language does not support paths, neighborhoods and several other graph operations, like connectivity (an exception is transitivity). There are no objects identifiers, but values.

Semantic db-models [100] have their origin in the necessity to provide more expressiveness and incorporate a richer set of semantics into the database from the user point of view. They allow database designers to represent objects and their relations in a natural and clear manner (similar to the way the user view an application) by using high-level abstraction concepts such as aggregation, classification and instantiation, sub- and super-classing, attribute inheritance and hierarchies [93]. A well-known example is the entity-relationship model [28]. It has become a basis for the early stages of database design, but due to lack of preciseness cannot replace models like relational or Object Oriented. Other examples of semantic db-models are IFO [3] and SDM [63]. For graph db-models research, semantic db-models are relevant because they are based on a graph-like structure which highlights the relations between the entities to be modeled.

Object oriented (O-O) db-models [75] appeared in the eighties, when the database community realized that the relational model was inadequate for data intensive domains (Knowledge base, engineering applications). O-O databases were motivated by the emergence of non-conventional database applications consisting of complex objects systems with many semantically interrelated components as in CAD/CAM, computer graphics or information retrieval. According to the O-O programming paradigm on which they are based, their objective is representing data as a collection of objects that are organized in classes and have complex values and methods associated with them. Although O-O db-models permit much richer structures than the relational db-model, they still require that all data conform to a predefined schema [4].

O-O db-models have been related to graph db-models due to the explicit or implicit graph structure in their definitions [85, 13, 59]. Nevertheless, there remain important differences rooted in the form that each of them models the world. O-O db-models view the world as a set of complex objects having certain state (data) and interacting among them by methods. On the contrary, graph db-models, view the world as a network of relations, emphasizing the interconnection of the data, and the properties of these relations. The emphasis of O-O db-models is on the dynamics of the objects, their values and methods. In contrast, graph db-models emphasizes the interconnection while maintaining the structural and semantic complexity of the data. A detailed comparison between these db-models may be founded in [17, 72, 93, 116].

Semistructured db-models [23, 2]. The need for semistructured data (also called unstructured data) was motivated by: the increased existence of unstructured data, data exchange and, data browsing [23]. In semistructured data the structure is irregular, implicit and partial; the schema does not restrict the data, only describes it, is very large and rapidly evolving; the information associated with a schema is contained within the data (data contains data and its description, so it is self-describing) [2]. Among the most representative models are OEM [97], Lorel [4], UnQL [24], ACeDB [120] and Strudel [44]. Generally, semistructured data is represented by a tree-like structure. Nevertheless cycles between data are possible, establishing in this way a structural relation with graph db-models. Some authors characterize semistructured data as rooted directed connected graphs [24].

2.4 Graph Data Model Motivations and Applications

Graph db-models are motivated by real-life applications where information about interconnectivity of its pieces is a salient feature. We will divide these application areas in Classical and Complex networks.

Classical Applications. The applications that motivated the introduction of the notion of graph databases were manifold:

1. Generalizations of classical db-models [79]. Classical models were criticized for their lack of semantics, the flat structure of the data they allow, the difficulties for the user to "see" the connectivity of the data, and the difficult to model complex objects [84].

2. On the same direction, the observation that graphs have been integral part of the database design process in semantic and object-oriented db-models, brought the idea of introducing a model in which both, data manipulation and data representation were graph based [59].

3. Limitations of expressive power of languages for complex applications motivated also the search for models that resemble more closely such applications [99].

4. Limitations (at the time) of knowledge representation systems [78], and the need for intricate but flexible knowledge representation and derivation techniques [99].

5. The need for improving functionalities of object-oriented db-models [104]. In this direction the application in mind were CASE, CAD, image processing, and scientific data analysis.

6. Graphical and visual interfaces, geographical, pictorial and multimedia systems [61, 34, 113].

7. Applications where data complexity exceeded the relational db-model capabilities also motivated graph databases. For instance, managing transport networks (train, plane, water, telecommunications) [87], spatially embedded networks like highway, public transport [58]. Several of these applications are now in the field of Geographical information systems and spatial databases.

8. There are other applications who motivated graph db-models: software systems, integration [73].

9. Lately, the emergence of hypertext on-line made evident the need for other db-models [125, 132, 11]. Together with hypertext, the Web created the need for a model more apt than classical ones for information exchange. This was one of the main motivation of semistructured models.

Complex Networks. Several areas have witness the emergence of huge networks of data which share some particular mathematical parameters, called complex networks [95, 9, 42]. The need for database management for some classes of these networks has been recently highlighted [96, 71, 127, 55]. Although it is not evident yet if from the point of view of databases one can treat them as a whole, we will describe them together for presentation purposes. After the survey of Newman [95], we will group them in four categories: social networks, information networks, technological networks and biological networks. Following we describe specific examples for each of them.

In *social networks* [64], nodes are people and groups while links show relationships or flows between the nodes. Some examples are friendship, business relationships, patterns of sexual contacts, research networks (collaboration, co-authorship), communication records (mail, telephone calls, email), Computer networks [134], National security [114]. There is growing activity in the area of Social Network analysis [20], visualization and data processing in such networks.

In *information networks* occur relations such as citations between academic papers [39], World Wide Web (hypertext, hypermedia) [47, 77, 22], peer-to-peer networks [94], relations between word classes in a thesaurus, preference networks.

In *technological networks* the structure is mainly governed by space and geography. Some examples are Internet (as network of computers), Electric power grids, airline routes, telephone networks, delivery network (post office). The area of *Geographic Information Systems (GIS)* is today covering a big part of this area (roads, railways, pedestrian traffic, rivers) [112, 91].

Biological networks represent biological information whose volume, management and analysis has become an issue due to the automation of the process of data gathering. Good example is the area of *Genomics*, where networks occur in gene regulation, metabolic pathways, chemical structure, map order and homology relationships between species [51]. There other kinds of biological networks, such as food webs, neural networks, etc. The area has a tremendous growth-rate. The reader can consult database proposals for genomics [55, 51, 62], an overview of models for biochemical path-

ways [41], a tutorial on Graph Data Management for Biology [96], and a model for Chemistry [18].

It is important to stress that classical query languages offer little help when dealing with the type of query needed in the above areas. As examples, data processing in GIS include geometric operations (area or boundary, intersection, inclusions, etc), topological operations (connectedness, paths, neighbors, etc) and metric operations (distance between entities, diameter of the network, etc). In genetic regulatory networks examples of measures are connected components (interactions between proteins) and degrees of nearest neighbors (strong pair correlations). In social networks, distance, neighborhoods, clustering coefficient of a vertex, clustering coefficient of a network, betweenness, size of giant connected components, size distribution of finite connected components [42]. Similar problems arise in the Semantic Web, where querying RDF data increasingly needs graph features [14].

3 Representative Graph Database Models

In this section we describe in some detail the most representative graph db-models, choosing those that define and use explicitly graph structures or generalizations of them. Additionally we describe other related models that use graphs, do not fit properly as graph db-models. In them, graphs are used, for example, for navigation, for defining views, or as language representation.

For each proposal, we present their data structures and, when available, their query languages and integrity constraint rules. In general, there are few implementations and no standard benchmarks, hence we avoid surveying this issue. For information about the existence of implementations see Figure 19. To give a flavor of the modeling in each proposal, we will run the following example about a toy genealogy shown in Figure 3.

NAME	LASTNAME
George	Jones
Ana	Stone
Julia	Jones
James	Deville
David	Deville
Mary	Deville

PERSON	PARENT
Julia	George
Julia	Ana
David	James
David	Julia
Mary	James
Mary	Julia

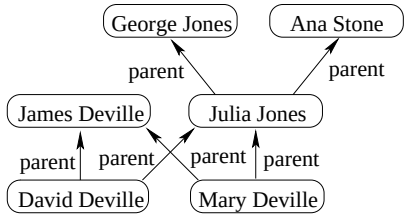

Figure 3: A genealogy diagram (right-hand side) represented as two tables (left-hand side) NAME-LASTNAME and PERSON-PARENT. (Children inherit the lastname of the father just for modeling purposes.)

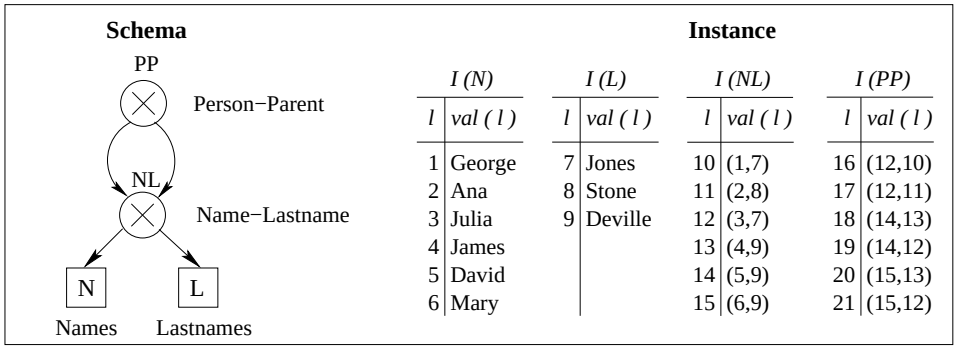

Figure 4: Logical Data Model. The schema (on the left) uses two basic type nodes for representing data values (N and L), and two product type nodes (NL and PP) to establish relations between data values in a relational style. The instance (on the right) is a collection of tables, one for each node of the schema. Note that internal nodes use pointers (names) to make reference to basic and set data data values defined by other nodes.

3.1 Logical Data Model (LDM)

Motivated by the lack of semantics in the relational db-model, Kuper and Vardi [79] proposed a db-model that generalizes the relational, hierarchical and network models. The model describes mechanisms to restructure data, a logical query language and an algebraic query language.

In LDM a *schema* is an arbitrary directed graph where each node has one of the following types: The *Basic* type $\square$ describes a node that contains the data stored; the *Composition* type TEX describes a node that contains tuples whose components are taken from the children of it; the *Collection* type $\bigcirc$ describes a node that contains sets, whose elements are taken from children of it. Summarizing, internal nodes are of type $\otimes$ or $\circledast$ representing structured data, terminal nodes are of type $\square$ and represent atomic data, and edges represent connections between data.

A second version of the model [80], besides renaming the nodes $\otimes$ and $\circledast$ as product and power respectively, incorporates a new type, the *Union* type $\textcircled{\cup}$, intended to represent a collection whose domain is the union of the domains of its children (see example in Figure 4).

A LDM database *instance* consists of an assignment of values to each node of the schema. In this sense, the instance of a node is a set of elements from the underlying domain (for basic type nodes) and tuples or sets taken from the instance of the node's children (for $\otimes$, $\circledast$ and types).

With the objective of avoiding cyclicity at the instance level, the model proposes to keep a distinction between memory locations and their content. Thus, instances consist of a set of *l-values* (the address space), plus an *r-value* (the data space) assigned to each of them. These features allow to model transitive relations like hierarchies and genealogies.

Over this structure a first order many-sorted language is defined. With this language, a query language and integrity constraints are defined. Finally, and algebraic language –equivalent to the logical language– is proposed, providing operations for node and relation creation, transformation and reduction of instances, and other operations like union, difference and projection.

LDM is a complete db-model (*i.e.* data structures plus query languages and integrity constraints) The model supports modeling of complex relations (*e.g.* hierarchies, recursive relations). The notion or virtual records (pointers to physical records) proves useful to avoid redundancy of data by allowing cyclicity at the schema and instance level. Due to the fact that the model is a generalization of other models (like the relational model), their techniques or properties can be translated into the generalized model. A relevant example is the definition of integrity constraints.

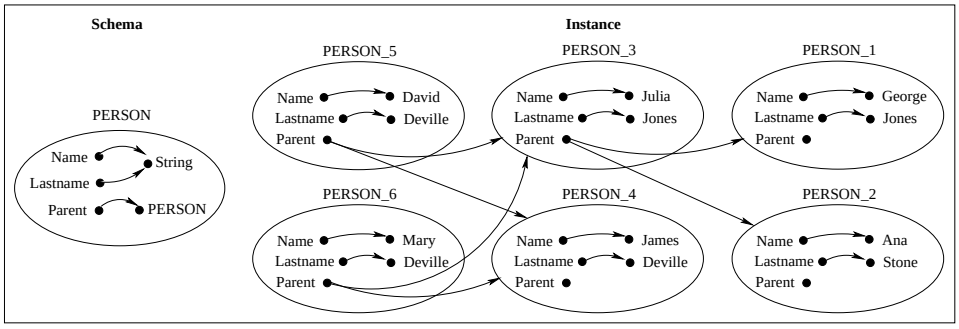

Figure 5: Hypernode Model. The schema (left) defines a *person* as a complex object with the properties *name* and *lastname* of type string, and *parent* of type person (recursively defined). The instance (on the right) shows the relations in the genealogy among different instances of person.

3.2 Hypernode Model

The Hypernode db-model was described in a sequence of papers [84, 104, 83]. An hypernode is a data structure allowing nesting of graphs: is a direct graph whose nodes can themselves be graphs (or hypernodes). Hypernodes can be used to represent simple (flat) and *complex objects* (hierarchical, composite, and cyclic) as well as mappings and records. A key feature is its inherent ability to *encapsulate information*. An example is presented in Figure 5.

The hypernode model was introduced Levene and Poulovassilis [84], who define the model and a declarative logic-based language structured as sequence of instructions ("hypernode programs"), used for querying and updating hypernodes. The implementation of a storage system based on the hypernode model is presented in [128].

In a second version [104], the notion of schema and type checking is introduced via the idea of types (primitive and complex), that are also represented by nested graphs. The model is completed with entity and referential integrity constraints over an hypernode repository. Moreover presents a rule-based query language called *Hyperlog*, which can support both querying and browsing with derivations and database updates.

A third version of the model [83] discusses a set of constraints (entity, referential and semantic) over hypernode databases and introduces the concept of *Hypernode functional dependency (HDF)*, denoted by $A \rightarrow B$, where A and B are sets of attributes, and the set of attributes A determines the value of the set of attributes B in all hypernodes of the database. In addition it presents another query and update language called HNQL, which

use compounded statements to produce HNQL programs.

The Hypernode is a complete db-model. It has a unique basic data structure which is simple and extensible allowing different levels of abstraction (nesting levels) and modularity. It allows representation of flat, hierarchical, composite, and cyclic objects, as well as functions (mappings) and relations (records). Has a simple representation of multi-valued attributes. Has a simple and intuitive representation of nested and composition relations, in the form of complex objects and sets of objects (objects represented as hypernodes). The hypernode model can also be regarded an object-oriented db-model supporting object identity (unique labels), complex objects, encapsulation (nesting of graphs), inheritance (structural), query completeness and persistence.

On the less positive aspects, there are some issues that deserve mention. Redundancy of data that can be generated by its basic value labels. The restrictions in the scheme level are limited, for example the specification of restrictions for missing information or multivalued relations is not possible. Nesting levels increase complexity of processing. Hyperlog programs are intractable in the general case.

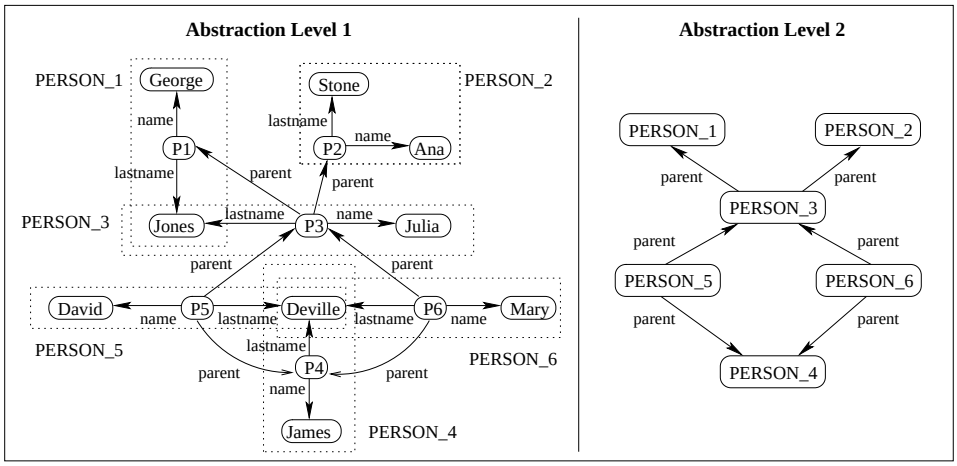

Figure 6: Simatic-XT. Here schema and instance are mixed. The relations Name-Lastname and Person-Parent are represented in two abstraction levels. In the first level (the most general), the graph contains the relations *name* and *lastname* to identify people (*P1, ..., P6*). In the second level we use the abstraction of *Person*, to compress the attributes *name* and *lastname* and represent only the relation *parent* between people.

3.3 Simatic-XT: A Data Model to Deal with Multi-scaled Networks

Motivated by modeling of transport networks (train, plane, water, telecommunications), Mainguenaud [86] proposed a graph (object-oriented) db-models that merges the concepts of graph and object oriented paradigm, focusing in the (graph) structure of the data but not on the behavior of entities to be modeled. An example is presented in Figure 6.

The model can be represented as a labeled directed multi-graph, defining three basic types: *Node type*, *Edge type*, and *Network type* (representing a graph). Additionally, the model introduces the notion of *Master Nodes* and *Master Edges*, to support levels of abstraction of sub-networks and paths respectively. Each object in the model has assigned an object identifier (OID), that permits identification and referencing. The level of abstraction is given by the nested level of Master Nodes and Edges Nodes. The model defines the attribute *in_edges* to represent the set of edges arriving in the subgraph (resp. path) that the Master Node (resp. Master Edge) represents. In the same form *out_edges* represent the set of edges leaving the Master node or Master Edge.

A sequel paper [81] presents a set of graph operators divided into three classes: *Basic operators*, managing the notion of abstraction (the Develop and Undevelop operators); *Elementary operators*, managing the notion of graph and sub-graph (Union, Concatenation, Selection and, Difference) and; *high level operators* (Paths, Inclusions and Intersections).

This proposal allows simple modeling and abstraction of complex objects and paths, and encapsulation at node and edge levels. It improves the representation and querying of paths between nodes, and the visualization of complex nodes and paths. At its current state, it lacks definition of integrity constraints.

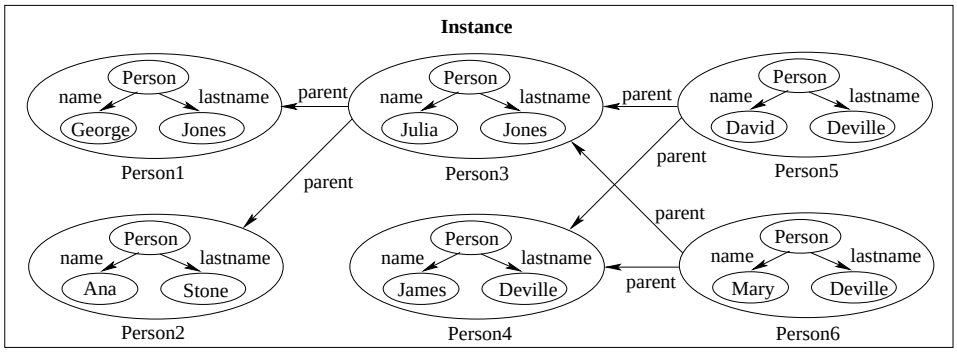

Figure 7: GGL. Schema and instances are mixed. *Packaged graph vertices* (*Person1, Person2, ...*) are used to encapsulate information about the graph defining a Person. Relations between these packages are established using edges labeled with *parent*.

3.4 Graph Database System for Genomics (GGL)

This db-model comes from the biology community and highlights the advantage of storing Genome maps as graphs. GGL includes a graph-theoretic db-model [54], a genome graph language [55], and query operators for the model [53, 52].

The model is based on binary relationships between objects. This model extends the basic notion of a graph by including vertices that represent edges types which allow specify relations between relations (higher-order relations), and encapsulated graphs as vertices. An example is presented in Figure 7.

A graph in GGL is basically a collection of: *Simple vertices* which model simple concepts and can be labeled or unlabeled; *Symbols* that define nodes without outgoing edges; *Edges* that connect two vertices and are labeled with a relation name; *Packaged graph vertices* that represent graphs which are packaged (encapsulated) into vertices; and *Relation type vertices* which are used to represent relations between relations (higher-order relations). According to this definition, the graph data structure consists of a directed labeled, possibly cyclic, graph which maintains hierarchically-ordered graphs.

For querying data in this model, two methods are proposed: The first [53] restrict the form of the query graph to be rooted directed acyclic graphs with equality constraints. The strategy is based in following the paths specified by the query-graph and returning the values that correspond to the end of the paths. The second, is a declarative programming language called

WEB [52], that defines queries as graphs with the same structures of the model and return the graphs in the database which *match* the query-graph.

Finally, the model defines two database-independent integrity constraints: Labels in a graph are uniquely named, and edges are composed of the labels and vertices of the graph in which the edge occurs.

The model was designed to support the requirements to model genome data, but also is generic enough to support complex interconnected structures. The distinction between schema and instance are blurred. Its nesting levels increase the complexity of modeling and processing.

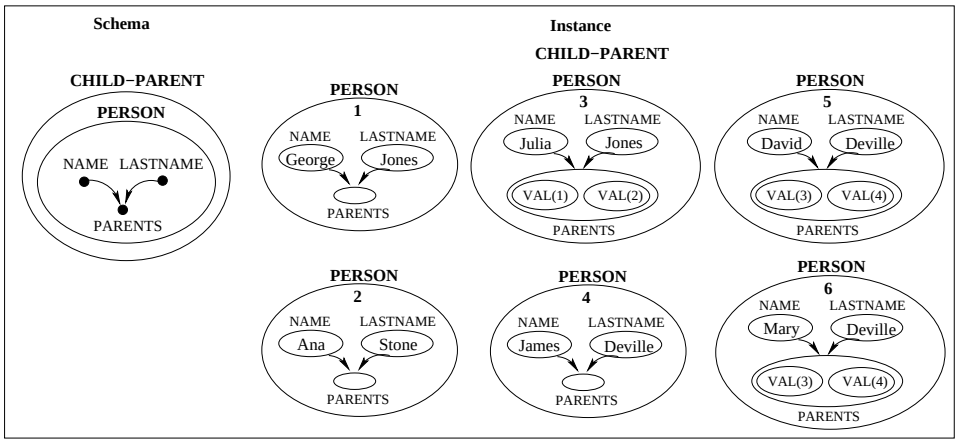

Figure 8: GROOVY. At the schema level (left), we model an object *PERSON* as an hypergraph that relates the attributes *NAME*, *LASTNAME* and *PARENTS*. Note the value functional dependency (VDF) *NAME,LASTNAME* $\rightarrow$ *PARENTS* logically represented by the directed hyperedge ({NAME,LASTNAME} {PARENTS}). This VFD asserts that NAME and LASTNAME uniquely determine the set of PARENTS.

3.5 Hypergraph-Based Data Model (GROOVY)

GROOVY (Graphically Represented Object-Oriented data model with Values [85]) is a proposal of object-oriented db-model which is formalized using *hypergraphs*, that is, a generalized notion of graph where the notion of edge is extended to *hyperedge*, which relates an arbitrary set of nodes [19]. The features of hypergraphs are used in several directions, *e.g.* to define functional dependencies. An example of hypergraph schema and instance is presented in Figure 8.

The model defines a set of structures for an object data model: (a) *Value Schema*, that define the attributes (or elements) that contain a class of objects. Attributes can be atomic or multi-valued (set or tuples). Attributes in value schemes can be themselves value schemas, allowing representation of complex objects and encapsulation of information. (b) *Instance*: Defines a set of valid *objects* over the value schema. An object is a pair $O=< i, v >$ where i is the object-ID (identity) and v is the object value (properties). (c) *Value Functional Dependencies*: Are used at the value schema level to assert that the value of a set of attributes uniquely determines the value of other attribute. The determined attribute can be single-valued or multi-valued. (d) *Object schema*: Is as triple $< N, F, S >$, where N is a value schema, F

is a set of value functional dependencies over N, and S is a set of subsets of N (sub-object schemas) including N itself.

The previous structure is defined in terms of hypergraphs, establishing a one-to-one correspondence between each object schema $< N, F, S >$ and a hypergraph interpreting N as nodes, F as directed hyperedges, and S as undirected hyperedges. Note that at the instance level, objects over object and class schemas can be represented as labeled hypergraphs.led hypergraphs.

In addition, *class schemas* are defined to introduce the notions of class and inheritance. A class schema corresponding to an object schema $< N, F, S >$ is an hypergraph $< N, F, H >$, where the H component indicates all the super-class schemas of the class-schema. A *class* over a class schema is just an instance of an object schema.

An hypergraph manipulation language (HML) for querying and updating hypergraphs is presented. It has two operators for querying hypergraphs by identifier or by value, and eight operators for manipulation (insertion and deletion) of hypergraphs and hyperedges.

The use of hypergraphs has several advantages. Introduces a single formalism for both sub-object sharing and structural inheritance, avoiding redundancy of data (values of common sub-objects are shared by their super-objects). Hypergraphs allow the definition of complex objects (using undirected hyperedges) and functional dependencies (using directed hyperedges). Allows supports for object-ID and (multiple) structural inheritance. Value functional dependences establish semantic integrity constraints for object schemas.

The notion of hypergraphs is also used in other proposals:

- Consens and Mendelzon [34] present a query and visualization system based on the concept of *hygraphs*, a version of hypergraphs. Their model defines an special type of edge called *Blob*, which relates a node with a set of nodes.

- Tompa [125] proposes a model for hypertext where nodes represent web pages and hyperedges represent user state and browsing.

- Watters and Shepherd [132] use hypergraphs to model data instances (in an existent database) and access to them. The model represents data instances as nodes in a hypergraph, and perform operations over both hyperedges and nodes representing data.

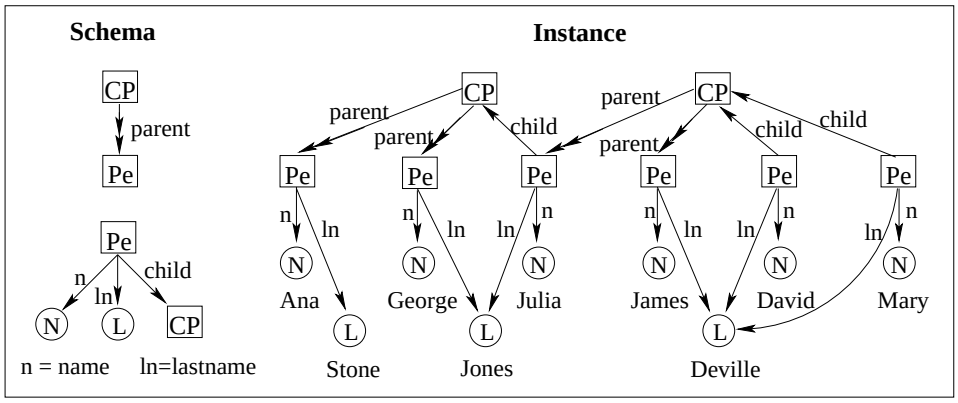

Figure 9: GOOD. In the schema, we use printable nodes N and L to represent names and lastnames respectively and non-printable nodes Pe(rson) and CP to represent relations Name-Lastname and Child-Parent respectively. The double arrow indicates relationship, and the simple arrow indicates functional relationship. The instance is got by assigning values to printable nodes and instantiating the CP and PE nodes.

3.6 Graph Object Oriented Data Model (GOOD)

The Graph Object Oriented Data Model [59] is a proposal oriented mainly to develop database end-user graphical interfaces [61]. In GOOD, schema and instances are represented by directed labeled graphs, and the data manipulation is expressed by graph transformations. An example of its application is the database management system presented in [49].

The model permits only two types of nodes, non-printable nodes (denoted by squares) and printable nodes (denoted by circles). There is not distinction between atomic, composed and set objects. There are two types of edges, functional (have a unique value, denoted by $\rightarrow$) and non-functional (multi-valued and denoted by $\twoheadrightarrow$). In a more detailed version [60] were added node and edges for representing set containment, object composition, generalization, and specialization. The GOOD schema and instance for the general example is presented in Figure 9.

GOOD includes a data transformation language with graphical syntax and semantics. It contains five basic graph transformation operations, four corresponding to elementary manipulation of graphs: addition of nodes and edges, deletion of nodes and edges. The fifth operation called abstraction, is used to group nodes on the basis of common functional or non-functional properties. The specification of all these operations relies on the notion of

pattern to describe subgraphs in a object base instance.

GOOD presents other features like macros (for more succinct expression of frequent operations), computational-completeness of the query language, and simulation of object-oriented characteristics like encapsulation and inheritance.

The model presented introduced several useful features. The notion of printable and non-printable nodes is relevant for design of graphical interfaces, although introduces additional information obscuring the semantics of relations. It has a simple definition of multivalued relations and allows recursive relations. Solves in a balanced way the redundancy of data problem. Nevertheless, the db-model is incomplete, currently lacking integrity constraints.

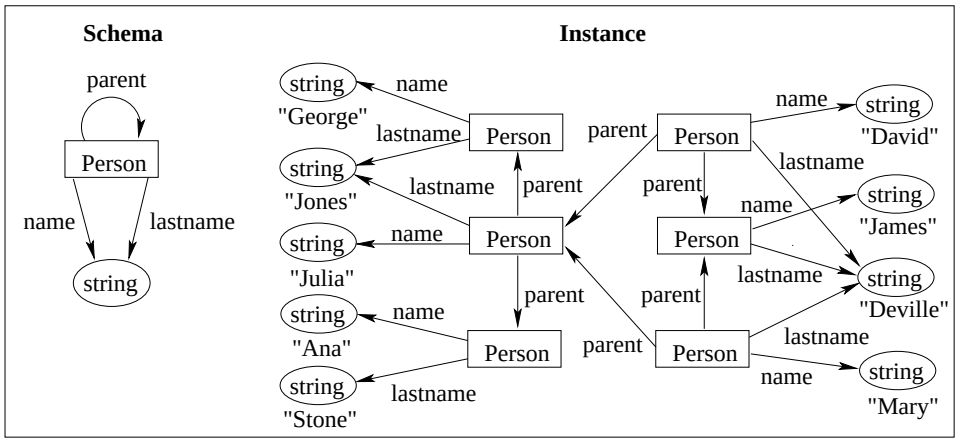

Figure 10: GMOD. In the schema, nodes represent abstract objects (*Person*) and labeled edges establish relations with primitive objects (properties *name* and *lastname*) and other abstract objects (*parent* relation). For building an instance, we instantiate the schema for each person by assigning values to oval nodes.

3.7 Graph-oriented Object Manipulation (GMOD)

GMOD [13] is a proposal of a general model for object database concepts oriented towards graph-oriented database user interfaces. Schema and instance are labeled digraphs. An example is presented in Figure 10.

The *schema* graph has two class of nodes, *abstract objects* (rectangular shape) representing class names and, *primitive objects* (oval shape) representing basic types. Edges represent properties of abstract objects. Distinction between single-value and multi-value properties is not considered.

The *instance* graph contains the data and includes instance nodes for abstract and primitive objects (represented like in the schema level). The latter have an additional label indicating their value (according to the primitive object domain). The same edges defined in the schema are used to represent the properties of instance objects, but their use is not necessarily required (incomplete information is allowed). Formally, there is a graph morphism from the graph instance (without the labels indicating value) to the schema.

The model uses graph pattern matching as a uniform object manipulation primitive for querying, specification, updating, manipulation, viewing and browsing.

The model allows a simple representation of objects and relations, hence

simple modeling. Also it allows incomplete information, and permits avoiding redundancy of data. The issue of property-dependent identity and a not completely transparent notion of object-ID incorporates some complexities.

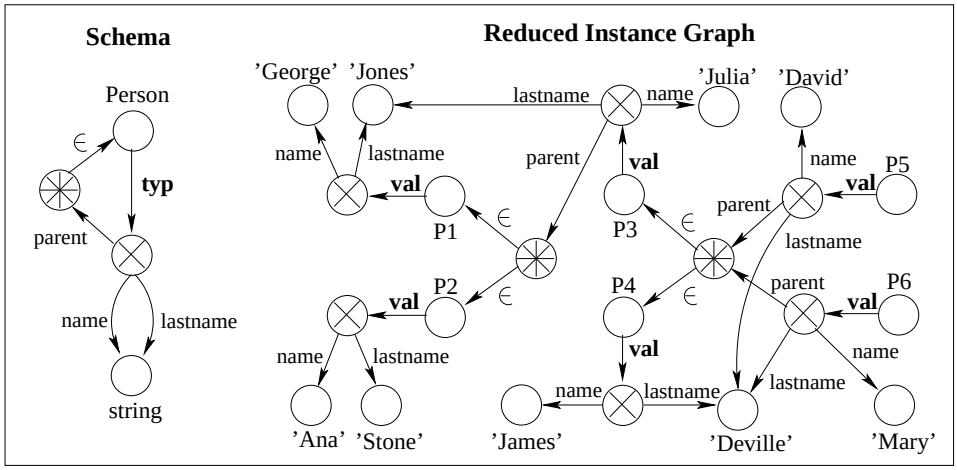

Figure 11: PaMaL. The example shows all the nodes defined in PaMaL: basic type (*string*), class (*Person*), tuple ($\otimes$), set ($\circledast$) nodes for the schema level, and atomic (*George, Ana*, etc.), instance (*P1, P2*, etc), tuple and set nodes for the instance level. Note the use of edges $\in$ to indicate elements in a set, and the edge **typ** to indicate the type of class *Person* (these edges are changed to **val** in the instance level).

3.8 Object Oriented Pattern Matching Language (PaMaL)

PaMaL is a graphical data manipulation language that uses patterns (represented as graphs) to specify the parts of the instance on which the operation has to be executed. Gemis and Paredaens [48] proposed this pattern-based query language based on a graphical object-oriented db-model as an extension of GOOD by an explicit representation of tuples and sets. An example is presented in Figure 11.

The *schema* defines four types of Nodes: $\bigcirc$ class nodes (upper-case labels), $\bigcirc$ basic-type nodes (lower-case labels), $\otimes$ tuple nodes, and $\circledast$ set nodes. There are four kinds of edges, indicating attribute of a tuple, type of the elements in a set (labeled with $\in$), type of the classes (labeled with **typ**), and hierarchical relationship (labeled with **isa**).

An *instance* graph may contain *atomic*, *instance*, *tuple* and *set* nodes (they are determined by the schema). *Atomic objects* are labeled with values from their domains and *instance objects* are labeled with object-ID's. *tuple* and *set* objects are identified by their outgoing edges, motivating the notion of *reduced instance graph* to merge nodes that represent the same set or tuple. To refer to the node that describes the properties (or content) of

an object, an edge labeled **val** is used and represents the edge **typ** in the schema.

PaMaL presents operators for addition, deletion (of nodes and edges) and an special operation that reduces instance graphs. It incorporates loop, procedure and program constructs that makes it a computationally complete language. Among the highlights of the model are the explicit definition of sets and tuples, the multiple inheritance, and the use of graphics to describe queries.

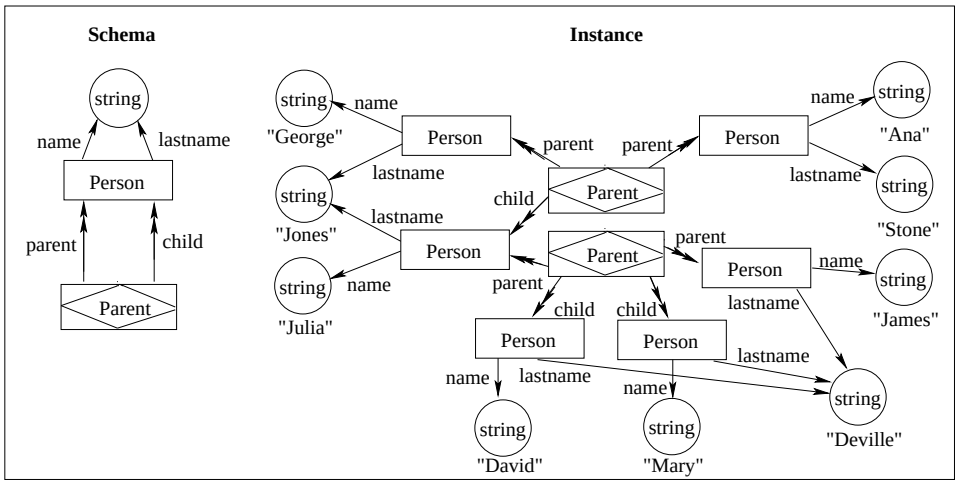

Figure 12: GOAL: The schema presented in the example shows the use of the object node *Person* with properties Name and Lastname. The association node *Parent* and the double headed edges *parent* and *child* allow to express the relation Person-Parent. At the instance level, we assign values to value nodes (*string*) and create instances for object and association nodes. Note that nodes with same value were merged (*e.g. Deville*).

3.9 Graph-based Object and Association Language (GOAL)

Motivated by the introduction of more complex db-models like object-oriented ones, and directed to offer the user a consistent graphical interface, Hidders and Paredaens [69] proposed a graph based db-model for describing schemes and instances of object databases. GOAL extends the model of GOOD by adding the concept of *association* nodes (similar to the entity relationship model). The main difference between associations and objects is that the identity of objects is independent of their properties, whereas associations are considered identical if they have the same properties. An example is presented in Figure 12.

Schema and instances in GOAL are represented as finite directed labeled graphs. A schema allows to define three types of nodes: *object nodes* that represent objects (rectangular nodes); *value nodes* that represent printable values such as string, integers or booleans (round nodes) and; *association nodes* that represent associations or relations among more than two nodes (diamond shape nodes). Objects and associations may have properties that are represented by edges. The model allows representation of functional

properties (single headed edges) and multi-valued properties (double headed edges), as well as ISA relations (double unlabeled arrows). An instance in GOAL assigns values to value nodes and creates instances for object and association nodes.

GOAL introduces the notion of consistent schema to enforce that objects only belong to the class they are labeled with and its super-classes. In addition GOAL presents a graph data manipulation language with operations for addition and deletion based on pattern matching. The addition (deletion) operation adds (deletes) nodes and/or edges at the instance level. A finite sequence of additions and deletions is called a *transformation*.

There are several novelties introduced by this model. Association nodes allow simple definition of multi-attribute and multi-valued relations. In contrast to the Entity Relationship model, GOAL supports relations between associations. Properties are optional, therefore it is possible to model incomplete information. Additionally, GOAL defines restrictions that introduce notions of consistent scheme and weak instance.

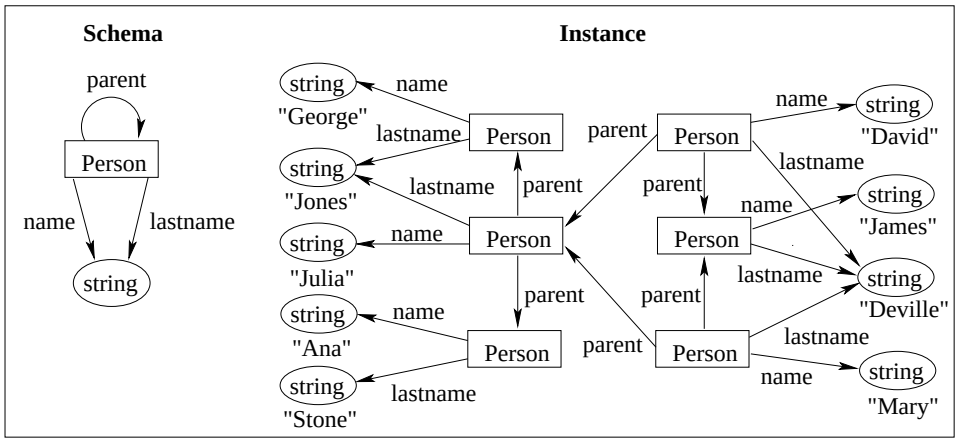

Figure 13: G-Log. The schema defines people as objects *Person*, each one identified by their properties *name*, *lastname* and *parent*. The latter establishes the relation child-parent. The instance is got in a similar way as in GMOD.

3.10 G-Log: A Graph-Based Query Language

G-Log [99] is a proposal of a declarative query language for graphs, which defines a graph-based db-model oriented to end-user interfaces dealing with complex objects. Schemes, instances and rules can be viewed as directed labeled graphs. An example is presented in Figure 13.

A *schema* in G-Log is a directed graph that contains nodes representing classes of objects and edges representing classes of relationships between objects. As in GOOD, we can distinguish between *Printable objects* representing objects with an atomic value (string, integer, reals, dates, text, images, sound) and *Non-printable* objects representing composite objects. Relationships can be mono-valued or multi-valued.

A G-Log *instance* contains instances of nodes and edges defined in the schema. The model requires that all node and edge labels occurring in the instance must occur in the schema. Each non-printable object represents a distinct complex object that is identified by their relations (property-dependent identity). Atomic values can occur only once in an instance, thus avoiding data redundancy.

Queries in G-log are expressed by programs which consist of a number of rules and use patterns (denoted as graphs with variables in the nodes and predicates in the edges) to match subgraphs in the instance.

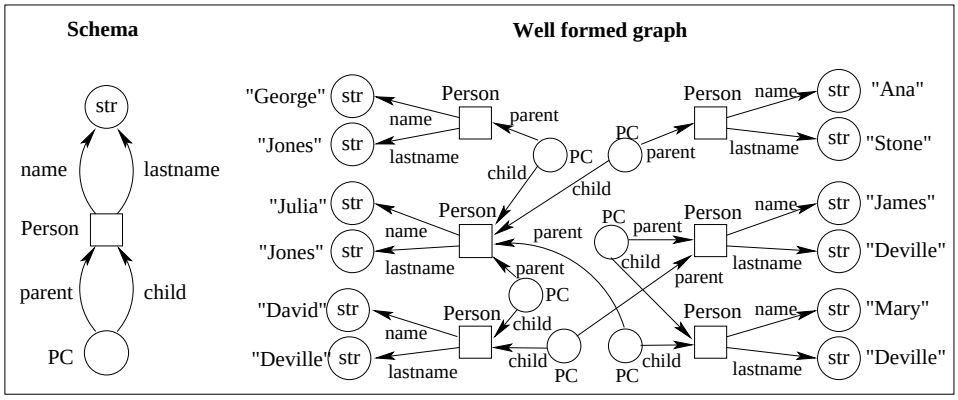

Figure 14: GDM. In the schema each entity Person (object node represented as a square) has assigned the attributes *name* and *lastname* (basic value nodes represented round and labeled *str*). We use the composite-value node *PC* to establish the relationship *child-Parent*. Note the redundancy introduced by the node *PC*. The instance is built by instantiating the schema for each person.

3.11 Graph Data Model (GDM)

GDM [68, 67] is a graph-based db-model based on GOOD, that adds explicit complex values, inheritance and n-ary symmetric relationships. Schema and instances in GDM are described by labeled graphs called instance graph and schema graph. An example is presented in Figure 14.

A *schema* graph contains nodes that represent classes and edges labeled with attribute names indicating that entities in that class may have that attribute. Three types of class nodes are allowed: *object*, *composite-value* and, *basic value*. An edge denoted by a double-line arrow defines an *ISA* relation between class nodes.

In an *instance* graph, nodes represent entities and edges represent attributes of these entities. We can have *object nodes* (depicted squared), *composite-value nodes* (round empty) and *basic value nodes* (round labeled with a basic-type name). An object node is labeled with zero o more class names indicating their membership to certain classes. If several edges with the same label leave a node, then it is a single set-valued attribute.

GDM introduces the concept of well-formed graph defining four constraints: (I-BVA) no edge leaves from a basic-value node; (I-BVT) each basic value node has assigned a real value that is in the domain of the basic-type of the node; (I-NS) for each class-free node n there is a path that

ends in n and starts in a class-labeled node; and (I-REA) composite-value nodes have either exactly one incoming edge or are labeled with exactly one class name, but not both. In addition the model considers the notion of consistency defining *extension relations* which are many-to-many relations between the nodes in the data graph and nodes in the schema graph, indicating correspondence between entities and classes.

The proposal includes a graph-based update language called GUL, that is based on pattern matching. GUL permits addition and deletion operations, plus a reduction operation that reduces well-formed data graphs to instance graphs by merging similar basic-value nodes and similar composite-value nodes.

The GDM model presents the following benefits. The independence of the definition of the notions of schema and instance permits that instances can exist without a schema, allowing representation of semi-structured data. Permits the explicit representation of complex values, inheritance (using ISA edges) and definition of n-ary symmetric relationships. The composite-value nodes allow simple definition of multi-attribute and multi-valued relations. Finally, let us remark that this model introduces notions of consistency and well-formed graphs.

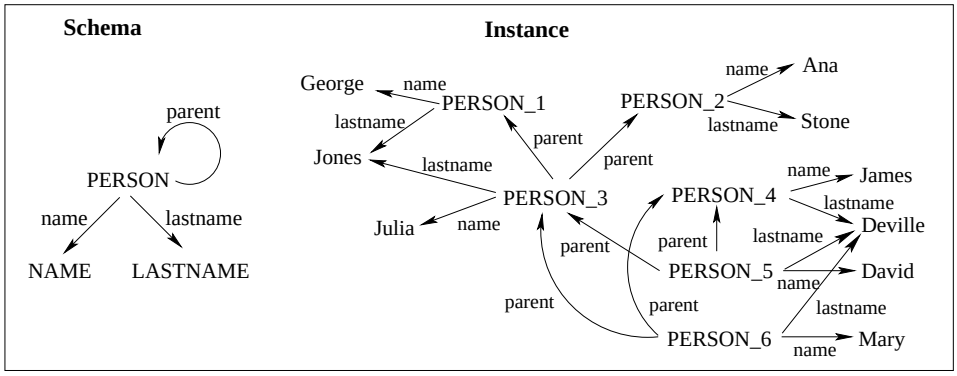

Figure 15: Gram. At the scheme level we use generalized names for definition of entities and relations. At the instance level, we create instance labels (*e.g.* PERSON_1) to represent entities, and use the edges (defined in the schema) to express relations between data and entities.

3.12 Gram: A Graph Data Model and Query Language

Motivated by hypertext querying, Amann and Scholl [11] introduce Gram, a graph db-model where data is organized as a graph. A schema in Gram is a directed labeled multigraph, where each node is labeled with a symbol called a *type*, which has associated a domain of values. In the same way, each edge has assigned a label representing a relation between types (see example in Figure 15). A feature of Gram is the use of special objects for explicit definition of paths called *walks*. An alternating sequence of nodes and edges represent and walk, which combined with other walks conforms other special objects called *hyperwalks*.

For querying the model (particularly path-like queries), an algebraic language based on regular expressions is proposed. For this purpose a hyperwalk algebra is defined, which presents unary operations (projection, selection, renaming) and binary operations (join, concatenation, set operations), all closed under the set of hyperwalks.

The proposal Gram does not define integrity constraints.

3.13 Related Data Models

Besides the models reviewed, there are other proposals that present graph-like features, although not explicitly designed to model the structure and connectivity of the information. In this section we will describe the most relevant of these.

3.13.1 GraphDB

Güting [58] proposes an explicit model named GraphDB, which allows simple modeling of graphs in an object oriented environment. The model permits an *explicit representation* of graphs by defining object classes whose objects can be viewed as nodes, edges and explicitly stored paths of a graph (which is the whole database instance).

A database in GraphDB is a collection of object classes partitioned into three kinds of classes: simple, link and path classes. Also there are data types, object types and tuple types. A *simple class* object has an object type, object identity and attributes whose values are either of a datatype (*e.g.* integer, string) or of an object type. An attribute may contain a reference to another object. Object classes are organized in a hierarchy of classes and there are related notions of subtyping among tuple, object and data types.

There are four types of operators to query GraphDB data: *Derive statements*: selection, join, projection and function operators; *Rewrite operations*: allow to replace objects or subsequences by other (new) objects; *Union operator*: designed for transforming heterogeneous sets of objects into a homogeneous one; *Graph operations*: Shortest path search.

The idea of modeling graphs using object oriented concepts is presented in other proposals, generically called *object-oriented graph models*. A typical example is GOQL [113], a proposal of graph query language for modeling and querying of multimedia application graphs (represented as DAGs). This proposal defines a object oriented db-model (similar to GraphDB) that defines four types of objects: node, edge, path and graph. GOQL uses an SQL-like syntax for construction, querying and manipulation of such objects.

3.13.2 Database Graph Views

A database graph view [57] provides a functional definition of graphs over data that can be stored in either relational, object oriented or file systems. In other words, the model proposes the definition of personalized graph views

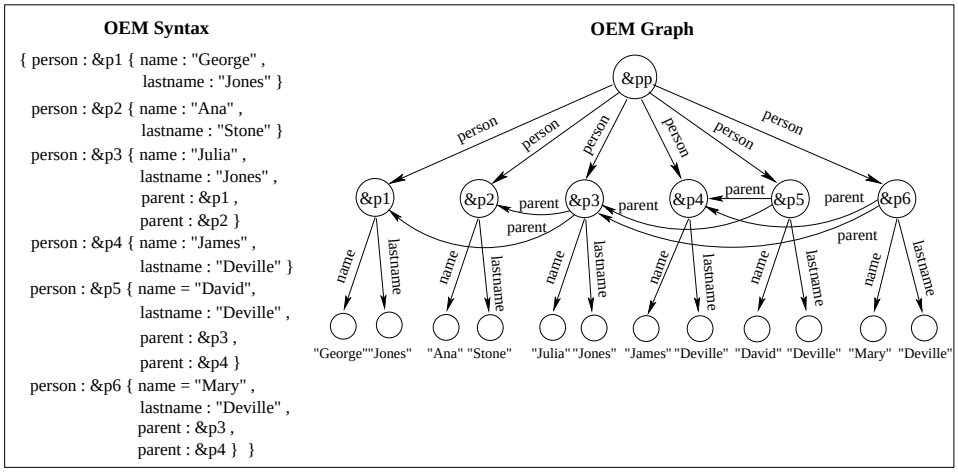

Figure 16: Object Exchange Model (OEM). Schema and instance are mixed. The data is modeled beginning in a root node &pp, with children *person nodes*, each of them identified by an Object-ID (*e.g.* &p2). These nodes have children that contain data (*name* and *lastname*) or references to other nodes (*parent*). Referencing permits to establish relations between distinct hierarchical levels. Note the tree structure obtained if one forgets the pointers to OIDs, a characteristic of semistructured data.

of the data with management and querying purposes, and independent of its implementation.

In brief the model define underlying graphs over the database and proposes a set of primitives called *derivation operators* for definition and querying of graph views. Unary derivation operators allow selection of nodes and edges. Binary derivation operators are used to build new graph views resulting from the union, intersection or difference of two graph views.

3.13.3 Object Exchange Model (OEM)

OEM [97] is a semistructured db-model that allows simple and flexible modeling of complex features of a source using the ideas of nesting and object identity from object oriented db-models (features such as classes, methods and inheritance are omitted). The main motivation of OEM was the information integration problem. Therefore it defines a syntax that is well suited for information exchange in heterogeneous dynamic environments. The data in OEM can be represented as a rooted directed connected graph. An example of OEM graph and syntax is presented in Figure 16.

OEM define objects with the structure $< OID, Label, Type, Value >$, where: OID is an unique identifier for the object (or null), *Label* is a character string that describes the object (expected to be human understandable), *Type* is the datatype of the object's value (atomic or set type) and, *Value* is a variable-length value for the object (either an atomic value or a set of objects). Data represented in OEM can be thought of as a graph with Object-IDs representing node-labels and OEM-labels representing edge-labels. Atomic objects are leaf nodes where the OEM-value is the node value.

The main feature of OEM data is that is self-describing, in the sense that it can be parsed without recurring to an external schema and uses human understandable labels that add semantic information about objects. Due to the fact that there is no notion of schema or object class (although each object defines its own schema), OEM offers the flexibility needed in heterogeneous dynamic environments.

OEM-QL is a declarative query language design to request OEM objects. The basic construct in OEM-QL is an SQL-like SELECT-FROM-WHERE expression.

3.13.4 eXtended Markup Language (XML)

The XML [21] model did not originate in the database community. It was introduced as an standard for exchanging information between Web applications. XML is an extensible version of HTML allowing annotating data with information about its meaning rather than just its presentation [131]. From an abstract point of view, XML data are labeled ordered trees (with labels on nodes), where internal nodes define the structure and leaves the data (scheme and data are mixed.).

Compared to graph data db-models, XML has a ordered-tree-like structure, which is a restricted type of graph. Nevertheless, XML additionally provides a referencing mechanism among elements that allows simulating arbitrary graphs. In this sense XML can simulate semistructured data.

In XML, the information about the hierarchical structure of the data is part of the data (in other words XML is self-describing); in contrast, in graph db-models this information is described by the scheme graph in a more flexible fashion using relations between entities. From this point of view, graph db-models use connections to explicitly represent generalization, compositions, hierarchy, classification, and any/other type of relations.

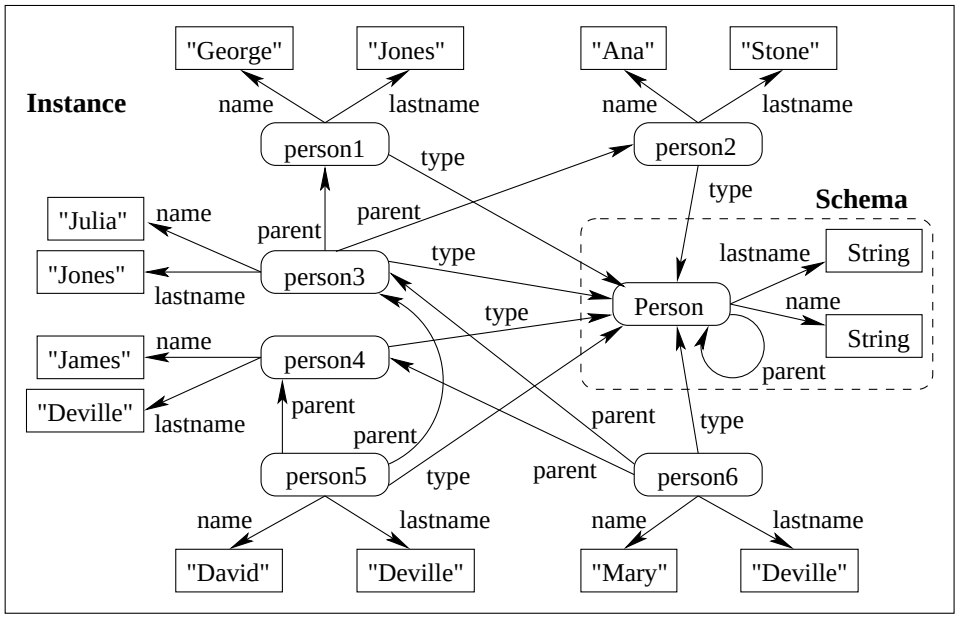

Figure 17: RDF. Schema and instance are mixed together. In the example, the edges labeled *type* disconnect the instance from the schema. The instance is built by the subgraphs obtained by instantiating the nodes of the schema, and establishing the corresponding parent edges between these subgraphs.

3.13.5 Resource Description Framework (RDF)

The Resource Description Framework (RDF) [76] is a recommendation of the W3C designed originally to represent metadata. The broad goal of RDF is to define a mechanism for describing resources that makes no assumptions about a particular application domain, nor defines (a priori) the semantics of any application domain. RDF is domain neutral and models information with graph-like structure.

An atomic RDF expression is a triple consisting of a subject (the resource being described), a predicate (the property) and an object (the property value). Each triple represents a statement of a relationship between the things that it links. A general RDF expression is a set of such triples, which can be intuitively considered as a labeled graph, called an RDF Graph [76], which formally is not a graph [65] (see example in Figure 17).

One of the main advantages (features) of the RDF model is its ability to interconnect resources in an extensible way. Thus, basic notions of graph theory like node, edge, path, neighborhood, connectivity, distance, degree,

etc., play a central role in this model.

Currently there is research work on storing information expressed in RDF, but none of these works define a graph db-model or even a db-model. In addition several languages for querying RDF data has been proposed and implemented, which follow the lines of database query languages like SQL, OQL, and XPath. A discussion of aspects related to querying RDF from a graph database perspective is presented in [14].

SPARQL [105] is a proposal of Protocol and Query Language designed for easy access to RDF stores. It defines a query language with a SQL-like style, where a simple query is based on query patterns, and query processing consists of binding of variables to generate pattern solutions (graph pattern matching).

4 Query Languages and Integrity Constraints

4.1 Graph Query Languages

A query language is a collection of operators or inferencing rules which can be applied to any valid instances of the data structure types of the model, with the objective of manipulating and querying data in those structures in any combinations desired [31].

A great deal of papers discuss the problems concerning the definition of a query language for a db-model [129, 70, 106, 66, 2, 5]. Also a variety of query languages and formal frameworks for studying them have been proposed and developed, including the relational db-model [27], semantic databases [16, 12], object-oriented databases [74], semistructured data [24, 2, 4] and the Web [5, 47].

Among graph db-models, there is substantial work focused in query languages, the problem of querying graphs and the visual presentation of results. Particular emphasis has been given to graphical query languages (See Figure 18 for an example). Following, we describe the most representative graph query languages.

- The Logical Database Model [79, 80] presents a logic very much in the spirit of relational tuple calculus, which uses fixed sort variables and atomic formulas to represent queries over a schema using the power of full first order languages. The result of a query consists of those objects over a valid instance that satisfy the query formula. In addition the model presents an alternative algebraic query language proven to be equivalent to the logical one.

- Cardelli et al. [26] introduced a spatial logic for reasoning about graphs and define a query language based in pattern matching and recursion. This Graph Logic combines first-order logic with additional structural connectives. A query ask for a substitution of variables such that a *satisfaction relation* determines which graph satisfy which formulae. The query language is based on *queries* that build new graphs from old and *transducers* that relates input graphs with output graphs.

- The proposal G-Log [99] includes a declarative language for complex objects with identity. It uses the expressive power of logic through the notion or *rule satisfaction*, to evaluate queries which are expressed as G-Log programs. These *G-Log programs* are sets of graph-based rules, which specify how the schema an instance of the database will

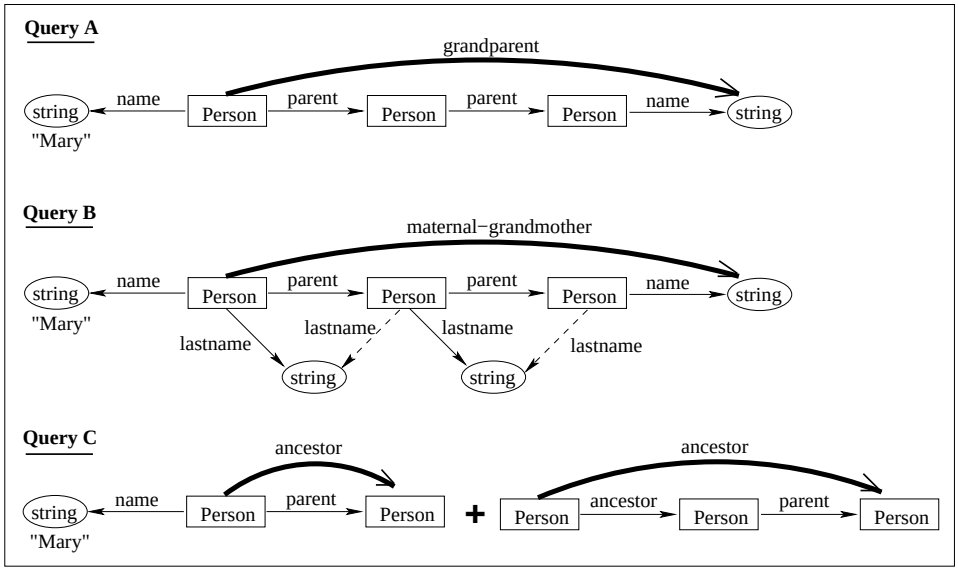

Figure 18: Example of a graphical query language. The figure shows a G-Log query for the instance in Figure 13. *Query A* asks for the names of Mary's grandparents (fixed path query). *Query B* asks for the name of the maternal grandmother of Mary (tree-like query). *Query C* calculates Mary's Ancestors (transitive closure).

change. G-Log is a graph-based, declarative, nondeterministic, and computationally complete query language that does not suffer from the copy-elimination problem.

- Oriented to search the Web, Flesca and Greco [45] show how to use partially ordered languages to define path queries to search databases and present results on their computational complexity. In addition, a query language based on the previous ideas is proposed in [46].

- In the context of graph-oriented object models, there are query languages that regard database transformations as *graph transformations* (which can be interpreted as database queries and updates). They are based on *graph-pattern matching* and allow the user to specify node insertions and deletions in a graphical way. GOOD [59] presented a graph-based language that is shown to be able to express all constructive database transformations. This language was followed by the proposals GMOD [13], PaMaL [48], GOAL [69], and GUL [68].

Additionally, GOAL includes the notion of *fixpoints* in order to handle the recursion derived from a finite list of additions and deletions. PaMaL proposed the inclusion of *Loop, Procedure and Programs constructs*, and PaMaL and GUL presented an operator that reduces instance graphs by deleting repeated data. Note that graph-oriented manipulation formalisms based on patterns allow a syntax-directed way of working much more natural than text-based interfaces.

- The query languages G, G+ y GraphLog integrate a family of related graphical languages defined over a general simple graph model.

  The graphical query language G [36] is based on regular expressions that allow simple formulation of recursive queries. A *graphical query* in G is a set of labeled directed multigraphs where nodes may be either variables or constants, and edges can be labeled with regular expressions. The result of a query is the union of all query graphs which match subgraphs from the instance.

  G evolved into a more powerful language called G+ [37] where a query graph remains as the basic building block. A simple query in G+ has two elements, a query graph that specifies the class of patterns to search and a summary graph that represent how to restructure the answer obtained by the query graph.

  GraphLog [35] is a query language for hypertext that extends G+ by adding negation and unifying the concept of a query graph. A query is now only one graph pattern containing one distinguished edge, which corresponds to the restructured edge of the summary graph in G+. The effect of the query is to find all instances of the pattern that occur in the database graph and for each one of them define a virtual link represented by the distinguished edge. GraphLog includes an implicit transitive closure operator, which replaces the usual recursion mechanism. The algorithms used in the GraphLog implementation are discussed in [92].

- Glide [50] is a graph query language where queries are expressed using a linear notation formed by labels and wild-cards (regular expressions). Glide use a method called GraphGrep based on subgraph matching to solve the queries.

- GROOVY [85] introduces a Hypergraph Manipulation Language (HML) for querying and updating labeled hypergraphs. It defines two basic operators for querying hypergraphs by identifier or by value, and eight

operators for manipulation (addition an deletion) of hypergraphs and hyperedges. Watters and Shepherd [132] presents a framework for general data access based in hypergraphs that include operators for creation of edges and set operators like intersection, union and difference. In a different context, Tomca [125] introduces basic operations over hypergraph structures representing user state and views in page-oriented hypertext data.

- The literature also include proposals of query languages that deal with *hypernode* structures. The Hypernode model [84] defines a logic-based query and update language, which is based in the expression of queries as sets of hypernode rules (h-rules) that are called an *hypernode program*. The query language defines an operator which infers new hypernodes from the instance using the set of rules in a hypernode program.

  This query language was extended by Hyperlog [104, 103] including deletions as well as insertions, and discussing in more detail the implementation issues. A full Turing-machine capability is obtained by adding composition, conditional constructs and iteration.

  In a procedural style, HNQL [83] defines a set of operators for declarative querying and updating of hypernodes. It also includes assignment, sequential composition, conditional (for making inferences), for loop, and while loop constructs.

- In the area of Geographic information Systems, the Simatic-XT model [81] defines a query language. It includes basic operators that deal with encapsulate data (nesting of hypernodes), set operators (union, concatenation, selection and difference) and high level operators (paths, inclusion and intersections).

- WEB [52, 53] is a declarative programming language based on a graph logic and oriented to querying genome data. WEB programs define graph templates for creating, manipulating and querying objects and relationships in the database. These operations are answered by matching graphs in valid instances.

- Models like Gram [11] and GOQL [113] propose SQL-Style query languages with explicit path expressions. *Gram* presents a query algebra where regular expressions over data types are used to select walks (paths) in a graph. It uses a data model where walks are the basic objects. A walk expression is a regular expression without union, whose language contains only alternating sequences of node and edge

types, starting and ending with a node type. The query language is based on a hyperwalk algebra with operations closed under the set of hyperwalks.

- Models like DGV [57] and GraphDB [58] define special operators for functional definition and querying of graphs. For example, a query in GraphDB consists of several steps, each of one computes operations that specify argument subgraphs in the form of regular expressions over edges that extend or restrict dynamically the database graph. GraphDB includes a class of objects called path class, which are used to represent several paths in the database.

One of the most fundamental graph problems in graph query languages is to compute reachability of information, which is traduced in path problems characterized and expressed by recursive queries. For example, path queries are relevant in GraphLog, Gram, Simatic-XT, DGV, GOQL, Flesca and Grego, Cardelli et al., and in less degree treated in Hypernode, GOAL, Hyperlog, GraphDB, G-Log, and HNQL. The notion of shortest path is considered in Flesca and Greco, G+, GraphLog, and DGV. Path and other relevant graph queries for RDF are discussed in [14].

The importance and computational complexity of path-based queries is studied in several works [8, 7, 6, 108]. Finding simple paths with desired properties in direct graphs is difficult, and essentially every nontrivial property gives rise to an NP-complete problem [111]. Yannakakis [135] surveyed a set of paths problems relevant to the database area including computing transitive closures, recursive queries and the complexity of path searching. Mannino and Shapiro [88] present a survey of extensions to database query languages for solve graph traversal problems.

4.2 Integrity Constraints

Integrity constraints are general statements and rules, which implicitly or explicitly define the set of consistent database states or changes of state or both [31]. Integrity Constraints are a relevant component in the design of db-models. While their primary role is to restrict the allowed instances of the schema (acting as a filter of invalid data), they are also useful in query optimization, schema design, and choice of efficient storage and access methods. Constraints are used to express database semantics like domain restrictions, specify relationships between components and state database behavior [124].

The utilization, specification, and complexity of constraints is determined by the richness of the db-model. Integrity constraints have been studied for the relational [38, 123], semantic [133, 124], object oriented [110], and semistructured [25, 10] db-models. Thalheim [124] presents a unifying framework for integrity constraints.

In the case of graph db-models, examples of integrity constraints include identity and referential integrity constraints, functional and inclusion dependencies, and schema-instance consistency. Next we describe some notions of integrity constraints defined in graph database models.

LDM [80] defines a logic that use variables to express well-formed formulas over a schema. Integrity constraints are expressed in terms of satisfaction of these LDM formulas.

The Hypernode Model [104] defines two integrity constraints: *Entity Integrity* enforces that each hypernode is a unique real world entity identified by their content; *Referential Integrity* requires that only existing entities be referenced. In [83] the notion of semantic constraints were considered. The concept of *Hypernode functional dependency*, denoted by $A \rightarrow B$, where A and B are sets of attributes, leave us express that the set of attributes A determines the value of the set of attributes B in all hypernodes of the database.

GGL [54] defines two integrity constraints: (1) labels in a graph are uniquely named; (2) edges are composed of the labels and vertices of the graph in which the edge occurs. These constraints are similar to primary key and foreign key (referential) integrity constraints in the relational db-model.

GROOVY [85] uses *directed hyperedges* to represent *Value Functional Dependencies (VFDs)*, which are used in the value schema level to establish semantic integrity constraints. A VFD asserts that the value of a set of attributes uniquely determines the value of other attribute.

DGM [68] defines conditions enforcing that primitive nodes are only leaves, the real value of a node depends of its domain, and two constraints regarding edges with class nodes (these were described in Section 3).

The notion of *Schema-Instance consistency* is explicitly considered in GOAL [69], G-Log [99], and GDM [68]. In the case of graph-based object oriented db-models this notion is translated into the creation of *Valid Instances*, for example GMOD [13] and PaMaL [48].

Characteristics | Database Model		LDM 1984	Hypernode 1990	GOOD 1990	GROOVY 1991	GMOD 1992	Simatic-XT 1992	Gram 1992	Pamal 1993	GOAL 1993	Hypernode2 1994	Hypernode3 1995	GGL 1995	G-Log 1995	GDM 2002
Based on:	Graph model	√		√		√		√	√	√			√	√	√
	Hypergraphs				√										
	Hypernodes		√				√				√	√			
	Object Oriented model	√		√	√	√			√	√					√
Directed Graph	Node labeled	√	√	√	√	√	√	√	√	√	√	√	√	√	√
	Edge labeled			√	√	√	√	√	√	√			√	√	√
Support:	Schema	√		√	√	√		√	√	√	√	√		√	√
	Classes and Types			√	√	√	√	√	√	√	√	√	√	√	√
	Inheritance				√				√	√					√
	Specia/Generalization			√					√	√					√
	Collections	√	√		√		√		√		√	√			±
	Complex objects		√		√		√		√		√	√	√		√
	Node Ids		√		√	±	√	√	√	±	√	√	√	√	
	Multiple inheritance				√				√	√					
	Recursive relations	√	√	√		√		√	√	√	√	√		√	
	Explicit path types						√	√							
Query Language	Algebraic - Procedural	√					√	√				√			
	Logic - Declarative	√	√		√						√		√	√	
	Graphical			√		√			√	√					√
	Query	√	√		√		√	√	√		√	√	√	√	
	Transformation	√		√		√				√					√
	Path queries		±	±			√	√	±	±	±	±	±	±	
Integrity Constraints		√			√		√		√	±	√	√	√	±	√
Implementation			√	√			√	√			√	√	√		√
Motivation		Complex objects	Complex objects	Graphical Interfaces	General	Hypermedia	Transport networks	Hypertext	Graphical Interfaces	Graphical Interfaces	Complex objects	Hypertext	Genomics	Complex objects	Complex objects

Figure 19: Main proposals on Graph Database Models and their characteristics ("√" indicates support and "±" partial support). LDM [79, 80], Hypernode [84], GOOD [59, 60], GROOVI [85], GMOD [13], Simatic-XT [86], Gram [11], PaMaL [48], GOAL [69], Hypernode2 [104], Hypernode3 [83], GGL [54], G-Log [99], GDM [68].

References

[1] International Standard ISO/IEC 13250 Topic Maps, December 1999.

[2] S. Abiteboul. Querying Semi-Structured Data. In *Proc. of the 6th Int. Conf. on Database Theory (ICDT)*, volume 1186 of *LNCS*, pages 1–18. Springer, Jan 1997.

[3] S. Abiteboul and R. Hull. IFO: A Formal Semantic Database Model. In *Proc. of the 3th Symposium on Principles of Database Systems (PODS)*, pages 119–132. ACM Press, 1984.

[4] S. Abiteboul, D. Quass, J. McHugh, J. Widom, and J. L. Wiener. The Lorel query language for semistructured data. *International Journal on Digital Libraries (JODL)*, 1(1):68–88, 1997.

[5] S. Abiteboul and V. Vianu. Queries and Computation on the Web. In *Proc. of the 6th Int. Conf. on Database Theory (ICDT)*, volume 1186 of *LNCS*, pages 262–275. Springer, Jan 1997.

[6] R. Agrawal and H. V. Jagadish. Efficient Search in Very Large Databases. In *Proc. of the 14th Int. Conf. on Very Large Data Bases (VLDB)*, pages 407–418. Morgan Kaufmann, Aug - Sept 1988.

[7] R. Agrawal and H. V. Jagadish. Materialization and Incremental Update of Path Information. In *Proc. of the 5th Int. Conf. on Data Engineering (ICDE)*, pages 374–383. IEEE Computer Society, Feb 1989.

[8] R. Agrawal and H. V. Jagadish. Algorithms for Searching Massive Graphs. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 6(2):225–238, 1994.

[9] R. Albert and A.-L. Barabasi. Statistical mechanics of complex networks. *Reviews of Modern Physics*, 74:47, Jan 2002.

[10] N. Alechina, S. Demri, and M. de Rijke. A Modal Perspective on Path Constraints. *Journal of Logic and Computation*, 13(6):939–956, 2003.

[11] B. Amann and M. Scholl. Gram: A Graph Data Model and Query Language. In *European Conference on Hypertext Technology (ECHT)*, pages 201–211. ACM, Nov - Dec 1992.

[12] M. Andries and G. Engels. A Hybrid Query Language for an Extended Entity-Relationship Model. Technical Report TR 93-15, Institute of Advanced Computer Science, Universiteit Leiden, May 1993.

[13] M. Andries, M. Gemis, J. Paredaens, I. Thyssens, and J. V. den Bussche. Concepts for Graph-Oriented Object Manipulation. In *Proc. of the 3rd Int. Conf. on Extending Database Technology (EDBT)*, volume 580 of *LNCS*, pages 21–38. Springer, March 1992.

[14] R. Angles and C. Gutierrez. Querying RDF Data from a Graph Database Perspective. In *Proc. 2nd European Semantic Web Conference (ESWC)*, number 3532 in LNCS, pages 346–360, 2005.

[15] M.-A. Aufaure-Portier and C. Trépied. A Survey of Query Languages for Geographic Information Systems. In *Proc. of the 3rd Int. Workshop on Interfaces to Databases*, pages 431–438, July 1976.

[16] M. Azmoodeh and H. Du. GQL, A Graphical Query Language for Semantic Databases. In *Proc. of the 4th Int. Conf. on Scientific and Statistical Database Management (SSDBM)*, volume 339 of *LNCS*, pages 259–277. Springer, June 1988.

[17] C. Beeri. Data Models and Languages for Databases. In *Proc. of the 2nd Int. Conf. on Database Theory (ICDT)*, volume 326 of *LNCS*, pages 19–40. Springer, Aug - Sept 1988.

[18] G. Benkö, C. Flamm, and P. F. Stadler. A Graph-Based Toy Model of Chemistry. *Journal of Chemical Information and Computer Sciences (JCISD)*, 43(1):1085–1093, Jan 2003.

[19] C. Berge. *Graphs and Hypergraphs*. North-Holland, Amsterdam, 1973.

[20] U. Brandes. *Network Analysis*. Number 3418 in LNCS. Springer-Verlag, 2005.

[21] T. Bray, J. Paoli, and C. M. Sperberg-McQueen. Extensible Markup Language (XML) 1.0, W3C Recommendation 10 February 1998. http://www.w3.org/TR/1998/REC-xml-19980210.

[22] A. Broder, R. Kumar, F. Maghoul, P. Raghavan, S. Rajagopalan, R. Stata, A. Tomkins, and J. Wiener. Graph structure in the Web. In *Proc. of the 9th Int. World Wide Web conference on Computer networks : the international journal of computer and telecommunications networking*, pages 309–320. North-Holland Publishing Co., 2000.

[23] P. Buneman. Semistructured Data. In *Proc. of the 16th Symposium on Principles of Database Systems (PODS)*, pages 117–121. ACM Press, May 1997.

[24] P. Buneman, S. Davidson, G. Hillebrand, and D. Suciu. A Query Language and Optimization Techniques for Unstructured Data. *SIGMOD Record.*, 25(2):505–516, 1996.

[25] P. Buneman, W. Fan, and S. Weinstein. Path Constraints in Semistructured and Structured Databases. In *Proc. of the 17th Symposium on Principles of Database Systems (PODS)*, pages 129–138. ACM Press, June 1998.

[26] L. Cardelli, P. Gardner, , and G. Ghelli. A Spatial Logic for Querying Graphs. In *Proc. of the 29th Int. Colloquium on Automata, Languages, and Programming (ICALP)*, LNCS, pages 597–610. Springer, July 2002.

[27] A. K. Chandra. Theory of Database Queries. In *Proc. of the 7th Symposium on Principles of Database Systems (PODS)*, pages 1–9. ACM Press, March 1988.

[28] P. P.-S. Chen. The Entity-Relationship Model - Toward a Unified View of Data. *ACM Transactions on Database Systems (TODS)*, 1(1):9–36, 1976.

[29] J. Chomicki. Temporal Query Languages: A Survey. In *Proc. of the First Int. Conf. on Temporal Logic (ICTL)*, pages 506–534. Springer-Verlag, 1994.

[30] E. F. Codd. A Relational Model of Data for Large Shared Data Banks. *Communications of the ACM*, 13(6):377–387, 1970.

[31] E. F. Codd. Data Models in Database Management. In *Proc. of the 1980 Workshop on Data abstraction, Databases and Conceptual Modeling*, pages 112–114. ACM Press, 1980.

[32] E. F. Codd. A Relational Model of Data for Large Shared Data Banks. *Communications of the ACM*, 26(1):64–69, 1983.

[33] J. Conklin. Hypertext: An Introduction and Survey. *IEEE Computer*, 20(9):17–41, 1987.

[34] M. Consens and A. Mendelzon. Hy+: a Hygraph-based query and visualization system. *SIGMOD Record*, 22(2):511–516, 1993.

[35] M. P. Consens and A. O. Mendelzon. Expressing Structural Hypertext Queries in Graphlog. In *Proc. of the 2th Conf. on Hypertext*, pages 269–292. ACM Press, 1989.

[36] I. F. Cruz, A. O. Mendelzon, and P. T. Wood. A Graphical Query Language Supporting Recursion. In *Proc. of the Association for Computing Machinery Special Interest Group on Management of Data*, pages 323–330. ACM Press, May 1987.

[37] I. F. Cruz, A. O. Mendelzon, and P. T. Wood. G+: Recursive Queries without Recursion. In *Proc. of the 2th Int. Conf. on Expert Database Systems (EDS)*, pages 645–666. Addison-Wesley, April 1989.

[38] C. J. Date. Referential Integrity. In *Proc. of the 7th Int. Conf. on Very Large Data Bases (VLDB)*, pages 2–12. IEEE Computer Society, Sept 1981.

[39] D. J. de S. Price. Networks of Scientific papers. *Science*, 149:510–515, 1965.

[40] Y. Deng and S.-K. Chang. A G-Net Model for Knowledge Representation and Reasoning. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 2(3):295–310, Dec 1990.

[41] Y. Deville, D. Gilbert, J. van Helden, and S. J. Wodak. An Overview of Data Models for the Analysis of Biochemical Pathways. In *Proc. of the First Int. Workshop on Computational Methods in Systems Biology*, page 174. Springer-Verlag, 2003.

[42] S. N. Dorogovtsev and J. F. F. Mendes. *Evolution of Networks - From Biological Nets to the Internet and WWW*. Oxford University Press, 2003.

[43] M. Erwig and R. Güting. Explicit Graphs in a Functional Model for Spatial Databases. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 6(5):787–804, 1994.

[44] M. Fernández, D. Florescu, J. Kang, A. Levy, and D. Suciu. Catching the boat with Strudel: Experiences with a Web-site Management System. In *Proc. of the 1998 ACM SIGMOD Int. Conf. on Management of Data*, pages 414–425. ACM Press, June 1998.

[45] S. Flesca and S. Greco. Partially Ordered Regular Languages for Graph Queries. In *Proc. of the 26th Int. Colloquium on Automata, Languages and Programming (ICALP)*, volume 1644 of *LNCS*, pages 321–330. Springer, July 1999.

[46] S. Flesca and S. Greco. Querying Graph Databases. In *Proc. of the 7th Int. Conf. on Extending Database Technology - Advances in Database Technology (EDBT)*, volume 1777 of *LNCS*, pages 510–524. Springer, March 2000.

[47] D. Florescu, A. Levy, and A. O. .Mendelzon. Database Techniques for the World-Wide Web: A Survey. *SIGMOD Record*, 27(3):59–74, 1998.

[48] M. Gemis and J. Paredaens. An Object-Oriented Pattern Matching Language. In *Proc. of the First JSSST Int. Symposium on Object Technologies for Advanced Software*, pages 339–355. Springer-Verlag, 1993.

[49] M. Gemis, J. Paredaens, I. Thyssens, and J. V. den Bussche. GOOD: A Graph-Oriented Object Database System. In *Proc. of the 1993 ACM SIGMOD Int. Conf. on Management of Data*, pages 505–510. ACM Press, 1993.

[50] R. Giugno and D. Shasha. GraphGrep: A Fast and Universal Method for Querying Graphs. In *Proc. of the IEEE Int. Conf. in Pattern recognition (ICPR)*, Aug 2002.

[51] M. Graves. Graph Data Models for Genomics. *Submitted to ACM Transactions on Database Systems (TODS).*

[52] M. Graves. *Theories and Tools for Designing Application-Specific Knowledge Base Data Models.* PhD thesis, University of Michigan, 1993.

[53] M. Graves, E. R. Bergeman, and C. B. Lawrence. Querying a Genome Database using Graphs. In *In Proc. of the 3th Int. Conf. on Bioinformatics and Genome Research*, 1994.

[54] M. Graves, E. R. Bergeman, and C. B. Lawrence. A Graph-Theoretic Data Model for Genome Mapping Databases. In *Proc. of the 28th Hawaii Int. Conf. on System Sciences (HICSS)*, page 32. IEEE Computer Society, 1995.

[55] M. Graves, E. R. Bergeman, and C. B. Lawrence. Graph Database Systems for Genomics. *IEEE Engineering in Medicine and Biology. Special issue on Managing Data for the Human Genome Project*, 11(6), 1995.

[56] R. L. Griffith. Three Principles of Representation for Semantic Networks. *ACM Transactions on Database Systems (TODS)*, 7(3):417–442, 1982.

[57] A. Gutiérrez, P. Pucheral, H. Steffen, and J.-M. Thévenin. Database Graph Views: A Practical Model to Manage Persistent Graphs. In *Proc. of the 20th Int. Conf. on Very Large Data Bases (VLDB)*, pages 391–402. Morgan Kaufmann, Sept 1994.

[58] R. H. Güting. GraphDB: Modeling and Querying Graphs in Databases. In *Proc. of 20th Int. Conf. on Very Large Data Bases (VLDB)*, pages 297–308. Morgan Kaufmann, Sept 1994.

[59] M. Gyssens, J. Paredaens, J. V. den Bussche, and D. V. Gucht. A Graph-Oriented Object Database Model. In *Proc. of the 9th Symposium on Principles of Database Systems (PODS)*, pages 417–424. ACM Press, 1990.

[60] M. Gyssens, J. Paredaens, J. V. den Bussche, and D. V. Gucht. A Graph-Oriented Object Database Model. Technical Report 91-27, University of Antwerp (UIA), Belgium, March 1991.

[61] M. Gyssens, J. Paredaens, and D. V. Gucht. A Graph-Oriented Object Model for Database End-User Interfaces. In *Proc. of the 1990 ACM SIGMOD Int. Conf. on Management of data*, pages 24–33. ACM Press, 1990.

[62] J. Hammer and M. Schneider. The GenAlg Project: Developing a New Integrating Data Model, Language, and Tool for Managing and Querying Genomic Information. *SIGMOD Record*, 33(2):45–50, 2004.

[63] M. Hammer and D. McLeod. The Semantic Data Model: A Modelling Mechanism for Data Base Applications. In *Proc. of the 1978 ACM SIGMOD Int. Conf. on Management of Data*, pages 26–36. ACM, May 1978.

[64] R. A. Hanneman. Introduction to Social Network Methods. Technical report, Department of Sociology, University of California, Riverside, 2001.

[65] J. Hayes and C. Gutierrez. Bipartite Graphs as Intermediate Model for RDF. In *Proc. of the 3th Int. Semantic Web Conference (ISWC)*, number 3298 in LNCS, pages 47–61. Springer-Verlag, Nov 2004.

[66] A. Heuer and M. H. Scholl. Principles of Object-Oriented Query Languages. In *Datenbanksysteme in Büro, Technik und Wissenschaft (BTW)*, volume 270 of *Informatik-Fachberichte*, pages 178–197. Springer, March 1991.

[67] J. Hidders. *A Graph-based Update Language for Object-Oriented Data Models*. PhD thesis, Technische Universiteit Eindhoven,, Dec 2001.

[68] J. Hidders. Typing Graph-Manipulation Operations. In *Proc. of the 9th Int. Conf. on Database Theory (ICDT)*, pages 394–409. Springer-Verlag, 2002.

[69] J. Hidders and J. Paredaens. GOAL, A Graph-Based Object and Association Language. *Advances in Database Systems: Implementations and Applications, CISM*, pages 247–265, Sept 1993.

[70] R. Hull and R. King. Semantic Database Modeling: Survey, Applications, and Research Issues. *ACM Computing Surveys*, 19(3):201–260, 1987.

[71] H. V. Jagadish and F. Olken. Data Management for the Biosciences: Report of the NLM Workshop on Data Management for Molecular and Cell Biology. Technical Report LBNL-52767, National Library of Medicine, 2003.

[72] L. Kerschberg, A. C. Klug, and D. Tsichritzis. A Taxonomy of Data Models. In *Proc. of Systems for Large Data Bases (VLDB)*, pages 43–64. North Holland and IFIP, Sept 1976.

[73] N. Kiesel, A. Schurr, and B. Westfechtel. GRAS: A Graph-Oriented Software Engineering Database System. In *IPSEN Book*, pages 397–425, 1996.

[74] M. Kifer, W. Kim, and Y. Sagiv. Querying Object-Oriented Databases. In *Proc. of the 1992 ACM SIGMOD Int. Conf. on Management of data*, pages 393–402. ACM Press, 1992.

[75] W. Kim. Object-Oriented Databases: Definition and Research Directions. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 2(3):327–341, 1990.

[76] G. Klyne and J. Carroll. Resource Description Framework (RDF) Concepts and Abstract Syntax. http://www.w3.org/TR/2004/REC-rdf-concepts-20040210/, Feb 2004.

[77] R. Kumar, P. Raghavan, S. Rajagopalan, D. Sivakumar, A. Tomkins, and E. Upfal. The Web as a Graph. In *Proc. of the 19th Symposium on Principles of Database Systems (PODS)*, pages 1–10. ACM Press, May 2000.

[78] H. S. Kunii. DBMS with Graph Data Model for Knowledge Handling. In *Proc. of the 1987 Fall Joint Computer Conference on Exploring technology: today and tomorrow*, pages 138–142. IEEE Computer Society Press, 1987.

[79] G. M. Kuper and M. Y. Vardi. A New Approach to Database Logic. In *Proc. of the 3th Symposium on Principles of Database Systems (PODS)*, pages 86–96. ACM Press, April 1984.

[80] G. M. Kuper and M. Y. Vardi. The Logical Data Model. *ACM Transactions on Database Systems (TODS)*, 18(3):379–413, 1993.

[81] B. Langou and M. Mainguenaud. Graph Data Model Operations for Network Facilities in a Geographical Information System. In *Proc. of the 6th Int. Symposium on Spatial Data Handling*, volume 2, pages 1002–1019, 1994.

[82] C. Lécluse, P. Richard, and F. Vélez. O2, an Object-Oriented Data Model. In *Proc. of the ACM SIGMOD Int. Conf. on Management of Data*, pages 424–433. ACM Press, June 1988.

[83] M. Levene and G. Loizou. A Graph-Based Data Model and its Ramifications. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 7(5):809–823, 1995.

[84] M. Levene and A. Poulovassilis. The Hypernode Model and its Associated Query Language. In *Proc. of the 5th Jerusalem Conf. on Information technology*, pages 520–530. IEEE Computer Society Press, 1990.

[85] M. Levene and A. Poulovassilis. An Object-Oriented Data Model Formalised Through Hypergraphs. *Data & Knowledge Engineering (DKE)*, 6(3):205–224, 1991.

[86] M. Mainguenaud. Simatic XT: A Data Model to Deal with Multiscaled Networks. *Computer, Environment and Urban Systems*, 16:281–288, 1992.

[87] M. Mainguenaud. Modelling the Network Component of Geographical Information Systems. *Int. Journal of Geographical Information Systems (IJGIS)*, 9(6):575–593, 1995.

[88] M. V. Mannino and L. D. Shapiro. Extensions to Query Languages for Graph Traversal Problems. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 2(3):353–363, 1990.

[89] S. D. Martin S. Lacher. On the Integration of Topic Maps and RDF Data. In *First Semantic Web Workshop*. Stanford, 2001.

[90] W. C. McGee. On User Criteria for Data Model Evaluation. *ACM Transactions on Database Systems (TODS)*, 1(4):370–387, 1976.

[91] C. B. Medeiros and F. Pires. Databases for GIS. *SIGMOD Record*, 23(1):107–115, March 1994.

[92] A. O. Mendelzon and P. T. Wood. Finding Regular Simple Paths in Graph Databases. In *Proc. of the 15th Int. Conf. on Very Large Data Bases (VLDB)*, pages 185–193. Morgan Kaufmann Publishers Inc., 1989.

[93] S. B. Navathe. Evolution of Data Modeling for Databases. *Communications of the ACM*, 35(9):112–123, 1992.

[94] W. Nejdl, W. Siberski, and M. Sintek. Design Issues and Challenges for RDF- and Schema-Based Peer-to-Peer Systems. *SIGMOD Record*, 32(3):41–46, 2003.

[95] M. E. J. Newman. The Structure and Function of Complex Networks. *SIAM Review*, 45(2):167–256, 2003.

[96] F. Olken. Tutorial on Graph Data Management for Biology. *IEEE Computer Society Bioinformatics Conference (CSB)*, Aug 2003.

[97] Y. Papakonstantinou, H. Garcia-Molina, and J. Widom. Object Exchange across Heterogeneous Information Sources. In *Proc. of the 11th Int. Conf. on Data Engineering (ICDE)*, pages 251–260. IEEE Computer Society, 1995.

[98] J. Paredaens and B. Kuijpers. Data Models and Query Languages for Spatial Databases. *Data & Knowledge Engineering (DKE)*, 25(1-2):29–53, 1998.

[99] J. Paredaens, P. Peelman, and L. Tanca. G-Log: A Graph-Based Query Language. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 7(3):436–453, 1995.

[100] J. Peckham and F. J. Maryanski. Semantic Data Models. *ACM Computing Surveys*, 20(3):153–189, 1988.

[101] S. Pepper. The TAO of Topic Maps. http://www.ontopia.net/topicmaps/materials/tao.html.

[102] S. Pepper and G. Moore. XML Topic Maps (XTM) 1.0 - TopicMaps.Org Specification. http://www.topicmaps.org/xtm/1.0/xtm1-20010806.html, Feb 2001.

[103] A. Poulovassilis and S. G. Hild. Hyperlog: A Graph-Based System for Database Browsing, Querying, and Update. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 13(2):316–333, 2001.

[104] A. Poulovassilis and M. Levene. A Nested-Graph Model for the Representation and Manipulation of Complex Objects. *ACM Transactions on Information Systems (TOIS)*, 12(1):35–68, 1994.

[105] E. Prud'hommeaux and A. Seaborne. SPARQL Query Language for RDF, W3C Working Draft 21 July. http://www.w3.org/TR/2005/WD-rdf-sparql-query-20050721/, 2005.

[106] R. Ramakrishnan and J. D. Ullman. A Survey of Research on Deductive Database Systems. *Journal of Logic Programming (JLP)*, 23(2):125–149, 1993.

[107] N. Roussopoulos and J. Mylopoulos. Using Semantic Networks for Database Management. In *Proc. of the Int. Conf. on Very Large Data Bases (VLDB)*, pages 144–172. ACM, Sept 1975.

[108] R.V.Guha, O. Lassila, E. Miller, and D. Brickley. Enabling Inferencing. *The Query Languages Workshop (QL)*, Dec 1998.

[109] H. Samet and W. G. Aref. Spatial Data Models and Query Processing. In *Modern Database Systems*, pages 338–360. 1995.

[110] K.-D. Schewe, B. Thalheim, J. W. Schmidt, and I. Wetzel. Integrity Enforcement in Object-Oriented Databases . In *Proc. of the 4th Int. Workshop on Foundations of Models and Languages for Data and Objects*, Oct 1993.

[111] D. Shasha, J. T. L. Wang, and R. Giugno. Algorithmics and Applications of Tree and Graph Searching. In *Proc. of the 21th Symposium on Principles of Database Systems (PODS)*, pages 39–52. ACM Press, 2002.

[112] S. Shekhar, M. Coyle, B. Goyal, D.-R. Liu, and S. Sarkar. Data Models in Geographic Information Systems. *Communications of the ACM*, 40(4):103–111, 1997.

[113] L. Sheng, Z. M. Ozsoyoglu, and G. Ozsoyoglu. A Graph Query Language and Its Query Processing. In *Proc. of the 15th Int. Conf. on Data Engineering (ICDE)*, pages 572–581. IEEE Computer Society, March 1999.

[114] A. Sheth, B. Aleman-Meza, I. B. Arpinar, C. Halaschek-Wiener, C. Ramakrishnan, C. Bertram, Y. Warke, D. Avant, F. S. Arpinar, K. Anyanwu, and K. Kochut. Semantic Association Identification and Knowledge Discovery for National Security Applications. *Journal of Database Management (JDM)*, 16(1):33–53, Jan-March 2005.

[115] D. W. Shipman. The Functional Data Model and the Data Language DAPLEX. *ACM Transactions on Database Systems (TODS)*, 6(1):140–173, 1981.

[116] A. Silberschatz, H. F. Korth, and S. Sudarshan. Data Models. *ACM Computing Surveys*, 28(1):105–108, 1996.

[117] J. F. Sowa. Conceptual Graphs for a Database Interface. *IBM Journal of Research and Development*, 20(4):336–357, 1976.

[118] J. F. Sowa. *Conceptual Structures: Information Processing in Mind and Machine*. Reading, MA, Addison-Wesley, 1984.

[119] J. F. Sowa. *Principles of Semantic Networks: Explorations in the Representation of Knowledge*. Morgan Kaufmann Publishers, 1991.

[120] L. D. Stein and J. Tierry-Mieg. AceDB: A genome Database Management System. *Computing in Science & Engineering (CiSE)*, 1(3):44–52, 1999.

[121] A. Tansel, J. Clifford, S. Gadia, S. Jajodia, A. Segev, and R. T. Snodgrass, editors. *Temporal Databases: Theory, Design, and Implementation*. Benjamin-Cummings, 1993.

[122] R. W. Taylor and R. L. Frank. CODASYL Data-Base Management Systems. *ACM Computing Surveys*, 8(1):67–103, 1976.

[123] B. Thalheim. *Dependencies in Relational Databases*. Leipzig, Teubner Verlag, 1991.

[124] B. Thalheim. An Overview on Semantical Constraints for Database Models. In *Proc. of the 6th Int. Conf. Intellectual Systems and Computer Science*, Dec 1996.

[125] F. W. Tompa. A Data Model for Flexible Hypertext Database Systems. *ACM Transactions on Information Systems (TOIS)*, 7(1):85–100, 1989.

[126] D. C. Tsichritzis and F. H. Lochovsky. Hierarchical Data-Base Management: A Survey. *ACM Computing Surveys*, 8(1):105–123, 1976.

[127] M. Tsvetovat, J. Diesner, and K. Carley. NetIntel: A Database for Manipulation of Rich Social Network Data. Technical Report CMU-ISRI-04-135, Carnegie Mellon University, School of Computer Science, Institute for Software Research International, 2004.

[128] E. Tuv, A. Poulovassilis, and M. Levene. A Storage Manager for the Hypernode Model. In *Proc. of the 10th British National Conference on Databases*, number 618 in LNCS, pages 59–77. Springer-Verlag, 1992.

[129] M. Y. Vardi. The Complexity of Relational Query Languages (Extended Abstract). In *Proc. of the 14th ACM Symposium on Theory of Computing (STOC)*, pages 137–146. ACM Press, 1982.

[130] P. Vassiliadis and T. Sellis. A Survey of Logical Models for OLAP Databases. *SIGMOD Record*, 28(4):64–69, 1999.

[131] V. Vianu. A Web Odyssey: from Codd to XML. *SIGMOD Record*, 32(2):68–77, 2003.

[132] C. Watters and M. A. Shepherd. A Transient Hypergraph-Based Model for Data Access. *ACM Transactions on Information Systems (TOIS)*, 8(2):77–102, 1990.

[133] G. E. Weddell. Reasoning about Functional Dependencies Generalized for Semantic Data Models. *ACM Transactions on Database Systems (TODS)*, 17(1):32–64, 1992.

[134] B. Wellman, J. Salaff, D. Dimitrova, L. Garton, M. Gulia, and C. Haythornthwaite. Computer Networks as Social Networks: Collaborative Work,Telework, and Virtual Community. *Annual Review of Sociology*, 22:213–238, 1996.

[135] M. Yannakakis. Graph-Theoretic Methods in Database Theory. In *Proc. of the 9th Symposium on Principles of Database Systems (PODS)*, pages 230–242. ACM Press, 1990.