Survey of Graph Database Models

RENZO ANGLES and CLAUDIO GUTIERREZ

Universidad de Chile

Graph database models can be defined as those in which 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 took off in the eighties and early nineties alongside object-oriented models. Their influence gradually died out with the emergence of other database models, in particular geographical, spatial, semistructured, and XML. Recently, the need to manage information with graph-like nature has reestablished the relevance of this area. The main objective of this survey is to present the work that has been conducted in the area of graph database modeling, concentrating on data structures, query languages, and integrity constraints.

Categories and Subject Descriptors: G.2.2 [**Discrete Mathematics**]: Graph Theory—*Graph labeling*; *Hypergraphs*; *Network problems*; *Path and circuit problems*; H.2.1 [**Database Management**]: Logical Design—*Data models*; *schema and Subschema*; H.2.3 [**Database Management**]: Languages—*Query languages; Data description languages (DDL)*; *Data manipulation languages (DML)*; *Query languages*; H.2.5 [**Database Management**]: Heterogeneous Databases; H.2.8 [**Database Management**]: Database Applications

General Terms: Design, Languages, Management

Additional Key Words and Phrases: Database systems, graph databases, database models, graph database models, graph query languages, graph integrity constraints

1. INTRODUCTION

The term "data model" has been widely used in the information management community: it covers various meanings. In the most general sense, a data[base] model (db-model)[1] is a collection of conceptual tools used to model representations of

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

This research was supported by Millenium Nucleus, Center for Web Research (P01-029-F), Chile. R. Angles was supported by Mecesup project No. UCH0109. C. Gutierrez was partially supported by FONDECYT No. 1030810 and No. 1070348.

Author's address: Computer Science Department, Universidad de Chile, Blanco Encalada 2120, Santiago, Chile; email: rangles@dcc.uchile.cl.

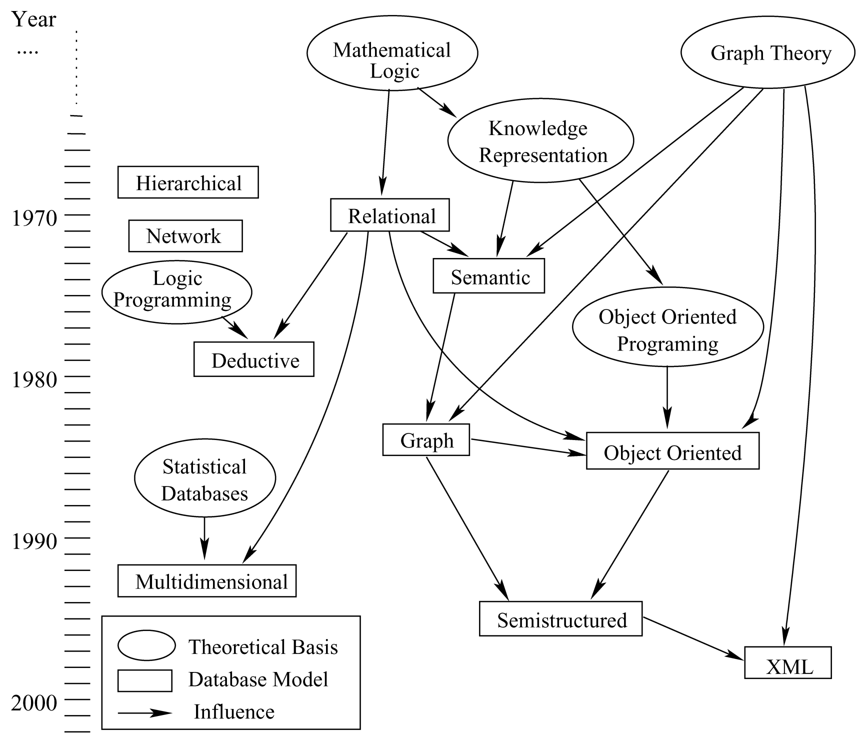


Fig. 1. Evolution of database models. Rectangles denote database models, arrows indicate influences, and circles denote theoretical developments. A time-line in years is shown on the left (based on a diagram by A. O. Mendelzon).

real-world entities and the relationships among them [Silberschatz et al. 1996]. The term is also often used to refer to a collection of data structure types. Mathematical frameworks for representing knowledge have also been called db-models [McGee 1976].

From a database point of view, the conceptual tools that make up a db-model should at least address data structuring, description, maintenance, and a way to retrieve or query the data. According to these criteria, a db-model consists of three components: a set of data structure types, a set of operators or inference rules, and a set of integrity rules [Codd 1980]. Note that several proposals for db-models only define the data structures, sometimes omitting operators and/or integrity rules.

Due to the philosophical and practical importance of conceptual modeling, db-models have become essential abstraction tools. Db-models can be used for: specifying permitted data types; supporting a general database design methodology; coping with database evolution; developing families of high-level languages for query and data manipulation; focusing on DBMS architecture; and being a vehicle for research into the behavioral properties of alternate data organization structures [Codd 1980].

Overview of Database Models. 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 existing db-models shows that there are many factors that influence their development (See Figure 1). Some of the more important factors are: the characteristics or structure of the domain to be modeled, the type of theoretical tools that interest the expected users, and of course, the hardware and software constraints.

In addition, each db-model proposal is based on certain theoretical principles, and serves as base for the development of related models.

Before the advent of the relational model, most db-models focused essentially on the specification of data structures on actual file systems. Two representative db-models are the hierarchical [Tsichritzis and Lochovsky 1976] and the network [Taylor and Frank 1976] models, both of which place emphasis on the physical level. Kerschberg et al. [1976] developed a taxonomy of db-models prior to 1976, comparing their mathematical structures and foundation, and the levels of abstraction used. The relational db-model was introduced by Codd [Codd 1970, 1983] and highlights the concept of abstraction levels by introducing the idea of separation between physical and logical levels. It is based on the notions of sets and relations. Due to its ease of use, it gained wide popularity among business applications. As opposed to previous models, semantic db-models [Peckham and Maryanski 1988] allow database designers to represent objects and their relations in a natural and clear manner, providing users with tools to faithfully capture the desired domain semantics. A well-known example is the entity-relationship model [Chen 1976]. Object-oriented db-models [Kim 1990] appeared in the eighties, when most of the research was concerned with so-called "advanced systems for new types of applications [Beeri 1988]." These db-models are based on the object-oriented paradigm and their goal is to represent data as a collection of objects, which are organized into classes, and are assigned complex values. Graph db-models made their appearance alongside object-oriented db-models. These models attempt to overcome the limitations imposed by traditional db-models with respect to capturing the inherent graph structure of data appearing in applications such as hypertext or geographic information systems, where the interconnectivity of data is an important aspect. Semistructured db-models [Buneman 1997] are designed to model data with a flexible structure, for example, documents and Web pages. Semistructured data is neither raw nor strictly typed, as in conventional database systems. These db-models appeared in the nineties. Closely related to them is the XML (eXtensible Markup Language) [Bray et al. 1998] model, which did not originate in the database community. Although originally introduced as a document exchange standard, it soon became a general purpose model, focusing on information with tree-like structure.

For a global view of db-models see Silberschatz et al. [1996], Navathe [1992], Beeri [1988], and Kerschberg et al. [1976].

Scope, Contributions and Organization of this Survey. The aim of this survey is to present the work that has been done in the area of graph database modeling. As such, this survey systematically analyzes the different developments in this area and indicates the relevant references, but does not evaluate the area.

The contributions and organization of this survey are as follows:

—the conceptualization of the notion of a graph db-model by proposing a definition that encompasses implicit and informal notions used in the literature (Section 2.1).

—a comparison of graph db-models with respect to other db-models, highlighting their importance as an area with its own motivations, applications and characteristic problems (Sections 2.2, 2.3 and 2.4);

—the definition of a set of typical characteristics of graph db-models and a comparative study of existing graph db-models based on these characteristics; we concentrate in presenting the main aspects of modeling: data structures, query languages and integrity constraints (Section 3);

—we present a uniform description of most relevant graph db-models (Appendix A).

We would like to warn the reader that there are several related topics that fall outside of the scope of this survey. Some of these topics are: graph visualization; graph data structures and algorithms for secondary memory; graph methods for databases; and in general, graph database system implementation.

On a side note, there are other important db-models as well as modeling frameworks that concern graph modeling, but due to the size limitations of this survey, are not covered here. Some of these are: temporal db-models [Tansel et al. 1993; Chomicki 1994], spatial db-models [Paredaens and Kuijpers 1998; Samet and Aref 1995], geographical information systems (GIS) [Shekhar et al. 1997; Aufaure-Portier and Trépied 1976], and multidimensional db-models [Vassiliadis and Sellis 1999]. Frameworks related to our topic, but not directly focusing on database issues are semantic networks [Sowa 1991; Griffith 1982], conceptual graphs [Sowa 1976, 1984], knowledge representation systems [Deng and Chang 1990], topic maps [Pepper and Moore 2001; ISO 1999], hypertext [Conklin 1987], and the recent family of models for representing ontologies on the Web, OWL [McGuinness and van Harmelen 2004].

2. GRAPH DATA MODELING

2.1. What is a Graph Data Model?

Although most papers on graph db-models use the term "graph data[base] model," few of them explicitly define the notion. Nevertheless, their views on what a graph db-model is do not differ substantially. In some cases, an implicit definition is given and compared to other models that involve graphs, like the semantic, object-oriented and semi-structured models.

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

—*Data and/or the schema* are represented by graphs, or by data structures generalizing the notion of graph (hypergraphs or hypernodes). There is a wide consensus on this, except for slight variations.

Let us review different authors' opinions on this issue. One approach is to encode the whole database using graphs [Güting 1994]. Levene and Loizou [1995] have a labeled directed graph as the single underlying data structure; the database consists of a single digraph. According to Kuper and Vardi [1984], a database schema for graph db-models is a directed graph, where leaves represent data and internal nodes represent connections among the data. In another approach, database schemas, instances, and rules are formalized using directed labeled graphs [Paredaens et al. 1995]. Kunii [1987] describes databases using a schema graph, which is a labeled directed graph. Another graph db-model formalized the representation of data structures using graphs, and stored these graphs in a database [Graves et al. 1995a]. Gyssens et al. [1990a] treat object-oriented database schemas and instances as graphs. Nodes of the instance graph represent the database objects. Hidders [2002] describes database instances and schemas using certain types of labeled graphs. Another approach is to organize the data into graphs [Amann and Scholl 1992]. Finally, Hidders and Paredaens [1993] use labeled graphs to represent schemas and instances.

An issue that concerns all graph db-models is the level of separation between schema and data (instances). In most cases, schema and instances can be clearly distinguished.

—*Data manipulation* is expressed by graph transformations [Gyssens et al. 1990a], or by operations whose main primitives are on graph features like paths, neighborhoods, subgraphs, graph patterns, connectivity, and graph statistics (diameter, centrality, etc.). Some db-models define a flexible collection of type constructors and operations, that are used to create and access the graph data structures [Graves et al. 1995a]. Another approach is to express all queries using a few powerful graph manipulation primitives [Güting 1994]. The operators of the language can be based on pattern matching: finding all occurrences of a prototypical piece of an instance graph [Hidders and Paredaens 1993].

—*Integrity constraints* enforce data consistency. These constraints can be grouped in schema-instance consistency, identity and referential integrity, and functional and inclusion dependencies. Examples of these are: labels with unique names [Graves et al. 1995b], typing constraints on nodes [Kuper and Vardi 1993], functional dependencies [Levene and Poulovassilis 1991], domain and range of properties [Klyne and Carroll 2004].

In summary, a graph db-model is a model in which the data structures for the schema and/or instances are modeled as a directed, possibly labeled, graph, or generalizations of the graph data structure, where data manipulation is expressed by graph-oriented operations and type constructors, and appropriate integrity constraints can be defined over the graph structure.

2.2. Why a Graph Data Model?

Graph db-models are applied in areas where information about data interconnectivity or topology is more important, or as important, as the data itself. In these applications, the data and relations among the data, are usually at the same level. Introducing graphs as a modeling tool has several advantages for this type of data.

—It allows for a more natural modeling of data. Graph structures are visible to the user and they allow a natural way of handling applications data, for example, hypertext or geographic data. Graphs have the advantage of being able to keep all the information about an entity in a single node and showing related information by arcs connected to it [Paredaens et al. 1995]. Graph objects (like paths and neighborhoods) may have first order citizenship; a user can define some part of the database explicitly as a graph structure [Güting 1994], allowing encapsulation and context definition [Levene and Poulovassilis 1990].

—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 users to express a query at a high level of abstraction. To some extent, this is the opposite of graph manipulation in deductive databases, where often, fairly complex rules need to be written [Güting 1994]. It is not important to require full knowledge of the structure to express meaningful queries [Abiteboul et al. 1997]. Finally, for purposes of browsing it may be convenient to forget the schema [Buneman et al. 1996].

—For implementation, graph databases may provide special graph storage structures, and efficient graph algorithms for realizing specific operations [Güting 1994].

2.3. Comparison with other Database Models

In this section, we compare graph db-models against the most influential database models. Table I presents a coarse-grained overview of this comparison. In the following text, we present the details of this comparison.

Physical db-models were the first to successfully manage large collections of data. Among the most important ones are the hierarchical [Tsichritzis and Lochovsky 1976] and network [Taylor and Frank 1976] models. These models lack a good abstraction level: it is difficult to separate the db-model from the actual implementation. The data

Table I. A Coarse-Grained Comparison of the Most Influential Database Models. We are Comparing the Following Aspects: Abstraction Level, Base Data Structure Used, and the Types of Information Objects on Which the Db-Model Focuses

Database model	**Abstraction level**	**Base data structure**	**Information focus**
Network	physical	pointers + records	records
Relational	logical	relations	data + attributes
Semantic	user	graph	schema + relations
Object-Oriented	physical/logical	objects	object + methods
Semistructured	logical	tree	data + components
Graph	logical/user	graph	data + relations

structures provided are not flexible, and not apt for modeling nontraditional applications. They permit database navigation at the record level by providing low-level operations that can be used to derive more abstract structures.

Relational db-model [Codd 1970, 1983] was introduced by Codd, and highlights the concept of abstraction levels by introducing a separation between the physical and logical levels. Gradually the focus shifted to modeling data as seen by applications and users [Navathe 1992]. This was the strength of the relational model, at a time when application domains managed relatively simple data (financial, commercial and administrative applications).

The relational model was a landmark development because it gave the data modeling discipline a mathematical foundation. 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. For example, the relational model is geared towards simple record-type data, where the data structure is known in advance (airline reservations, accounting, inventories, etc.). The schema is fixed, which makes it difficult to extend these databases. It is not easy to integrate different schemas, nor is it automatable. The query language cannot explore the underlying graph of relationships among the data, such as paths, neighborhoods, patterns.

Semantic db-models [Peckham and Maryanski 1988] appeared as there was a need to incorporate a richer and more expressive set of semantics into the database, from a user's viewpoint. Database designers can represent objects and their relations in a natural and clear manner (similar to the way users view an application) by using high-level abstraction concepts such as aggregation, classification, and instantiation, sub- and super-classing, attribute inheritance, and hierarchies [Navathe 1992].

In general, the extra semantics supports database design and evolution [Hull and King 1987]. A well-known example is the entity-relationship model [Chen 1976], which has become a basis for the early stages of database design. Other examples of semantic db-models are IFO [Abiteboul and Hull 1984] and SDM [Hammer and McLeod 1978]. Semantic db-models are relevant to graph db-model research because the semantic db-models reason about the graph-like structure generated by the relationships among the modeled entities.

Object-oriented (O-O) db-models [Kim 1990] appeared in the eighties, when the database community realized that the relational model was inadequate for data-intensive domains (knowledge bases, engineering applications). This research was motivated by the appearance of nonconventional database applications, involving complex data objects and complex object interactions, such as CAD/CAM software, computer graphics, and information retrieval.

According to the O-O programming paradigm on which these models are based, data is represented as a collection of objects that are organized into classes, and have complex values and methods. 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 [Abiteboul et al. 1997].

O-O db-models are similar to semantic models in that they provide mechanisms for constructing complex data by interrelating objects, and are fundamentally different in that they support forms of local behavior in a manner similar to object-oriented programming languages. For example, in O-O db-models identifiers are external to the objects and they remain invariant, whereas semantic models make up identifiers or keys based on internal attributes or internal properties; O-O supports information-hidding and encapsulation [Navathe 1992].

O-O db-models are related to graph db-models because of their explicit or implicit use of graph structures in definitions [Levene and Poulovassilis 1991; Andries et al. 1992; Gyssens et al. 1990a]. Nevertheless, there are important differences with respect to how each models the world. O-O db-models view the world as a set of complex objects having certain states (data), where interaction is via method passing. On the other hand, graph db-models view the world as a network of relations, emphasizing data interconnection, and the properties of these relations.

O-O db-models focus on object dynamics, their values and methods. Graph db-models focus instead on the interconnection, while maintaining the structural and semantic complexity of the data. Further comparison between O-O and graph db-models may be founded in Beeri [1988]; Kerschberg et al. [1976]; Navathe [1992]; and Silberschatz et al. [1996].

Semistructured db-models [Buneman 1997; Abiteboul 1997] were motivated by the increased existence of semistructured data (also called unstructured data), data exchange, and data browsing [Buneman 1997]. In semistructured data, the structure is irregular, implicit, and partial; the schema does not restrict the data, it only describes it, a feature that allows extensible data exchanges; the schema is large and constantly evolving; the data is self-describing, as it contains schema information [Abiteboul 1997].

Among the most representative models are OEM [Papakonstantinou et al. 1995], Lorel [Abiteboul et al. 1997], UnQL [Buneman et al. 1996], ACeDB [Stein and Tierry-Mieg 1999], and Strudel [Fernández et al. 1998]. Generally, semistructured data is represented using a tree-like structure. However, cycles among data nodes are possible, which leads to graph-like structures as in graph db-models. Some authors characterize semistructured data as rooted directed connected graphs [Buneman et al. 1996].

2.4. Graph Data Models: Motivations and Applications

Graph db-models are motivated by real-life applications where component interconnectivity is a key feature. We will divide these application areas into classical applications and complex networks.

Classical Applications. Many applications played a part in motivating the need for graph databases:

—generalizations of the classical db-models [Kuper and Vardi 1984]; classical models were criticized for their lack of semantics, the flatness of the permitted data structures, the difficulties the user has to “see” the data connectivity, and how difficult it is to model complex objects [Levene and Poulovassilis 1990];

—applications where data complexity exceeded the capabilities of the relational db-model also motivated the introduction of graph databases; for instance, managing transport networks (train, plane, water, telecommunications) [Mainguenaud 1995], and spatially embedded networks like highway, public transport [Güting 1994]. Several of these applications are now in the field of GIS and spatial databases;

—the limited expressive power of current query languages motivated the search for models that allowed better representation of more complex applications [Paredaens et al. 1995];

—limitations at the time of knowledge representation systems [Kunii 1987], and the need for intricate but flexible knowledge representation and derivation techniques [Paredaens et al. 1995];

—after observing that graphs have been an integral part of the database design process in semantic and object-oriented db-models, the idea of having a model where both data manipulation and representation are graph-based emerged [Gyssens et al. 1990a];

—the need for improving the functionality offered by object-oriented db-models [Poulovassilis and Levene 1994]; for example, CASE, CAD, image processing, and scientific data analysis software all fall into this category;

—graphical and visual interfaces, geographical, pictorial, and multimedia systems [Gyssens et al. 1990b; Consens and Mendelzon 1993; Sheng et al. 1999];

—software systems and integration [Kiesel et al. 1996];

—the appearance of on-line hypertext evidenced the need for other db-models, like the ones suggested by Tompa [1989], Watters and Shepherd [1990], and Amann and Scholl [1992]; together with hypertext, the Web created the need for a more apt model than the classical ones for information exchange.

Complex Networks. Several areas have witnessed the emergence of huge data networks with interesting mathematical properties, called complex networks [Newman 2003; Albert and Barabási 2002; Dorogovtsev and Mendes 2003]. The need for database management for certain types of these networks has been recently highlighted [Olken 2003; Jagadish and Olken 2003; Tsvetovat et al. 2004; Graves et al. 1995b]. Although it is not yet evident if we can group these databases into one category, we will present them in this manner. As in Newman's [2003] survey, we will subdivide this category into four subcategories: social networks, information networks, technological networks and biological networks. In the following text, we describe each subcategory via an example.

—In *social networks* [Hanneman 2001], nodes are people or groups, while links show relationships or flows among nodes. Some examples are friendships, business relationships, sexual contact patterns, research networks (collaboration, coauthorship), communication records (mail, telephone calls, email), computer networks [Wellman et al. 1996], and national security [Sheth et al. 2005]. There is growing activity in the area of social network analysis [Brandes 2005], and also in visualization and data processing techniques for these networks.

—*Information networks* model relations representing information flow, such as citations among academic papers [de S. Price 1965], World Wide Web (hypertext, hypermedia) [Florescu et al. 1998; Kumar et al. 2000; Broder et al. 2000], peer-to-peer networks [Nejdl et al. 2003], relations among word classes in a thesaurus, and preference networks.

—In *technological networks*, the spatial and geographical aspects of the structure are dominant. Some examples are the Internet (as a computer network), electric power grids, airline routes, telephone networks, delivery network (post office), and *Geographic Information Systems (GIS)* are today covering a big part of this area (roads, railways, pedestrian traffic, rivers) [Shekhar et al. 1997; Medeiros and Pires 1994].

—*Biological networks* represent biological information whose volume, management and analysis has become an issue due to the automation of the process of data gathering. A good example is the area of genomics, where networks occur in gene regulation, metabolic pathways, chemical structure, map order and homology relationships among species [Graves http://www.xweave.com/people/mgraves/pubs/]. There are other kinds of biological networks, such as food webs, neural networks, and soon. The reader can consult database proposals for genomics [Graves et al. 1995b; Graves http://www.xweave.com/people/mgraves/pubs/; Hammer and Schneider 2004], an overview of models for biochemical pathways [Deville et al. 2003], a tutorial on Graph Data Management for Biology [Olken 2003], and a model for Chemistry [Benkö et al. 2003].

It is important to stress that classical query languages offer little help when dealing with the type of queries needed in these 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 nearest neighbor degrees (strong pair correlations). In social networks, some measures are: distance, neighborhoods, clustering coefficient of a vertex, clustering coefficient of a network, betweenness, and size of giant connected components, and size distribution of finite connected components [Dorogovtsev and Mendes 2003]. Similar problems occur in the Semantic Web, where RDF queries would benefit from being able to reason about graph aspects [Angles and Gutierrez 2005].

3. GRAPH DATABASE MODELS

This section presents the work that has been done in the database community on graph db-models. It begins with a brief historical overview of the main developments in the area in the form of proposals of graph db-models. Then, a comparative study of the main features of graph db-models is presented.

Comparison among db-models are typically done by either using a set of common features [Fry and Sibley 1976; Tsichritzis and Lochovsky 1976; Peckham and Maryanski 1988] or by defining a general model used as a comparison basis [Hull and King 1987]. The evaluation presented in this survey follows a conceptual organization of modeling features by considering three general components in the database modeling process: (a) basic foundations of the model and the data structures available at schema and instance level; (b) approaches to enforce data consistency (integrity constraints); and (c) languages for querying and manipulating the database. The study emphasizes conceptual modeling issues rather than implementation aspects. Figure 4 summarizes the set of features and other information about graph db-models.

3.1. Brief Historical Overview

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 that did not have links to the graph database work in the nineties); the emergence of XML captured all the attention of those working on hypertext; people working on graph databases moved to particular applications like spatial data, Web, and 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.

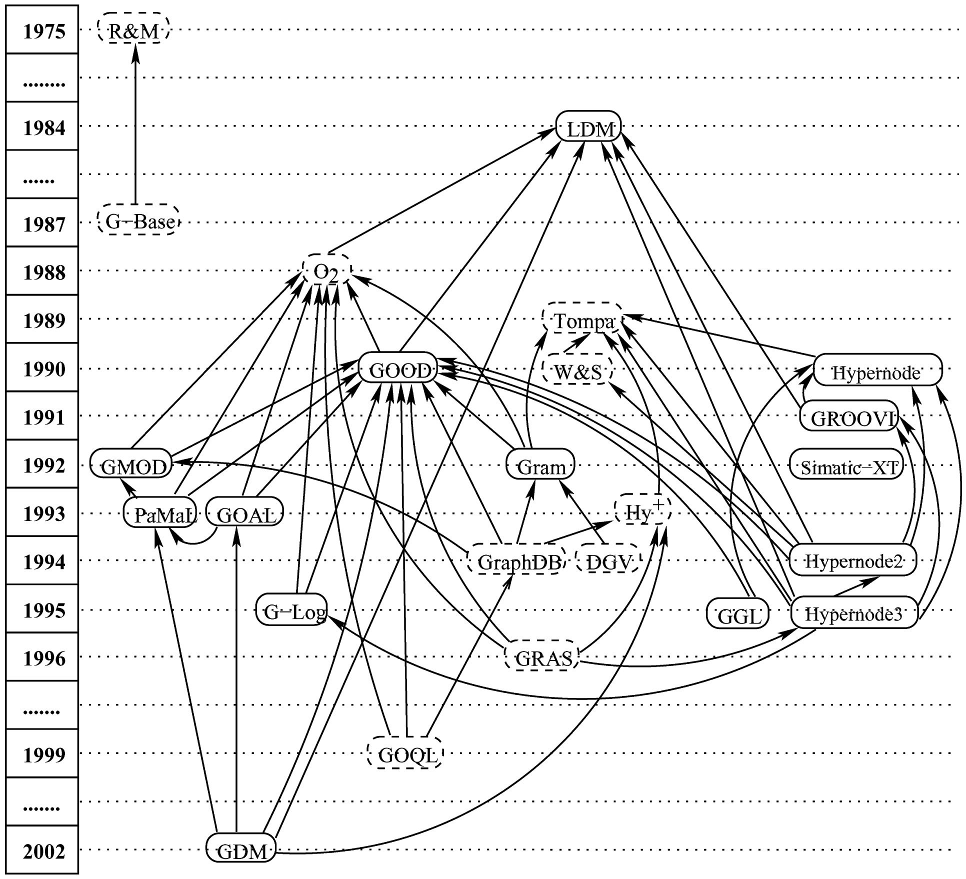


Fig. 2. Graph db-models development; nodes indicate models and arrows citations; dashed nodes represent related works in graph db-models: DGV [Gutiérrez et al. 1994], GDM [Hidders 2002], GGL [Graves et al. 1995a], GMOD [Andries et al. 1992], GOAL [Hidders and Paredaens 1993], GOOD [Gyssens et al. 1990a, 1990b, 1991], GOQL [Sheng et al. 1999], Gram [Amann and Scholl 1992], GRAS [Kiesel et al. 1996], GraphDB [Güting 1994], GROOVY [Levene and Poulovassilis 1991], G−Base [Kunii 1987], G−Log [Paredaens et al. 1995], Hypernode [Levene and Poulovassilis 1990], Hypernode2 [Poulovassilis and Levene 1994], Hypernode3 [Levene and Loizou 1995], Hy+ [Consens and Mendelzon 1993], LDM [Kuper and Vardi 1984; Kuper and Vardi 1993], O_2 [Lécluse et al. 1988], PaMaL [Gemis and Paredaens 1993], R&M [Roussopoulos and Mylopoulos 1975], Simatic-XT [Mainguenaud 1992], Tompa [Tompa 1989], W&S [Watters and Shepherd 1990].

In an early approach, Roussopoulos and Mylopoulos [1975] facing the failure of the systems of their time 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 [Shipman 1981], whose goal was to provide a "conceptually natural" database interface. A different approach proposed the Logical Data Model (LDM) [Kuper and Vardi 1984], where an explicit graph db-model intended to generalize the relational, hierarchical and network models. Later Kunii [1987] proposed a graph db-model for representing complex structures of knowledge called G-Base.

In the late eighties, Lécluse et al. [1988] introduced O_2, an object-oriented db-model based on a graph structure. Along the same lines, GOOD [Gyssens et al. 1990a] is an 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 [Andries et al. 1992], which proposes a number of concepts for graph-oriented database user interfaces; Gram [Amann and Scholl 1992], which is an explicit graph db-model for hypertext data; PaMaL [Gemis and Paredaens 1993], which extends GOOD with explicit representation of tuples and sets; GOAL [Hidders and Paredaens 1993], which introduces the notion of association nodes; G-Log [Paredaens et al. 1995], which proposed a declarative query language for graphs; and GDM [Hidders 2002], which incorporates representation of n-ary symmetric relationships.

There were proposals that used generalization of graphs with data modeling purposes. Levene and Poulovassilis [1990] introduced a db-model based on nested graphs, called the Hypernode Model, on which subsequent work was developed [Poulovassilis and Levene 1994; Levene and Loizou 1995]. The same idea was used for modeling multi-scaled networks [Mainguenaud 1992] and genome data [Graves et al. 1995a]. GROOVY [Levene and Poulovassilis 1991] 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 [Consens and Mendelzon 1993]; modeling of data instances and access to them [Watters and Shepherd 1990]; and representation of user state and browsing [Tompa 1989].

There are several other proposals that deal with graph data models. Güting [1994] proposed GraphDB, intended for modeling and querying graphs in object-oriented databases and motivated by managing information in transport networks. Database Graph Views [Gutiérrez et al. 1994] proposed an abstraction mechanism to define and manipulate graphs stored in either relational object-oriented or file systems. The project GRAS [Kiesel et al. 1996] uses attributed graphs for modeling complex information from software engineering projects. The well known OEM [Papakonstantinou et al. 1995] model aims at providing integrated access to heterogeneous information sources, focusing on information exchange.

Another important and recent line of development has to do with data representation models and the World Wide Web. Among them are data exchange models like XML [Bray et al. 1998], metadata representation models like RDF [Klyne and Carroll 2004] and ontology representation models like OWL [McGuinness and van Harmelen 2004].

3.2. Data Structures

The representation of entities and relations is fundamental to graph db-models. An entity or object represents something that exists as a single and complete unit. A relation is a property or predicate that establishes a connection between two or more entities. In this section, we analyze the data structures used for modeling entities and relations in graph db-models.

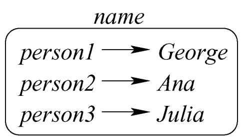
One of the most distinctive characteristic of a graph db-model is a framework for the representation of connectivity among entities, distinct from attributes (relational model), standard abstractions (semantic models), complex objects (O-O models), or composition relations (semistructured models).

All graph db-models have as their formal foundation, variations on the basic mathematical definition of a graph, for example, directed or undirected graphs, labeled or unlabeled edges and nodes, hypergraphs, and hypernodes. On top of this basic layer, models present diverse features influenced by the semantic or object-oriented approaches. For example, the data representation of GOOD, GMOD, G-Log and Gram is simply a digraph with labeled nodes and labeled edges between them. The Hypernode Model, Simatic-XT and GGL emphasize the use of hypernodes for representing nested complex objects. GROOVY is centered on the use of hypergraphs as a formalism for modeling complex objects, subobject sharing, integrity constraints, and structural inheritance.

Representing the database as a simple flat graph (with many interconnected nodes) has the drawback that, in practice, it is difficult to present the information to the user in a clear way. In contrast, a hypernode database consists of a set of nested graphs, which provides inherent support for data abstraction and the ability to represent each real-world object as a separate database entity [Poulovassilis and Levene 1994]. Additionally, the use of hypernodes and hypergraphs allows a concept to grow from a simple undefined concept (primitive object) to one defined by multiple complex relations (complex object). In contrast, a pure-graph approach (where nodes and edges are different structures) does not provide such expressiveness and extensibility.

Note that, hypergraphs can be modeled by hypernodes by (i) encapsulating the contents of each undirected hyperedge within a further hypernode, and (ii) replacing each directed hyperedge by two hypernodes related by a labeled edge. In contrast, multi-level nesting provided by hypernodes cannot be easily captured by hypergraphs.

Most models have explicit labels on edges, except LDM, GROOVY, and the Hypernode model. LDM has an order among the edges, that induces implicit labeling. Although the Hypernode model and GROOVY do not use labeled edges, the task of representing relations (and its names) can be attained by encapsulating edges, that represent the same relation (same label edges), within one hypernode (or hyperedge) labeled with the relation-name. For example, we can represent the set of labeled arcs $person1 \xrightarrow{name} George$, $person2 \xrightarrow{name} Ana$ and $person3 \xrightarrow{name} Julia$, by the hypernode:



Next, we will study in more detail the representation of entities and relations.

3.2.1. Representation of Entities. Models represent entities at both instance and schema levels. Specifically, entity types, relation types, and schema restrictions are related to the definition of the database schema, and instances of entities and relations conform the database instances.

Several models (e.g. GOOD, GMOD, G-Log and Gram) represent both schema and instance as a labeled digraph. An exception is LDM, whose schemas are digraphs, where leaves represent data, and internal nodes represent structured data. LDM instances consist of two-column tables, each of which associates entities of a particular type (primitive, tuple or set).

A *schema* graph defines: *entity types* represented as nodes labeled with zero (GMOD) or one type name; *primitive entities* represented as nodes labeled with basic types; and *relations* represented as edges labeled with relation-names. Relations are only defined for entity types (primitive entities have no properties) and each primitive entity has an associated domain of constants.

An *instance* graph contains: *concrete entities* represented as nodes labeled by either an entity type name or an object identifier; *primitive values* represented as nodes labeled with a value from the domain of a primitive entity; and *relations* represented as edges labeled with the corresponding relation-name according to the schema.

This approach was extended in other models by including nodes for explicit representation of tuples and sets (PaMaL, GDM), and n-ary relations (GOAL,GDM). These types of nodes allow the definition of complex structures. Tuple and set nodes of PaMaL are unlabeled, enabling the definition (instantiation) of more than one entity class (respectively concrete entity) using the same tuple and set node, providing data reduction. Association nodes of GOAL provide simple definition of multiattribute and multivalued n-ary relations. Composite-value entities in GDM are used for representing both tuples and n-ary relations.

The basic structure of a graph (nodes and edges) is complemented with the use of hypernodes (Hypernode model, Simatic-XT and GGL) and hypergraphs (GROOVY), extensions that provide support for nested structures. A novel feature of GROOVY is the use of hypergraphs for defining value functional dependencies. The hypernode model is characterized by using nested graphs at the schema and instance levels. A database consists of a set of hypernodes defining types and their respective instances. GGL introduces, in addition to its support for hypernodes (called Master-nodes), the notion of Master-edge for encapsulation of paths.

Although hypernodes are used in several models, they are used in different forms. Simatic-XT and GGL use hypernodes as an abstraction mechanism consisting in packaging other graphs as an encapsulated vertex, whereas the Hypernode model additionally uses hypernodes to represent other abstractions, for example, complex objects, relations, and so forth.

A common characteristic of models based on simple, as well as extended graph structures, is the support for defining nontraditional data types, features procured by the definition of complex objects. In this sense, LDM, PaMaL, GOAL and GDM allow the representation of complex objects by defining special constructs (tuple, set, or association nodes). Hypernodes and Hypergraphs are flexible data structures that support the representation of arbitrarily complex objects and present the inherent ability to encapsulate information.

3.2.2. Representation of Relations. Roughly speaking, two types of relations occurring in graph db-models can be distinguished: Simple relations, which connect two entities under a simple semantics (e.g. attributes), and are easily represented in graphs as edge labels; and complex relations, which conform networks of relations (e.g. hierarchical) with additional semantics (e.g. composition), and whose representation depends on the data structures provided by each model.

In what follows, we will discuss specific types of relations supported by graph db-models.

—*Attributes*. This relation represents a property (monovalued or multivalued) directly linked to an entity. Most graph db-models represent attributes by using labeled edges directly related to nodes.

  LDM and PaMaL define tuple nodes to describe a set of attributes that are (re)used to define an entity. In the case of the Hypernode model and GROOVY, attributes are triples $< node, edge, node >$ inside hypernodes or hyperedges, which represent complex objects. The (unlabeled) edge establishes the relation between the attribute-name (first node) and the attribute-value (second node). GOOD and GOAL define edges (called functional and nonfunctional) to distinguish between monovalued and multivalued attributes.

—*Entities*. If the relation of two or more objects conceptually describes a distinct model object, the relation is considered an entity. This approach implies support for higher-order relations (relations among relations). Most models do not support this feature because relations are represented as simple labeled edges.

  Partial support is presented in GOAL, where association nodes may have properties. The Hypernode model and GROOVY have inherent support because property-names can be complex objects represented as hypernodes or hyperedges. GGL supports this feature by labeling edges with types.

—*Neighborhood relations*. Models with a basic graph structure provide simple support and visualization of neighborhood relations, although special structures used by some of these models (e.g. tuple-nodes in PaMaL) can obscure the simplicity of representation. In the case of hypergraph and hypernode base models, neighborhood relations are translated into nested relations. A particular modeling structure is the Master-edge of Simatic-XT, which is used for representing *Path relations*.

—*Standard abstractions*. The most frequently used are aggregation (is-part-of relation) and its opposite, composition (is-composed-by relation), association (n-ary relations), and grouping or sets. Some models provide explicit representation of tuples (LDM,PaMaL,GDM), sets (LDM,PaMaL) and n-ary relations (GOAL, GDM). Models based on hypernodes and hypergraphs support grouping by joining entities within a hypernode or a hyperedge.

—*Derivation and inheritance*. These abstractions are represented at the schema level by relations of subclass and superclass (ISA) and at the instance level by relations of instantiation (is-of-type). The notion of inheritance (only structural-attribute inheritance) is supported using these relations, allowing entity classes with overlapping structure to share their semantic content.

ISA relations are explicitly supported in PaMaL, GOAL, and GDM by considering ISA-labeled edges as class-hierarchy links. However, we should note that this type of relation can be implicitly considered as an attribute relation. The Hypernode and GROOVY models show how structural inheritance is supported naturally by nested-graph structures. The representation of object-class schemas by means of hypergraphs leads to a natural formalization of the notion of object sharing and structural inheritance.

—*Nested relations*. These are recursively specified relations defined by nested objects. This feature is naturally supported by using hypernode or hypergraph structures.

3.3. Integrity Constraints

Integrity constraints are general statements and rules that define the set of consistent database states, or changes of state, or both [Codd 1980]. Integrity constraints have been studied for the relational [Date 1981; Thalheim 1991], semantic [Weddell 1992; Thalheim 1996], object-oriented [Schewe et al. 1993], and semistructured [Buneman et al. 1998; Alechina et al. 2003] db-models. Thalheim [1996] presents a unifying framework for integrity constraints.

In the case of graph db-models, examples of integrity constraints include schema-instance consistency, identity and referential integrity constraints; functional and inclusion dependencies. Next we study each of them.

3.3.1. Schema-Instance Consistency.

Entity types and type checking constitute a powerful data-modeling and integrity checking tool, since they allow database schemas to be represented and enforced. Some graph db-models do not define a schema, for example, the first version of Hypernode, Simatic-XT, and initial versions of GGL.

The approach applied in most models (GOOD, GMOD, PaMaL, GOAL, G-log, GDM and Gram) to confront the issue of schema-instance consistency in general follows two guidelines: (i) the instance should contain only concrete entities and relations from entity types and relations that were defined in the schema; (ii) an entity in the instance may only have those relations or properties defined for its entity type (or for a super-type in the case of inheritance). Then, all node and edge labels occurring in the instance must also occur in the scheme; but the opposite is not required in some models (GOOD, GMOD, GOAL and G-Log). In this way, incomplete or nonexisting information can be incorporated in the database. PaMaL, GDM, and Gram do not support incomplete information because they establish an instantiation function between schema and instance.

LDM defines a logic (similar to relational tuple calculus) used to specify integrity constraints on LDM schemas and to define views. Integrity constraints are LDM formulas, which enforce that instance objects satisfy certain conditions (satisfaction of LDM formulas). That is, given a database and a sentence in the logic, one can effectively test whether or not the sentence is true in the database.

GROOVY introduces the notion of object-class schemas over which objects are defined. An object schema defines valid objects (value-schemas), value functional dependencies, and valid shared values among objects (sub-object schemas). These restrictions are formalized using a hypergraph representation. Indeed, there is a one-to-one correspondence between each object schema and a hypergraph, where objects are vertices, value functional dependencies are directed hyperedges, and sub-object schemas are undirected hyperedges.

Additionally, conflicts of inheritance are discussed in some models. The general approach is the notion of consistent schema to prevent problems. GOAL establishes that objects only belong to the class they are labeled with, and their super-classes. This implies that an object can only belong to two classes at once if these classes have a common subclass. PaMaL does not provide a conflict resolution mechanism, since the authors consider that this has to be part of the implementation.

In Poulovassilis and Levene [1994], the authors show that testing a hypernode repository for well-typedness can be performed in polynomial time with respect to the size of the repository. GDM presents a systematic study of the complexity of well-typedness checking: deciding well-typedness of a pattern with no ISA edges under a schema graph with no implicit object class nodes, is in PTIME; deciding well-typedness of a pattern under a schema graph with no implied object class nodes is co-NP complete; deciding well-typedness of an addition/deletion under a schema graph with no implied object class nodes, is in PSPACE; deciding well-typedness of an addition under a schema graph with no implied object class nodes and no composite-value classes is in PTIME.

Checking consistency is also related to restructuring, queries and updates of the database, for example, deletion of entity types or relations in the schema implies the deletion of concrete entities in the instance. Given an arbitrary GOOD program: a sequence of GOOD operations, statically checking the consistency of an edge addition in the program is undecidable in general. In GMOD the notion of schema-instance consistency is referenced to consistency of schema transformations in object-oriented databases [Zicari 1991]. In PaMal an addition or deletion operation specifies a valid graph transformation between two instance graphs (schema transformations are not included).

Schema-instance separation. Another aspect to consider is the degree to which schema and instance are different objects in the database (an issue discussed largely in semistructured db-models). Representation of (un)structured entities means that the data (does not) respect a previously defined data type. In this sense, a data model can be classified as structured or nonstructured depending on whether it allows or does not allow the definition of a schema to restrict the database instance to well-typed data. In most models there is a separation between the database schema and the database instance. An exception is the hypernode model presented in Levene and Loizou [1995], where the lack of typing constraints on hypernodes has the advantage that changes to the database can be dynamic.

Redundancy of data. Because an instance graph can contain many instances of entities, for example, nonprintable nodes (GOOD), set/tuple nodes (PaMaL), association nodes (GOAL), or composite-value nodes (GDM), the database presents redundant information. One possible solution is to introduce an operation that is used to group entities on the basis of some of their relations. This abstraction creates a unique entity for each equivalence class of duplicate entities; as such, it acts as a duplicate eliminator. The corresponding operations are Abstraction (GOOD), Reduce (PaMaL) and Reduction (GDM). This operation is informally mentioned in GOAL as a merging of nodes representing the same value.

3.3.2. Object Identity and Referential Integrity. Set-based data models such as the relational model and the nested relational model are value-based; that is, tuples in relations (respectively nested relations) are identified by the values of their attributes. On the other hand, in O-O db-models, object identity is independent of attribute values and is achieved by equipping each database object with a unique identifier, for example, a label. Graph db-models implement the two types of identification, but several advantages accrue with identifiers: arbitrarily complex objects can be identified and referred; objects can share common subobjects, thus making possible the construction of general networks; query and update operation can be simplified.

In some models, concrete entities like instance nodes (PaMaL, Gram), edges (GGL), hypernodes (Hypernode based models), hyperedges (GROOVY) are labeled with entity identifiers, whereas in models like GOOD, GMOD, GOAL, G-Log and GDM, each entity is identified by its attributes, although it exists independently of their properties (it is considered as object identity).

The Hypernode Model [Poulovassilis and Levene 1994] defines two integrity constraints: *Entity Integrity* assures that each hypernode is a unique real world entity identified by its content; *Referential Integrity* requires that only existing entities be referenced. Similarly, GGL [Graves et al. 1995a] establishes that: labels in a graph are uniquely named; 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.

3.3.3. Functional Dependencies. In the Hypernode model [Levene and Loizou 1995] the notion of semantic constraints was considered. The concept of *Hypernode functional dependency*, denoted by $A \to B$, where A and B are sets of attributes, let us express that A determines the value of B in all hypernodes of the database.

GROOVY [Levene and Poulovassilis 1991] 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 object value restricted to a set of attributes uniquely determines the object value restricted to a further attribute.

Finally, let us remark that the notion of integrity constraint in graph db-models is concentrated in the creation of consistent instances and the correct identification and reference of entities. The notion of functional dependency is a feature taken from the relational model, which cannot be easily presented in all graph db-models.

3.4. Query and Manipulation Languages

A query language is a collection of operators or inferencing rules that 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 [Codd 1980]. Many papers discuss the problems concerning the definition of a query language for a db-model [Vardi 1982; Hull and King 1987; Ramakrishnan and Ullman 1993; Heuer and Scholl 1991; Abiteboul 1997; Abiteboul and Vianu 1997]. Also a variety of query languages and formal frameworks for studying them have been proposed and developed, including the relational db-model [Chandra 1988], semantic databases [Azmoodeh and Du 1988; Andries and Engels 1993], object-oriented databases [Kifer et al. 1992], semistructured data [Buneman et al. 1996; Abiteboul 1997; Abiteboul et al. 1997], and the Web [Abiteboul and Vianu 1997; Florescu et al. 1998].

Among graph db-models, there is substantial work focused on query languages, the problem of querying graphs, the visual presentation of results, and graphical query languages. Due to the volume of the research done, this specific area by itself deserves a thorough survey. Considering the scope of this survey, in this section we limit ourselves to describing the most important graph query languages to serve as reference resources.

The Logical Database Model [Kuper and Vardi 1984, 1993] presents a logic language 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.

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

—The graphical query language G [Cruz et al. 1987] 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 that match subgraphs from the instance.

—G evolved into a more powerful language called G+ [Cruz et al. 1989], in which 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, which represents how to restructure the answer obtained by the query graph.

—GraphLog [Consens and Mendelzon 1989] 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, to 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 Mendelzon and Wood [1989].

The proposal G-Log [Paredaens et al. 1995], includes a declarative language for complex objects with identity. It uses the logical notion of *rule satisfaction* to evaluate queries that are expressed as G-Log programs. *G-Log programs* are sets of graph-based rules, which specify how the schema and instance of the database will change. G-Log is a graph-based, declarative, nondeterministic, and computationally complete query language that does not suffer from the copy-elimination problem. G-Log is a good example of graphical query language (see Figure 3).

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 [Gyssens et al. 1990a] 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 [Andries et al. 1992], PaMaL [Gemis and Paredaens 1993], GOAL [Hidders and Paredaens 1993], and GUL [Hidders 2002]. Additionally, GOAL includes the notion of

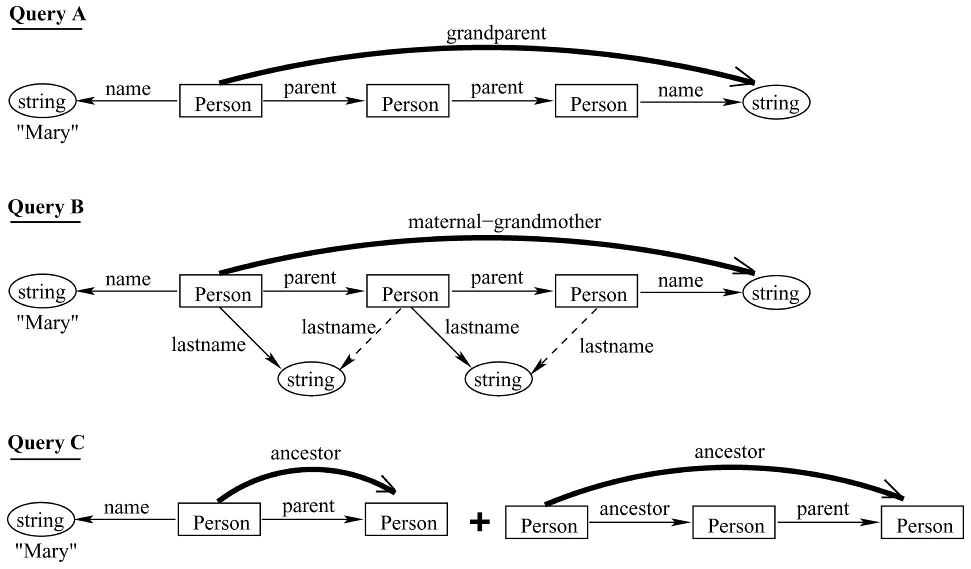


Fig. 3. Example of a graphical query language. The figure shows a G-Log query for the instance in Figure 12. Query A asks for the names of Mary's grandparents (fixed path query). Query B asks for the name of the maternal grandmother of Mary (tree-like query). Query C calculates Mary's Ancestors (transitive closure).

fixpoints in order to handle the recursion derived from a finite list of additions and deletions. PaMaL proposed the inclusion of loop, and procedure and programs constructs, and PaMaL and GUL presented an operator that reduces instance graphs by deleting repeated data. We should note that graph-oriented manipulation formalisms based on patterns allow a syntax-directed way of working much more naturally than text-based interfaces.

GROOVY [Levene and Poulovassilis 1991] 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 and deletion) of hypergraphs and hyperedges.

Watters and Shepherd [1990] present a framework for general data access based on hypergraphs that include operators for creation of edges and set operators like intersection, union, and difference. In a different context, Tompa [1989] 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:

—Levene and Poulovassilis [1990] define a logic-based query and update language, where queries are expressed as sets of hypernode rules (h-rules) that are called hypernode programs. The query language defines an operator that infers new hypernodes from the instance, using the set of rules in a hypernode program.

—This query language was extended by Hyperlog [Poulovassilis and Levene 1994; Poulovassilis and Hild 2001], 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. Hyperlog is computationally and update complete, although the evaluation of Hyperlog programs is intractable in the general case.

—In a procedural style, HNQL [Levene and Loizou 1995] 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.

Glide [Giugno and Shasha 2002] is a graph query language where queries are expressed using a linear notation formed by labels and wild-cards (regular expressions). Glide uses a method called GraphGrep based on subgraph matching to answer the queries.

Oriented to search the Web, Flesca and Greco [1999] 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 Flesca and Greco [2000].

Cardelli et al. [2002] introduced a spatial logic for reasoning about graphs, and defined a query language based on pattern matching and recursion. This graph logic combines first-order logic with additional structural connectives. A query asks for a substitution of variables such that a satisfaction relation determines which graphs satisfy which formulae. The query language is based on queries that build new graphs from old, and transducers that relate input graphs with output graphs.

In the area of geographic information systems, the Simatic-XT model [Langou and Mainguenaud 1994] defines a query language. It includes basic operators that deal with encapsulated data (nesting of hypernodes), set operators (union, concatenation, selection, and difference), and high level operators (paths, inclusion, and intersections).

WEB [Graves 1993; Graves et al. 1994] 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 relations in the database. These operations are answered by matching graphs in valid instances.

Models like Gram [Amann and Scholl 1992] and GOQL [Sheng et al. 1999] 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 [Gutiérrez et al. 1994] and GraphDB [Güting 1994] define special operators for functional definition and querying of graphs. For example, a query in GraphDB consists of several steps, each of which computes operations that specify argument subgraphs in the form of regular expressions over edges that dynamically extend or restrict 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 translates into path problems characterized and expressed by recursive queries. For example, path queries are relevant in GraphLog, Gram, Simatic-XT, DGV, GOQL, Flesca and Greco [1999], Cardelli et al. [2002], and to a lesser in Hypernode, GOAL, Hyperlog, GraphDB, G-Log, and HNQL. The notion of shortest path is considered in Flesca and Greco [2002], G+, GraphLog, and DGV. Path and other relevant graph queries for RDF are discussed in Angles and Gutierrez [2005].

The importance and computational complexity of path-based queries is studied in several works [Agrawal and Jagadish 1994, 1989,1988; Guha et al. 1998]. Finding simple paths with desired properties in direct graphs is difficult, and essentially every nontrivial property gives rise to an NP-complete problem [Shasha et al. 2002].

Yannakakis [1990] surveyed a set of path problems relevant to the database area including computing transitive closures, recursive queries, and the complexity of path searching. Mannino and Shapiro [1990] present a survey of extensions to database query languages that solve graph traversal problems.

APPENDIX

A. REPRESENTATIVE GRAPH DATABASE MODELS

In this section we describe the most representative graph db-models mentioned in Table I and other related models that do not fit properly as graph db-models, but use graphs, for example, for navigation, for defining views, or as language representation.

For each proposal, we present their data structures, query language, and integrity constraint rules. In general, there are few implementations and no standard benchmarks, hence we avoid surveying implementations. (For information about the existence of implementations see Figure 4.) To give a flavor of the modeling in each proposal, we will use as a running example, the toy genealogy shown in Figure 5.

A.1. Logical Data Model (LDM)

Motivated by the lack of semantics in the relational db-model, Kuper and Vardi [1984] proposed a db-model that generalizes the relational, hierarchical, and network models. The model describes mechanisms to restructure data, plus logical and algebraic query languages.

In LDM a *schema* is an arbitrary directed graph where each node has one of the following types: the *Basic* type □ describes a node that contains the stored data; the *Composition* type ⩀ describes a node that contains tuples whose components are taken from its children; the *Collection* type ◯ describes a node that contains sets, whose elements are taken from its child. Internal nodes are of type ⊗ or ⊛, representing structured data, terminal nodes are of type □, and represent atomic data, and edges represent connections among data.

A second version of the model [Kuper and Vardi 1993], besides renaming the nodes Composition and Collection as Product ⊗ and Power ⊛ respectively, incorporates a new type, the Union type ⨃, intended to represent a collection whose domain is the union of the domains of its children (see example in Figure 6).

An LDM database instance consists of an assignment of values to each node of the schema. 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 ⊗, ⊛ 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 enable modeling of transitive relations like hierarchies and genealogies.

A first order many-sorted language is defined over this structure. A query language and integrity constraints are defined with this language. Finally, an 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 (data structures plus query languages and integrity constraints) that supports modeling of complex relations, for example, hierarchies and recursive relations. The notion of virtual records (pointers to physical records) proves useful to avoid redundancy of data by allowing cyclicity at the schema and instance

Characteristics | Database Model		LDM	Hypernode	GOOD	GROOVY	GMOD	Simatic-XT	Gram	Pamal	GOAL	Hypernode2	Hypernode3	GGL	GDM
		1984	1990	1990	1991	1992	1992	1992	1993	1993	1994	1995	1995	2002
Basic Foundation	Graph model	√		√		√		√	√	√			√	√
	Hypergraphs				√									
	Hypernodes		√				√				√	√	√	
	Object Oriented model	√		√	√	√			√	√				√
Digraph	Node labeled	√	√	√	√	√	√	√	√	√	√	√	√	√
	Edge labeled			√	√	√	√	√	√	√			√	√
Support:	Schema	√		√	√	√		√	√	√	√	√		√
	Complex objects	√	√		√		√		√	√	√	√	√	√
	Higher-order relations		√		√					±	√	√	√	√
	Tuples	√							√					√
	Sets	√							√					
	N-ary relations									√				√
	Grouping		√		√		√				√	√	√	
	Derivation and Inheritance				√				√	√	±			√
	Nested relations		√		√		√				√	√	√	
Query Language	Algebraic - Procedural	√					√	√				√		
	Logic - Declarative	√	√		√						√		√	
	Graphical			√		√			√	√				√
	Query	√	√		√		√	√	√		√	√	√	
	Transformation	√		√		√				√				√
	Path queries		±	±			√	√	±	±	±	±	±	
Integrity Constraints	Schema-Instance Consistency	√		√	√	√		√	√	√	√	√		√
	Identity and Referential Integrity		√	±	√	±		√	√	±	√	√	√	±
	Functional Dependencies				√							√		
Implementation			√	√			√	√			√	√	√	√
Motivation		Complex objects	Complex objects	Graphical Interfaces	General	Hypermedia	Transport networks	Hypertext	Graphical Interfaces	Graphical Interfaces	Complex objects	Hypertext	Genomics	Complex objects

Fig. 4. Main proposals on Graph Database Models and their characteristics ("$\surd$" indicates support and "±" partial support). LDM [Kuper and Vardi 1984, 1993], Hypernode [Levene and Poulovassilis 1990], GOOD [Gyssens et al. 1990a, 1990b, 1991], GROOVI [Levene and Poulovassilis 1991], GMOD [Andries et al. 1992], Simatic-XT [Mainguenaud 1992], Gram [Amann and Scholl 1992], PaMaL [Gemis and Paredaens 1993], GOAL [Hidders and Paredaens 1993], Hypernode2 [Poulovassilis and Levene 1994], Hypernode3 [Levene and Loizou 1995], GGL [Graves et al. 1995a], GDM [Hidders 2002]. A working description of each model is given in Appendix A.

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

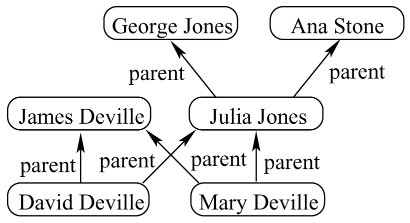
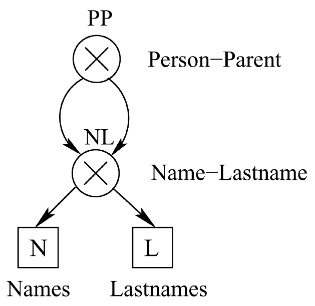


Fig. 5. A genealogy diagram (right-hand side) represented as two tables (left-hand side) NAME-LASTNAME and PERSON-PARENT (Children inherit the lastname of the father just for modeling purposes).

Schema



Instance

I (N)

l	*val (l)*
1	George
2	Ana
3	Julia
4	James
5	David
6	Mary

I (L)

l	*val (l)*
7	Jones
8	Stone
9	Deville

I (NL)

l	*val (l)*
10	(1,7)
11	(2,8)
12	(3,7)
13	(4,9)
14	(5,9)
15	(6,9)

I (PP)

l	*val (l)*
16	(12,10)
17	(12,11)
18	(14,13)
19	(14,12)
20	(15,13)
21	(15,12)

Fig. 6. 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 among 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 values defined by other nodes.

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.

A.2. Hypernode Model

The Hypernode db-model was described in a sequence of papers [Levene and Poulovassilis 1990; Poulovassilis and Levene 1994; Levene and Loizou 1995]. A hypernode is a directed graph whose nodes can themselves be graphs (or hypernodes), allowing nesting of graphs. 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.

The Hypernode model was introduced by Levene and Poulovassilis [1990], who define the model and a declarative logic-based language structured as a 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 Tuv et al. [1992].

In a second version Poulovassilis and Levene [1994], introduce the notion of schema and type checking via the idea of types (primitive and complex), that are also represented by nested graphs. (See an example in Figure 7.) The model is completed with entity and referential integrity constraints over a hypernode repository. Moreover it presents a rule-based query language called Hyperlog, which can support both querying and browsing with derivations as well as database updates, and is intractable in the general case.

A third version of the model [Levene and Loizou 1995] discusses a set of constraints (entity, referential, and semantic) over hypernode databases and introduces the concept

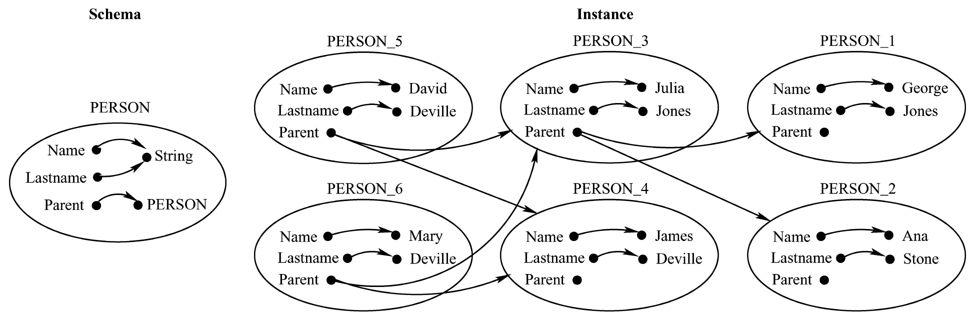


Fig. 7. Hypernode Model. The schema (left) defines a *person* as a complex object with the properties name and lastname of type string, and parent of type person (recursively defined). The instance (on the right) shows the relations in the genealogy among different instances of person.

of hypernode functional dependency (HDF), denoted by $A \rightarrow B$, where A and B are sets of attributes, and A determines the value of 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.

Summarizing, the main features of the Hypernode model are: it is based on a nested graph structure that is simple and formal; it has the ability to model arbitrary complex objects in a straightforward manner; it can provide the underlying data structure of an object-oriented data model; it can enhance the usability of a complex objects database system via a graph-based user interface.

The drawbacks are that data redundancy can be generated by its basic value labels, and restrictions in the schema level are limited. For example, it is not possible to specify restrictions for missing information or multivalued relations.

A.3. Hypergraph-Based Data Model (GROOVY)

GROOVY (Graphically Represented Object-Oriented data model with Values [Levene and Poulovassilis 1991]) is a proposal for a object-oriented db-model, which is formalized using hypergraphs. That is, a generalization of graphs where the notion of edge is extended to hyperedge, which relates an arbitrary set of nodes [Berge 1973]. An example of hypergraph schema and instance is presented in Figure 8.

The model defines a set of structures for an object data model: value schemas, objects over value shemas, value functional dependencies, object schemas, object over object schemas, and class schemas. The model shows that these structures can be defined in terms of hypergraphs.

A *Value Schema* defines the attributes (atomic or multi-valued) that contain a class of objects. Attributes in value schemas can themselves be value schemas, allowing representation of complex objects and encapsulation of information. An object over a value schema is a pair $O = < i, v >$, where i is the object-ID (identity) and v is the object value (properties). A value functional dependency is used at the value schema level to assert that the value of a set of attributes uniquely determines the value of other attributes. The determined attribute can be single-valued or multi-valued. An object schema is a 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 including N itself. S represents subobject schemas that describe the potential sharing between objects and subjects. A class schema is a triple $< N, F, H >$, such that $H \subseteq \mathcal{P}(N)$ and $N \in H$. H defines a superclass schema/subclass schema relationship that induces a partial ordering of class schemas, the inheritance lattice. There is a one-to-one

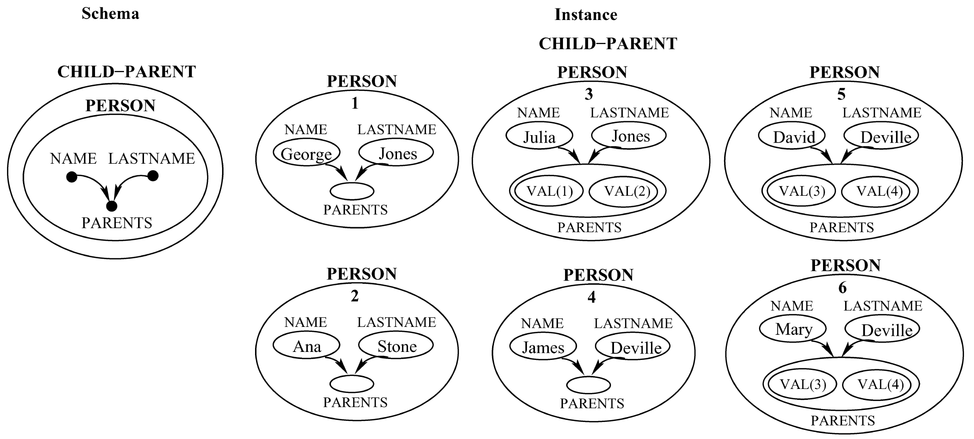


Fig. 8. GROOVY. At the schema level (left), we model an object PERSON as a hypergraph that relates the attributes NAME, LASTNAME and PARENTS. Note the value functional dependency (VDF) NAME,LASTNAME → PARENTS logically represented by the directed hyperedge ({NAME, LASTNAME} {PARENTS}). This VFD asserts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

correspondence between each object schema $< N, F, S >$, and a hypergraph, interpreting N as vertices, F as directed hyperedges, and S as undirected hyperedges. The same approach is applied for representing class schemas and objects in the instance level.

A 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. It introduces a single formalism for both subobject sharing and structural inheritance, avoiding redundancy of data (values of common subobjects are shared by their superobjects). Hypergraphs allow the definition of complex objects (using undirected hyperedges), functional dependencies (using directed hyperedges), object-ID and multiple structural inheritance. Value functional dependences establish semantic integrity constraints for object schemas.

Let us mention that GROOVY influenced the development of the Hypernode model, providing another approach to modeling complex objects. If we compare both models, we can see that hypergraphs can be modeled by hypernodes by encapsulating the contents of each hyperedge within a further hypernode. In contrast, the multilevel nesting provided by hypernodes cannot easily be captured by hypergraphs [Poulovassilis and Levene 1994].

The notion of hypergraphs is also used in other proposals:

—Consens and Mendelzon [1993] 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 [1989] proposes a model for hypertext where nodes represent Web pages and hyperedges represent user state and browsing.

—Watters and Shepherd [1990] 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 performs operations over both hyperedges and nodes representing data.

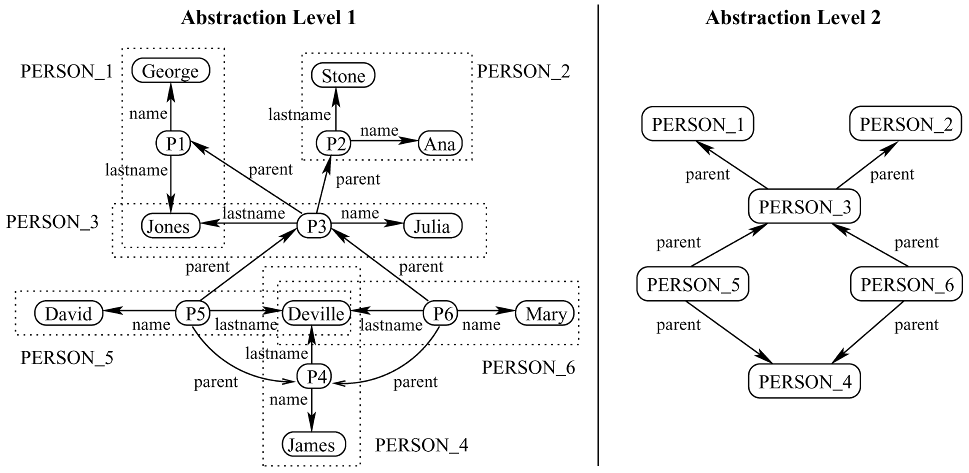


Fig. 9. Simatic-XT. This model does not define an schema. The relations Name-Lastname and Person-Parent are represented in two abstraction levels. In the first level (the most general), the graph contains the relations Name and Lastname to identify people (*P1, . . . , P6*). In the second level we use the abstraction of Person, to compress the attributes Name and Lastname and represent only the relation Parent between people.

A.4. Simatic-XT: A Data Model to Deal with Multi-Scaled Networks

Motivated by modeling of transport networks (train, plane, water, telecommunications), Mainguenaud [1992] proposed a graph (object-oriented) db-model that merges the concepts of graph and object-oriented paradigm, focusing on the graph structure of the data but not on the behavior of entities to be modeled. An example is presented in Figure 9.

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 notions of Master Nodes and Master Edges, to support levels of abstraction of subnetworks and paths, respectively. Each object in the model has 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 represents the set of edges leaving the Master Node or Master Edge.

A sequel paper [Langou and Mainguenaud 1994] 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.

A.5. 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 [Graves et al. 1995a], a genome graph language [Graves et al. 1995b], and query operators for the model [Graves et al. 1994; Graves 1993].

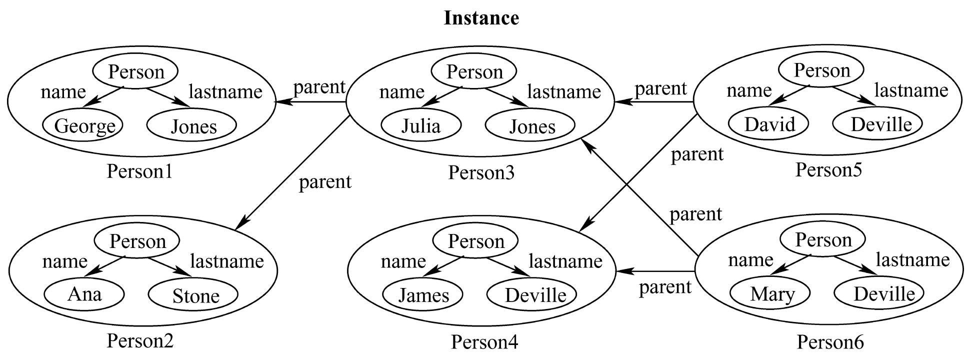


Fig. 10. GGL. Schema and instances are mixed. Packaged graph vertices (Person1, Person2, ...) are used to encapsulate information about the graph defining a Person. Relations among these packages are established using edges labeled with *parent*.

The model is based on binary relationships between objects. This model extends the basic notion of a graph by including vertices that represent edge types that enable specification of relations among relations (higher-order relations), and encapsulated graphs as vertices. An example is presented in Figure 10.

A graph in GGL is basically a collection of: simple vertices, which model simple concepts and can be labeled or unlabeled; symbols, which define nodes without outgoing edges; edges, which connect two vertices and are labeled with a relation name; packaged graph vertices, which represent graphs that are packaged (encapsulated) into vertices; 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.

Two methods are proposed for querying data in this model. The first [Graves et al. 1994], restricts the form of the query graph to be rooted directed acyclic graphs with equality constraints. This strategy is based on following the paths specified by the query-graph and returning the values that correspond to the ends of the paths. The second, is a declarative programming language called WEB [Graves 1993], which defines queries as graphs with the same structures as the model and returns the graphs in the database that match the query-graph.

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

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

A.6. Graph Object-Oriented Data Model (GOOD)

The Graph Object-Oriented Data Model [Gyssens et al. 1990a] is a proposal oriented mainly to develop database end-user graphical interfaces [Gyssens et al. 1990b]. 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 Gemis et al. [1993]. The GOOD schema and instance for the general example are presented in Figure 11.

The model permits only two types of nodes: nonprintable nodes (denoted by squares) and printable nodes (denoted by circles). There is no distinction among atomic,

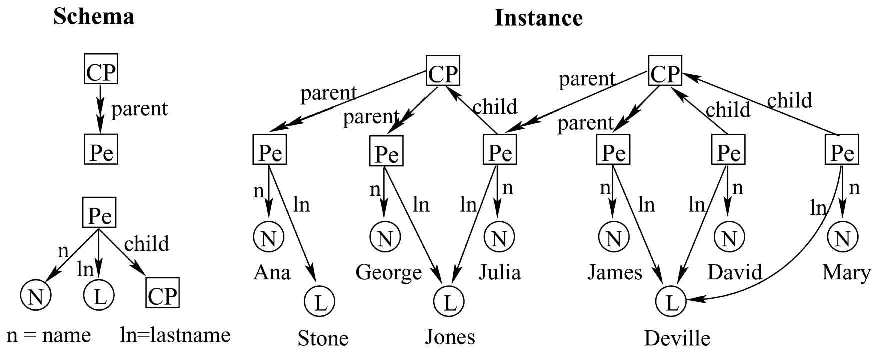


Fig. 11. GOOD. In the schema, we use printable nodes N and L to represent names and lastnames respectively, and nonprintable nodes, Pe(rson) and CP, to represent relations Name-Lastname, and Child-Parent, respectively. A double arrow indicates a nonfunctional relationship, and a simple arrow indicates a functional relationship. The instance is obtained by assigning values to printable nodes and instantiating the CP and PE nodes.

composed, and set objects. There are two types of edges, functional (having a unique value, denoted by $\rightarrow$) and nonfunctional (multi-valued and denoted by $\twoheadrightarrow$). A more detailed version [Gyssens et al. 1991] added node and edges for representing set containment, object composition, generalization, and specialization.

GOOD includes a data transformation language with graphical syntax and semantics. It contains four elementary graph transformation operations: addition and deletion of nodes and edges, plus a fifth operation called abstraction, used to group nodes on the basis of common functional or nonfunctional properties. The specification of all these operations relies on the notion of pattern to describe subgraphs in the database instance. GOOD studies other issues like macros (for more succinct expression of frequent operations), computational-completeness of the query language, and simulation of object-oriented characteristics (inheritance).

This model introduced several useful features. The notion of printable and nonprintable nodes is relevant for the design of graphical interfaces. It has a simple definition of multivalued relations and allows recursive relations. It solves in a balanced way, the data redundancy problem.

A.7. Graph-Oriented Object Manipulation (GMOD)

GMOD [Andries et al. 1992] is a proposal of a general model for object database concepts focused on graph-oriented database user interfaces. Schema and instance are labeled digraphs. An example is presented in Figure 12.

The schema graph has two classes of nodes, abstract objects (rectangular shape), representing class names and, primitive objects (oval shape), representing basic types. Edges represent properties of abstract objects. A 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 as 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.

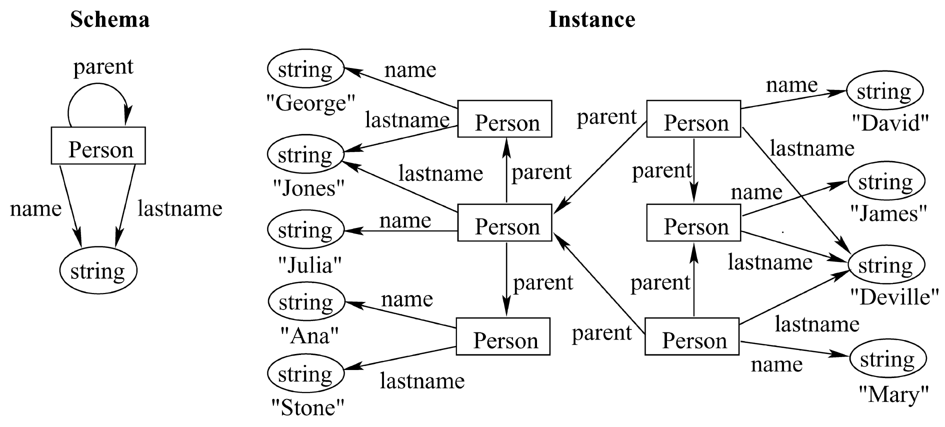


Fig. 12. GMOD. In the schema, nodes represent abstract objects (Person) and labeled edges establish relations with primitive objects (properties name and lastname), and other abstract objects (parent relation). For building an instance, we instantiate the schema for each person by assigning values to oval nodes.

The model allows a simple representation of objects and relations, and 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 in the modeling.

G-Log [Paredaens et al. 1995] is a proposal of a declarative query language for graphs, which works on the data structures defined by GMOD. Queries in G-log are expressed by programs that 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.

A.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 [1993] 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 of PaMaL schema and instance is presented in Figure 13.

The schema defines four types of nodes: ◯ class nodes (upper-case labels), ◯ basic-type nodes (lower-case labels), ⊗ tuple nodes, and ⊛ set nodes. There are four kinds of edges, indicating attribute of a tuple, type of the elements in a set (labeled with ∈), 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-IDs. 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 a special operation that reduces instance graphs. It incorporates loop, procedure, and program constructs, making it a computationally complete language. Among the highlights of the model are the explicit definition of sets and tuples, multiple inheritance, and the use of graphics to describe queries.

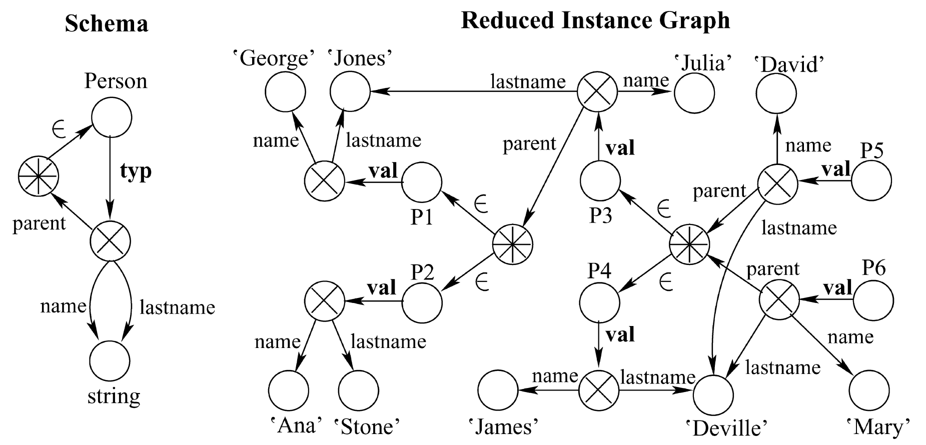


Fig. 13. PaMaL. The example shows all the nodes defined in PaMaL: basic type (string), class (Person), tuple (⊗), set (⊛) nodes for the schema level, and atomic (George, Ana, etc.), instance (*P1, P2*, etc), tuple and set nodes for the instance level. Note the use of edges ∈ to indicate elements in a set, and the edge **typ** to indicate the type of class Person (these edges are changed to **val** in the instance level).

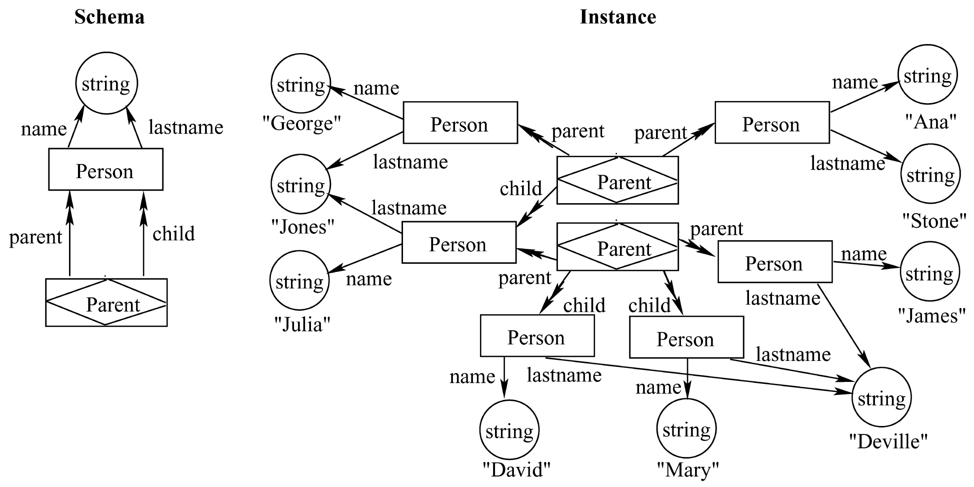


Fig. 14. GOAL: The schema presented in the example shows the use of the object node Person with properties Name and Lastname. The association node Parent and the double headed edges parent and child allow expression of the relation Person-Parent. At the instance level, we assign values to value nodes (string) and create instances for object and association nodes. Note that nodes with same value were merged (e.g. Deville).

A.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 [1993] proposed a graph based db-model for describing schemas 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 14.

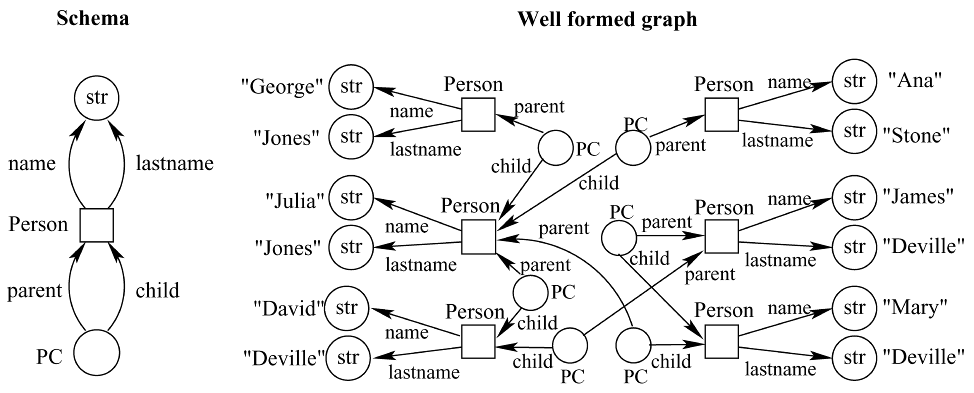


Fig. 15. GDM. In the schema each entity Person (object node represented as a square) has assigned the attributes *name* and *lastname* (basic value nodes represented round and labeled str). We use the composite-value node PC to establish the relationship Parent-Child. Note the redundancy introduced by the node PC. The instance is built by instantiating the schema for each person.

Schema and instances in GOAL are represented as finite directed labeled graphs. A schema enables the definition of three types of nodes: *object nodes*, which represent objects (rectangular nodes); *value nodes*, which represent printable values such as string, integers or Booleans (round nodes) and; *association nodes*, which 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 ensure 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 schema and weak instance.

A.10. Graph Data Model (GDM)

GDM [Hidders 2002, 2001] is a graph-based db-model based on GOOD, which 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 respectively. An example is presented in Figure 15.

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 or more class names indicating their membership to certain

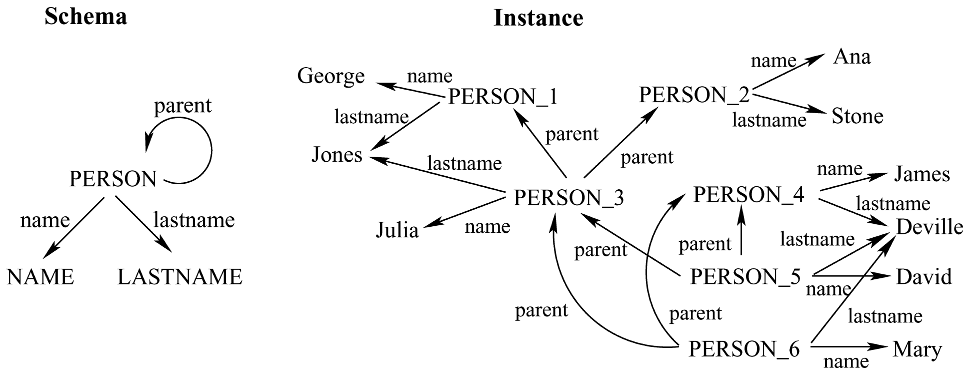


Fig. 16. Gram. At the schema level we use generalized names for definition of entities and relations. At the instance level, we create instance labels (e.g. PERSON_1) to represent entities, and use the edges (defined in the schema) to express relations between data and entities.

classes. If several edges with the same label leave a node, then it is a single set-valued attribute.

GDM introduces the concept of a well-formed graph, defining four constraints: (I-BVA) no edge leaves from a basic-value node; (I-BVT) each basic value node has a real value assigned 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 among 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, which 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 allows instances to exist without a schema, allowing representation of semistructured data. It 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.

A.11. Gram: A Graph Data Model and Query Language

Motivated by hypertext querying, Amann and Scholl [1992] 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 16). A feature of Gram is the use of regular expressions for explicit definition of paths, called walks. An alternating sequence of nodes and edges represent a 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.

A.12. 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.

A.12.1. GraphDB. Güting [1994] 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. There are also data types, object types, and tuple types. There are four types of operators to query GraphDB data: Derive statements, Rewrite operations, Union operator, and 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 [Sheng et al. 1999], a proposal of a graph query language for modeling and querying of multimedia application graphs represented as DAGs. This proposal defines an object-oriented db-model, similar to GraphDB, which defines four types of objects: node, edge, path, and graph. GOQL uses an SQL-like syntax for construction, querying, and manipulation of such objects.

A.12.2. Database Graph Views. A database graph view [Gutiérrez et al. 1994] 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 of the data for the purposes of management and querying, independent of its implementation. In brief the model defines 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.

A.12.3. Object Exchange Model (OEM). OEM [Papakonstantinou et al. 1995] is a semistructured db-model, which 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. An example of OEM is presented in Figure 17.

The data in OEM can be represented as a rooted, directed, connected, 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 it 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 Syntax

```
{ person : &p1 { name : "George" ,
                 lastname : "Jones" }
  person : &p2 { name : "Ana" ,
                 lastname : "Stone" }
  person : &p3 { name : "Julia" ,
                 lastname : "Jones" ,
                 parent : &p1 ,
                 parent : &p2 }
  person : &p4 { name : "James" ,
                 lastname : "Deville" }
  person : &p5 { name = "David",
                 lastname : "Deville" ,
                 parent : &p3 ,
                 parent : &p4 }
  person : &p6 { name = "Mary" ,
                 lastname : "Deville" ,
                 parent : &p3 ,
                 parent : &p4 }  }
```

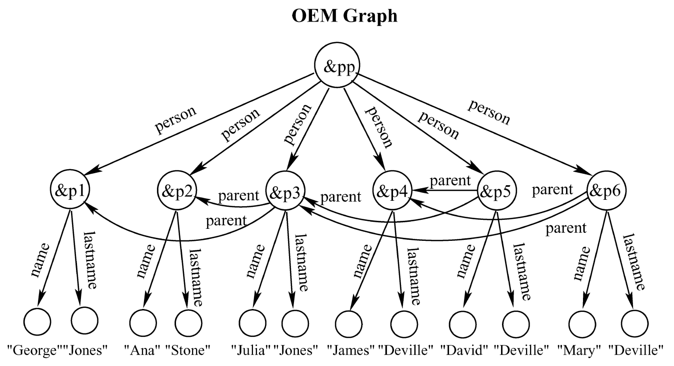


Fig. 17. Object Exchange Model (OEM). Schema and instance are mixed. The data is modeled beginning in a root node &pp, with children *person nodes*, each of them identified by an Object-ID (e.g. &p2). These nodes have children that contain data (*name* and *lastname*) or references to other nodes (*parent*). Referencing permits establishing relations between distinct hierarchical levels. Note the tree structure obtained if one forgets the pointers to OIDs, a characteristic of semistructured data.

A.12.4. eXtended Markup Language (XML). The XML [Bray et al. 1998] model did not originate in the database community. It was introduced as a standard for exchanging information between Web applications. XML allows annotating data with information about its meaning rather than just its presentation [Vianu 2003]. 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 (schema and data are mixed).

Compared to graph data db-models, XML has an ordered-tree-like structure, which is a restricted type of graph. Nevertheless, XML additionally provides a referencing mechanism among elements, which 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 schema graph in a more flexible fashion, using relations among entities. From this point of view, graph db-models use connections to explicitly represent generalization, compositions, hierarchy, classification, and any other types of relations.

A.12.5. Resource Description Framework (RDF). RDF [Klyne and Carroll 2004] is a recommendation of the W3C, originally designed to represent metadata. The broad goal of RDF is to define a mechanism for describing resources, which makes no assumptions about a particular application domain. One of the main advantages (features) of the RDF model is its ability to interconnect resources in an extensible way. Thus, RDF models information with graph-like structure, where basic notions of graph theory like node, edge, path, neighborhood, connectivity, distance, degree, and so on play a central role.

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 subject and the object. A general RDF expression is a set of such triples, which can be intuitively considered as a labeled graph, called an RDF Graph (see example in Figure 18), although formally it is not a graph [Hayes and Gutierrez 2004].

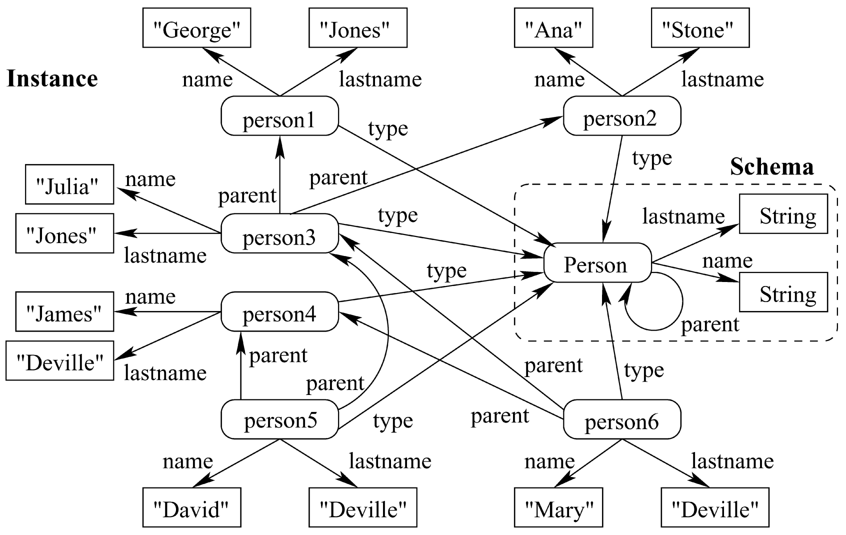


Fig. 18. RDF. Schema and instance are mixed together. In the example, the edges labeled *type* disconnect the instance from the schema. The instance is built by the subgraphs obtained by instantiating the nodes of the schema, and establishing the corresponding parent edges between these subgraphs.

Currently there is research work on storing information expressed in RDF, but none of this work defines a graph db-model or even a db-model. In addition several languages for querying RDF data have been proposed and implemented, which follow the lines of database query languages like SQL, OQL, and XPath [Furche et al. 2006]. A discussion of aspects related to querying RDF from a graph database perspective is presented in Angles and Gutierrez [2005].

SPARQL [Prud'hommeaux and Seaborne 2005] is a proposal of a 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 graph patterns, and query processing consists of the binding of variables to generate pattern solutions (graph pattern matching).

ACKNOWLEDGMENTS

We wish to thank Jocelyn Simmonds for revising the English of the first sections.

REFERENCES

ABITEBOUL, S. 1997. Querying semi-structured data. In *Proceedings of the 6th International Conference on Database Theory (ICDT)*. LNCS, vol. 1186. Springer, 1–18.

ABITEBOUL, S. AND HULL, R. 1984. IFO: A formal semantic database model. In *Proceedings of the 3th Symposium on Principles of Database Systems (PODS)*. ACM Press, 119–132.

ABITEBOUL, S., QUASS, D., MCHUGH, J., WIDOM, J., AND WIENER, J. L. 1997. The Lorel query language for semistructured data. *Int. J. Dig. Libr. 1*, 1, 68–88.

ABITEBOUL, S. AND VIANU, V. 1997. Queries and computation on the Web. In *Proceedings of the 6th International Conference on Database Theory (ICDT)*. LNCS, vol. 1186. Springer, 262–275.

AGRAWAL, R. AND JAGADISH, H. V. 1988. Efficient search in very large databases. In *Proceedings of the 14th International Conference on Very Large Data Bases (VLDB)*. Morgan Kaufmann, 407–418.

AGRAWAL, R. AND JAGADISH, H. V. 1989. Materialization and incremental update of path information. In *Proceedings of the 5th International Conference on Data Engineering (ICDE)*. IEEE Computer Society, 374–383.

AGRAWAL, R. AND JAGADISH, H. V. 1994. Algorithms for searching massive graphs. *IEEE Trans. Knowl. Data Eng. 6*, 2, 225–238.

ALBERT, R. AND BARABÁSI, A.-L. 2002. Statistical mechanics of complex networks. *Rev. Mod. Phy. 74*, 47.

ALECHINA, N., DEMRI, S., AND DE RIJKE, M. 2003. A modal perspective on path constraints. *J. Logic Computation 13*, 6, 939–956.

AMANN, B. AND SCHOLL, M. 1992. Gram: A Graph Data Model and Query Language. In *European Conference on Hypertext Technology (ECHT)*. ACM, 201–211.

ANDRIES, M. AND ENGELS, G. 1993. A hybrid query language for an extended entity-relationship model. Tech. Rep. TR 93-15, Institute of Advanced Computer Science, Universiteit Leiden. May.

ANDRIES, M., GEMIS, M., PAREDAENS, J., THYSSENS, I., AND DEN BUSSCHE, J. V. 1992. Concepts for graph-oriented object manipulation. In *Proceedings of the 3rd International Conference on Extending Database Technology (EDBT)*. LNCS, vol. 580. Springer, 21–38.

ANGLES, R. AND GUTIERREZ, C. 2005. Querying RDF data from a graph database perspective. In *Proceedings of the 2nd European Semantic Web Conference (ESWC)*. Number 3532 in LNCS. 346–360.

AUFAURE-PORTIER, M.-A. AND TRÉPIED, C. 1976. A survey of query languages for geographic information systems. In *Proceedings of the 3rd International Workshop on Interfaces to Databases*. 431–438.

AZMOODEH, M. AND DU, H. 1988. GQL, A graphical query language for semantic databases. In *Proceedings of the 4th International Conference on Scientific and Statistical Database Management (SSDBM)*. LNCS, vol. 339. Springer, 259–277.

BEERI, C. 1988. Data models and languages for databases. In *Proceedings of the 2nd International Conference on Database Theory (ICDT)*. LNCS, vol. 326. Springer, 19–40.

BENKÖ, G., FLAMM, C., AND STADLER, P. F. 2003. A graph-based toy model of chemistry. *J. Chem. Inform. Computer Science (JCISD) 43*, 1 (Jan), 1085–1093.

BERGE, C. 1973. *Graphs and Hypergraphs*. North-Holland, Amsterdam.

BRANDES, U. 2005. *Network Analysis*. Number 3418 in LNCS. Springer-Verlag.

BRAY, T., PAOLI, J., AND SPERBERG-MCQUEEN, C. M. 1998. Extensible Markup Language (XML) 1.0, W3C Recommendation 10, (February). http://www.w3.org/TR/1998/REC-xml-19980210.

BRODER, A., KUMAR, R., MAGHOUL, F., RAGHAVAN, P., RAJAGOPALAN, S., STATA, R., TOMKINS, A., AND WIENER, J. 2000. Graph structure in the Web. In *Proceedings of the 9th International World Wide Web conference on Computer Networks: The International Journal of Computer and Telecommunications Networking*. North-Holland Publishing Co., 309–320.

BUNEMAN, P. 1997. Semistructured data. In *Proceedings of the 16th Symposium on Principles of Database Systems (PODS)*. ACM Press, 117–121.

BUNEMAN, P., DAVIDSON, S., HILLEBRAND, G., AND SUCIU, D. 1996. A query language and optimization techniques for unstructured data. *SIGMOD Record. 25*, 2, 505–516.

BUNEMAN, P., FAN, W., AND WEINSTEIN, S. 1998. Path constraints in semistructured and structured databases. In *Proceedings of the 17th Symposium on Principles of Database Systems (PODS)*. ACM Press, 129–138.

CARDELLI, L., GARDNER, P., AND GHELLI, G. 2002. A spatial logic for querying graphs. In *Proceedings of the 29th International Colloquium on Automata, Languages, and Programming (ICALP)*. LNCS. Springer, 597–610.

CHANDRA, A. K. 1988. Theory of database queries. In *Proceedings of the 7th Symposium on Principles of Database Systems (PODS)*. ACM Press, 1–9.

CHEN, P. P.-S. 1976. The entity-relationship model—toward a unified view of data. *ACM Trans. Database Syst. 1*, 1, 9–36.

CHOMICKI, J. 1994. Temporal query languages: a survey. In *Proceedings of the First International Conference on Temporal Logic (ICTL)*. Springer-Verlag, 506–534.

CODD, E. F. 1970. A relational model of data for large shared data banks. *Commun. ACM 13*, 6, 377–387.

CODD, E. F. 1980. Data models in database management. In *Proceedings of the 1980 Workshop on Data abstraction, Databases, and Conceptual Modeling*. ACM Press, 112–114.

CODD, E. F. 1983. A relational model of data for large shared data banks. *Commun. ACM 26*, 1, 64–69.

CONKLIN, J. 1987. Hypertext: An introduction and survey. *IEEE Comput. 20*, 9, 17–41.

CONSENS, M. AND MENDELZON, A. 1993. Hy+: A hygraph-based query and visualization system. *SIGMOD Record 22*, 2, 511–516.

CONSENS, M. P. AND MENDELZON, A. O. 1989. Expressing structural hypertext queries in graphlog. In *Proceedings of the 2th Conference on Hypertext*. ACM Press, 269–292.

Cruz, I. F., Mendelzon, A. O., and Wood, P. T. 1987. A graphical query language supporting recursion. In *Proceedings of the Association for Computing Machinery Special Interest Group on Management of Data*. ACM Press, 323–330.

Cruz, I. F., Mendelzon, A. O., and Wood, P. T. 1989. G+: recursive queries without recursion. In *Proceedings of the 2th International Conference on Expert Database Systems (EDS)*. Addison-Wesley, 645–666.

Date, C. J. 1981. Referential integrity. In *Proceedings of the 7th International Conference on Very Large Data Bases (VLDB)*. IEEE Computer Society, 2–12.

de S. Price, D. J. 1965. Networks of scientific papers. *Science 149*, 510–515.

Deng, Y. and Chang, S.-K. 1990. A G-Net model for knowledge representation and reasoning. *IEEE Trans. Knowl. Data Eng. 2*, 3 (Dec), 295–310.

Deville, Y., Gilbert, D., van Helden, J., and Wodak, S. J. 2003. An overview of data models for the analysis of biochemical pathways. In *Proceedings of the First International Workshop on Computational Methods in Systems Biology*. Springer-Verlag, 174.

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

Fernández, M., Florescu, D., Kang, J., Levy, A., and Suciu, D. 1998. Catching the boat with strudel: experiences with a Web-site management system. In *Proceedings of the 1998 ACM SIGMOD International Conference on Management of Data*. ACM Press, 414–425.

Flesca, S. and Greco, S. 1999. Partially ordered regular languages for graph queries. In *Proceedings of the 26th International Colloquium on Automata, Languages and Programming (ICALP)*. LNCS, vol. 1644. Springer, 321–330.

Flesca, S. and Greco, S. 2000. Querying graph databases. In *Proceedings of the 7th International Conference on Extending Database Technology—Advances in Database Technology (EDBT)*. LNCS, vol. 1777. Springer, 510–524.

Florescu, D., Levy, A., and .Mendelzon, A. O. 1998. Database techniques for the World-Wide Web: A survey. *SIGMOD Record 27*, 3, 59–74.

Fry, J. P. and Sibley, E. H. 1976. Evolution of data-base management systems. *ACM Comput. Surv. 8*, 1.

Furche, T., Linse, B., Bry, F., Plexousakis, D., and Gottlob, G. 2006. RDF querying: language constructs and evaluation methods compared. In *Reasoning Web*. Number 4126 in LNCS. 1–52.

Gemis, M. and Paredaens, J. 1993. An object-oriented pattern matching language. In *Proceedings of the First JSSST International Symposium on Object Technologies for Advanced Software*. Springer-Verlag, 339–355.

Gemis, M., Paredaens, J., Thyssens, I., and den Bussche, J. V. 1993. GOOD: A graph-oriented object database system. In *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data*. ACM Press, 505–510.

Giugno, R. and Shasha, D. 2002. GraphGrep: A fast and universal method for querying graphs. In *Proceedings of the IEEE International Conference in Pattern recognition (ICPR)*.

Graves, M. Graph data models for genomics. http://www.xweave.com/people/in graaves/pubs.

Graves, M. 1993. Theories and tools for designing application-specific knowledge base data models. Ph.D. dissertation, University of Michigan.

Graves, M., Bergeman, E. R., and Lawrence, C. B. 1994. Querying a genome database using graphs. In *Proceedings of the 3th International Conference on Bioinformatics and Genome Research*.

Graves, M., Bergeman, E. R., and Lawrence, C. B. 1995a. A graph-theoretic data model for genome mapping databases. In *Proceedings of the 28th Hawaii International Conference on System Sciences (HICSS)*. IEEE Computer Society, 32.

Graves, M., Bergeman, E. R., and Lawrence, C. B. 1995b. Graph database systems for genomics. *IEEE Eng. Medicine Biol. Special issue on Managing Data for the Human Genome Project 11*, 6.

Griffith, R. L. 1982. Three principles of representation for semantic networks. *ACM Trans. Database Syst. 7*, 3, 417–442.

Guha, R.V., Lassila, O., Miller, E., and Brickley, D. 1998. Enabling inferencing. *The Query Languages Workshop (QL)*.

Gutiérrez, A., Pucheral, P., Steffen, H., and Thévenin, J.-M. 1994. Database graph views: A practical model to manage persistent graphs. In *Proceedings of the 20th International Conference on Very Large Data Bases (VLDB)*. Morgan Kaufmann, 391–402.

Güting, R. H. 1994. GraphDB: modeling and querying graphs in databases. In *Proceedings of the 20th International Conference on Very Large Data Bases (VLDB)*. Morgan Kaufmann, 297–308.

GYSSENS, M., PAREDAENS, J., DEN BUSSCHE, J. V., AND GUCHT, D. V. 1990a. A graph-oriented object database model. In *Proceedings of the 9th Symposium on Principles of Database Systems (PODS)*. ACM Press, 417–424.

GYSSENS, M., PAREDAENS, J., DEN BUSSCHE, J. V., AND GUCHT, D. V. 1991. A graph-oriented object database model. Tech. Rep. 91-27, University of Antwerp (UIA), Belgium. (March).

GYSSENS, M., PAREDAENS, J., AND GUCHT, D. V. 1990b. A graph-oriented object model for database end-user interfaces. In *Proceedings of the 1990 ACM SIGMOD International Conference on Management of Data*. ACM Press, 24–33.

HAMMER, J. AND SCHNEIDER, M. 2004. The GenAlg project: developing a new integrating data model, language, and tool for managing and querying genomic information. *SIGMOD Record 33*, 2, 45–50.

HAMMER, M. AND MCLEOD, D. 1978. The semantic data model: a modelling mechanism for data base applications. In *Proceedings of the 1978 ACM SIGMOD International Conference on Management of Data*. ACM, 26–36.

HANNEMAN, R. A. 2001. Introduction to social network methods. Tech. Rep., Department of Sociology, University of California, Riverside.

HAYES, J. AND GUTIERREZ, C. 2004. Bipartite graphs as intermediate model for RDF. In *Proceedings of the 3th International Semantic Web Conference (ISWC)*. Number 3298 in LNCS. Springer-Verlag, 47–61.

HEUER, A. AND SCHOLL, M. H. 1991. Principles of object-oriented query languages. In *Datenbanksysteme in Büro, Technik und Wissenschaft (BTW)*. Informatik-Fachberichte, vol. 270. Springer, 178–197.

HIDDERS, J. 2001. A graph-based update language for object-oriented data models. Ph.D. dissertation, Technische Universiteit Eindhoven.

HIDDERS, J. 2002. Typing graph-manipulation operations. In *Proceedings of the 9th International Conference on Database Theory (ICDT)*. Springer-Verlag, 394–409.

HIDDERS, J. AND PAREDAENS, J. 1993. GOAL, A graph-based object and association language. *Advances in Database Systems: Implementations and Applications, CISM*, 247–265.

HULL, R. AND KING, R. 1987. Semantic database modeling: Survey, applications, and research issues. *ACM Comput. Surv. 19*, 3, 201–260.

ISO. 1999. International Standard ISO/IEC 13250 Topic Maps.

JAGADISH, H. V. AND OLKEN, F. 2003. Data management for the biosciences: report of the NLM Workshop on Data Management for Molecular and Cell Biology. Tech. Rep. LBNL-52767, National Library of Medicine.

KERSCHBERG, L., KLUG, A. C., AND TSICHRITZIS, D. 1976. A taxonomy of data models. In *Proceedings of Systems for Large Data Bases (VLDB)*. North Holland and IFIP, 43–64.

KIESEL, N., SCHURR, A., AND WESTFECHTEL, B. 1996. GRAS: A graph-oriented software engineering database system. In *IPSEN Book*. 397–425.

KIFER, M., KIM, W., AND SAGIV, Y. 1992. Querying object-oriented databases. In *Proceedings of the 1992 ACM SIGMOD International Conference on Management of Data*. ACM Press, 393–402.

KIM, W. 1990. Object-oriented databases: definition and research directions. *IEEE Trans. Knowl. Data Eng. 2*, 3, 327–341.

KLYNE, G. AND CARROLL, J. 2004. Resource description framework (RDF) concepts and abstract syntax. http://www.w3.org/TR/2004/REC-rdf-concepts-20040210/.

KUMAR, R., RAGHAVAN, P., RAJAGOPALAN, S., SIVAKUMAR, D., TOMKINS, A., AND UPFAL, E. 2000. The Web as a graph. In *Proceedings of the 19th Symposium on Principles of Database Systems (PODS)*. ACM Press, 1–10.

KUNII, H. S. 1987. DBMS with graph data model for knowledge handling. In *Proceedings of the 1987 Fall Joint Computer Conference on Exploring technology: Today and Tomorrow*. IEEE Computer Society Press, 138–142.

KUPER, G. M. AND VARDI, M. Y. 1984. A new approach to database logic. In *Proceedings of the 3th Symposium on Principles of Database Systems (PODS)*. ACM Press, 86–96.

KUPER, G. M. AND VARDI, M. Y. 1993. The Logical Data Model. *ACM Trans. Database Syst. 18*, 3, 379–413.

LANGOU, B. AND MAINGUENAUD, M. 1994. Graph data model operations for network facilities in a geographical information system. In *Proceedings of the 6th International Symposium on Spatial Data Handling*. Vol. 2. 1002–1019.

LÉCLUSE, C., RICHARD, P., AND VÉLEZ, F. 1988. O2, an object-oriented data model. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*. ACM Press, 424–433.

LEVENE, M. AND LOIZOU, G. 1995. A graph-based data model and its ramifications. *IEEE Trans. Knowl. Data Eng. 7*, 5, 809–823.

Levene, M. and Poulovassilis, A. 1990. The Hypernode model and its associated query language. In *Proceedings of the 5th Jerusalem Conference on Information technology*. IEEE Computer Society Press, 520–530.

Levene, M. and Poulovassilis, A. 1991. An object-oriented data model formalised through hypergraphs. *Data Knowl. Eng. 6*, 3, 205–224.

Mainguenaud, M. 1992. Simatic XT: A data model to deal with multi-scaled networks. *Comput. Environ. Urban Syst. 16*, 281–288.

Mainguenaud, M. 1995. Modelling the network component of geographical information systems. *Int. J. Geog. Inform. Syst. 9*, 6, 575–593.

Mannino, M. V. and Shapiro, L. D. 1990. Extensions to query languages for graph traversal problems. *IEEE Trans. Knowl. Data Eng. 2*, 3, 353–363.

McGee, W. C. 1976. On user criteria for data model evaluation. *ACM Trans. Database Syst. 1*, 4, 370–387.

McGuinness, D. L. and van Harmelen, F. 2004. OWL Web ontology language overview, W3C recommendation 10 (February). http://www.w3.org/TR/2004/REC-owl-features-20040210/.

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

Mendelzon, A. O. and Wood, P. T. 1989. Finding regular simple paths in graph databases. In *Proceedings of the 15th International Conference on Very Large Data Bases (VLDB)*. Morgan Kaufmann Publishers Inc., 185–193.

Navathe, S. B. 1992. Evolution of data modeling for databases. *Communications of the ACM 35*, 9, 112–123.

Nejdl, W., Siberski, W., and Sintek, M. 2003. Design issues and challenges for RDF- and schema-based peer-to-peer systems. *SIGMOD Record 32*, 3, 41–46.

Newman, M. E. J. 2003. The structure and function of complex networks. *SIAM Rev. 45*, 2, 167–256.

Olken, F. 2003. Tutorial on graph data management for biology. *IEEE Computer Society Bioinformatics Conference (CSB)*.

Papakonstantinou, Y., Garcia-Molina, H., and Widom, J. 1995. Object exchange across heterogeneous information sources. In *Proceedings of the 11th International Conference on Data Engineering (ICDE)*. IEEE Computer Society, 251–260.

Paredaens, J. and Kuijpers, B. 1998. Data models and query languages for spatial databases. *Data & Knowledge Engineering (DKE) 25*, 1–2, 29–53.

Paredaens, J., Peelman, P., and Tanca, L. 1995. G-Log: A graph-based query language. *IEEE Trans. Knowl. Data Eng. 7*, 3, 436–453.

Peckham, J. and Maryanski, F. J. 1988. Semantic data models. *ACM Comput. Surv. 20*, 3, 153–189.

Pepper, S. and Moore, G. 2001. XML topic maps (XTM) 1.0—TopicMaps.Org Specification. http://www.topicmaps.org/xtm/1.0/xtm1-20010806.html.

Poulovassilis, A. and Hild, S. G. 2001. Hyperlog: A graph-based system for database browsing, querying, and update. *IEEE Trans. Knowl. Data Eng. 13*, 2, 316–333.

Poulovassilis, A. and Levene, M. 1994. A nested-graph model for the representation and manipulation of complex objects. *ACM Trans. Inform. Syst. 12*, 1, 35–68.

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

Ramakrishnan, R. and Ullman, J. D. 1993. A survey of research on deductive database systems. *J. Logic Prog. 23*, 2, 125–149.

Roussopoulos, N. and Mylopoulos, J. 1975. Using semantic networks for database management. In *Proceedings of the International Conference on Very Large Data Bases (VLDB)*. ACM, 144–172.

Samet, H. and Aref, W. G. 1995. Spatial data models and query processing. In *Modern Database Systems*. 338–360.

Schewe, K.-D., Thalheim, B., Schmidt, J. W., and Wetzel, I. 1993. Integrity enforcement in object-oriented databases. In *Proceedings of the 4th International Workshop on Foundations of Models and Languages for Data and Objects*.

Shasha, D., Wang, J. T. L., and Giugno, R. 2002. Algorithmics and applications of tree and graph searching. In *Proceedings of the 21th Symposium on Principles of Database Systems (PODS)*. ACM Press, 39–52.

Shekhar, S., Coyle, M., Goyal, B., Liu, D.-R., and Sarkar, S. 1997. Data models in geographic information systems. *Commun. ACM 40*, 4, 103–111.

Sheng, L., Ozsoyoglu, Z. M., and Ozsoyoglu, G. 1999. A graph query language and its query processing. In *Proceedings of the 15th International Conference on Data Engineering (ICDE)*. IEEE Computer Society, 572–581.

SHETH, A., ALEMAN-MEZA, B., ARPINAR, I. B., HALASCHEK-WIENER, C., RAMAKRISHNAN, C., BERTRAM, C., WARKE, Y., AVANT, D., ARPINAR, F. S., ANYANWU, K., AND KOCHUT, K. 2005. Semantic association identification and knowledge discovery for national security applications. *J. Database Manag. 16*, 1 (Jan-March), 33–53.

SHIPMAN, D. W. 1981. The functional data model and the data language DAPLEX. *ACM Trans. Database Syst. 6*, 1, 140–173.

SILBERSCHATZ, A., KORTH, H. F., AND SUDARSHAN, S. 1996. Data models. *ACM Comput. Surv. 28*, 1, 105–108.

SOWA, J. F. 1976. Conceptual graphs for a database interface. *IBM J. Res. Devel. 20*, 4, 336–357.

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

SOWA, J. F. 1991. *Principles of Semantic Networks: Explorations in the Representation of Knowledge*. Morgan Kaufmann Publishers.

STEIN, L. D. AND TIERRY-MIEG, J. 1999. AceDB: A genome database management system. *Comput. Sci. Eng. 1*, 3, 44–52.

TANSEL, A., CLIFFORD, J., GADIA, S., JAJODIA, S., SEGEV, A., AND SNODGRASS, R. T., Eds. 1993. *Temporal Databases: Theory, Design, and Implementation*. Benjamin-Cummings.

TAYLOR, R. W. AND FRANK, R. L. 1976. CODASYL data-base management systems. *ACM Comput. Surv. 8*, 1, 67–103.

THALHEIM, B. 1991. *Dependencies in Relational Databases*. Leipzig, Teubner Verlag.

THALHEIM, B. 1996. An overview on semantical constraints for database models. In *Proceedings of the 6th International Conference Intellectual Systems and Computer Science*.

TOMPA, F. W. 1989. A data model for flexible hypertext database systems. *ACM Trans. Inform. Syst. 7*, 1, 85–100.

TSICHRITZIS, D. C. AND LOCHOVSKY, F. H. 1976. Hierarchical data-base management: A survey. *ACM Comput. Surv. 8*, 1, 105–123.

TSVETOVAT, M., DIESNER, J., AND CARLEY, K. 2004. NetIntel: A database for manipulation of rich social network data. Tech. Rep. CMU-ISRI-04-135, Carnegie Mellon University, School of Computer Science, Institute for Software Research International.

TUV, E., POULOVASSILIS, A., AND LEVENE, M. 1992. A storage manager for the hypernode model. In *Proceedings of the 10th British National Conference on Databases*. Number 618 in LNCS. Springer-Verlag, 59–77.

VARDI, M. Y. 1982. The complexity of relational query languages (extended abstract). In *Proceedings of the 14th ACM Symposium on Theory of Computing (STOC)*. ACM Press, 137–146.

VASSILIADIS, P. AND SELLIS, T. 1999. A survey of logical models for OLAP Databases. *SIGMOD Record 28*, 4, 64–69.

VIANU, V. 2003. A Web odyssey: From Codd to XML. *SIGMOD Record 32*, 2, 68–77.

WATTERS, C. AND SHEPHERD, M. A. 1990. A transient hypergraph-based model for data access. *ACM Trans. Inform. Syst. 8*, 2, 77–102.

WEDDELL, G. E. 1992. Reasoning about functional dependencies generalized for semantic data models. *ACM Trans. Database Syst. 17*, 1, 32–64.

WELLMAN, B., SALAFF, J., DIMITROVA, D., GARTON, L., GULIA, M., AND HAYTHORNTHWAITE, C. 1996. Computer networks as social networks: collaborative work,telework, and virtual community. *Ann. Rev. Sociol. 22*, 213–238.

YANNAKAKIS, M. 1990. Graph-theoretic methods in database theory. In *Proceedings of the 9th Symposium on Principles of Database Systems (PODS)*. ACM Press, 230–242.

ZICARI, R. 1991. A framework for schema updates in an object-oriented database system. In *Proceedings of the 7th International Conference on Data Engineering (ICDE)*. IEEE Computer Society, 2–13.