In Pullan & al. (2000) we presented the Prometheus Taxonomic Model. Here we discuss the issues that arose while implementing this model in relation to the main database models and query languages currently available. The Prometheus model and its query language were constructed through examination of the processes by which taxonomists build classifications; that is, the conceptualisation of taxonomic entities and the definition of their inter-relationships. The following important principles underpinning this work are:

1. Specimens are the fundamental unit of taxonomy.
2. The processes of classification and naming are distinct.
3. Many overlapping, possibly contradictory taxonomic classifications co-exist.

We argued (Pullan & al., 2000) that most existing database systems that support taxonomic work do not make a clear distinction between the process of classification and that of naming. They use names to represent taxa and do not allow the construction of taxon concepts in a manner that is independent of nomenclature. This prevents them from simultaneously representing multiple classifications.

In Prometheus we overcame these problems by elaborating two structures: the Circumscribed Taxon (CT) and the Nomenclatural Taxon (NT). CTs are constructed by grouping specimens in order to form representations of taxa at species rank and below. CTs are represented by taxa and do not allow the construction of taxon concepts in a manner that is independent of nomenclature. This prevents them from simultaneously representing multiple classifications.

In Prometheus we overcame these problems by elaborating two structures: the Circumscribed Taxon (CT) and the Nomenclatural Taxon (NT). CTs are constructed by grouping specimens in order to form representations of taxa at species rank and below. CTs are represented by taxa and do not allow the construction of taxon concepts in a manner that is independent of nomenclature. This prevents them from simultaneously representing multiple classifications.

Types of databases commonly used for handling taxonomic data are compared. It is shown that none of the existing types of databases fully support the requirements of taxonomic data. The results of this comparison provide a rationale for the design and implementation of a new database system based on an Extended Object-Oriented model. The new model, the Prometheus Object-Oriented Model (POOM) and its query language Prometheus Object-Oriented Language (POOL), are discussed in relation to the requirements of taxonomic data and the Prometheus Taxonomic Data Model. POOM provides the ability to represent the semantics of relationships between data. This allows us to build the type of graph structures that are found in taxonomic hierarchies. In addition, the system can query and manipulate these graphs (POOL). These features have been used to improve the original Prometheus Taxonomic Data Model. These improvements and the advantages they offer are described and discussed.

**KEYWORDS:** databases, graphs, Prometheus Taxonomic Model, taxonomy.
laid out in *ICBN*.

In this paper we consider in more detail the structure of taxonomic data and examine the degree to which the main existing database models and query languages support the storage and manipulation of taxonomic data relative to the features we have highlighted. We finish by describing a new object-oriented database model (*POOM*), and explain how this has allowed us to implement the taxonomic model as described. Please note that in order to avoid confusion over some of the terminology we have included a brief glossary of terms as an Appendix.

**IMPORTANT FEATURES OF TAXONOMIC DATA**

The process of modelling taxonomic data and working practices of taxonomists has given us a number of insights into the data themselves. These insights have implications for the choice of database model and query language appropriate for the storing of taxonomic data and are listed below.

1. Classifications are built through the aggregation of taxa (see Fig. 1). The structure of classification data is therefore hierarchic. To represent multiple classifications there must be multiple independent classification hierarchies. We can also find hierarchies within nomenclatural data, however the strict interpretation of nomenclatural hierarchies differs from classification hierarchies. In our opinion nomenclatural hierarchies convey almost no classification information. The only classification information that can be drawn from a binomial is that the genus contains the generitype and the type of the named species. On the basis of name alone, no classification relationship can be assumed with other species described in the genus. Taxonomic data are, therefore, constituted of different kinds of hierarchies that coexist and serve multiple purposes.

**Implication:** Database management systems (DBMS) used to store and manipulate taxonomic data must be able to provide a generic and flexible mechanism for supporting the definition, manipulation and querying of hierarchies.

2. The ranks that are used in the process of building classification hierarchies are ordered. For plants, this order is defined in the *ICBN*.

**Implication:** Classification hierarchies are directed hierarchies, and this directionality must be supported.

3. The *ICBN* specifies that not all ranks are mandatory, and it is possible to skip ranks when hierarchies are built.

**Implication:** The support for hierarchies must be flexible enough to allow hierarchies to be constructed

---

**Fig. 1.** An illustration of how specimen lists can be used to construct representations of taxa. This figure shows species as the lowest rank, although any taxonomic rank from species or below could be used as a starting point.
using different sequences of ranks. It must also allow for the comparison of hierarchies based on the rank of constituent nodes rather than simply using the depth of nodes in the hierarchy.

4. It is important to recognise that classifications are matters of opinion and are therefore subjective. A published taxon is a reflection of the opinion of the taxonomist at the time of publication.

**Implication:** The database representation of a published taxon should never be extended to include information beyond that which was included in the original publication. For example, specimens given the same name as, but not included in, a published taxon concept must not be added to it. The data model used to represent the taxonomic schema must be able to clearly define classifications and allow mechanisms to be implemented that ensure that published classifications cannot be modified.

5. Having recognised that published opinions are immutable, it is also important to recognise that when taxonomists undertake a revision of a group, they usually start from existing classifications. They then modify their definitions and relationships to other taxa to reflect their current opinion.

**Implication:** It is important that the DBMS used to implement the model is capable of copying existing classifications so that the process of revision as described above is supported.

6. A plant name is defined as a unique combination of publication, author, type, name elements and rank. A name is only fully represented as the set of these constituents. In modelling terms, taxon names are complex entities with intra-object (aggregation) relationships between the independent constituent entities of the whole. Inter-object (association) relationships link the complex entity as a whole to other entities in the database.

**Implication:** Support must be provided for the representation and manipulation of complex entities.

7. Specimen-based taxon concepts can be automatically named by examination of the types within a taxon definition and subsequent application of the appropriate nomenclatural rules. However, when taxa are stored as aggregations of other taxa (e.g., a genus defined as the aggregation of a set of species), discovery of all the type information will require the recursive querying of its classification hierarchy.

It can also be argued that circumscriptions based on specimen lists are the only information that can be compared across classifications (Pullan & al., 2000).

Comparison of classifications will also require the recursive querying of the hierarchic classification data so that appropriate specimen lists can be extracted.

**Implication:** Recursive querying of the hierarchical data structures must be supported.

8. Specimens become type specimens when they are used in the publication of a new name or are later designated as such. The use of a specimen as a type does not alter the fundamental nature of the specimen, it simply changes its behaviour relative to the rules. In this case, it means that the specimen must be considered during the process of naming a taxon.

**Implication:** The database model must support the acquisition of roles that can alter the behaviour of the entities or of the DBMS towards these entities.

9. Taxa are named according to the rules of the relevant nomenclatural code. These rules change over time.

**Implication:** It is important that a taxonomic DBMS offers ways to define rules independent of the data. This independence provides the basis for a more flexible rule-based system that is able to support new rules when they are promulgated and is able to support multiple nomenclatural codes simultaneously. In addition, independence of the rules from the data means that when the rules change, the data are not affected, and the user application does not need rewriting.

In summary, we have seen that taxonomic data place the following requirements on an underlying database model and query language:

1. The generic ability to create directed hierarchies of entities that may overlap and yet can be readily and clearly distinguished.
2. Support for composite entities, e.g., names that are defined as a set of other entities.
3. The ability to acquire roles or change behaviour according to context, e.g., specimens that become type specimens.
4. Support for constraints and/or deductive rules, e.g., for implementing nomenclatural codes and deriving names.
5. Support for manipulation of hierarchies and recursive behaviour, e.g., for deriving names or comparing classification hierarchies.

In the following sections we study the four main types of databases (relational, graph, semi-structured, and object-oriented), and examine the degree of support they offer for the features of taxonomic data described above. When considering the degree of support for
storing hierarchical information (requirement 1), we have restricted our assessment to the ability of the model to store and manipulate graph structures. This is because one of the most powerful ways of representing hierarchical data structures is through the use of graphs (finite sets of nodes linked together by edges).

### RELATIONAL DATABASES

Most taxonomic databases are implemented using relational databases, e.g., *ALICE* (Allkin & Winfield, 1989), *BRAHMS* (Filer, 1994), *PANDORA* (Pankhurst, 1993), *TROPICOS* (Crosby & Magill, 1988). Indeed, in a recent survey of over 448 databases with a taxonomic content, over 300 of the applications were designed and built using relational systems (Berendsohn, 2000). However, when considered against the requirements listed above, the following limitations of relational systems are apparent.

**Support for the definition of graphs.** — Relational databases were not designed to support the representation of graphs, which are essentially recursive (or nested) structures. However, graphs can be represented in a relational database as a set of tables that record the existence of edges between the nodes. This flattened representation for graphs might be acceptable in some circumstances, but the semantics of graph representation and querying are not built into the database model. This function would need to be re-implemented for each database application.

**Support for composite entities.** — Relational databases were not designed to handle complex data: they were designed to efficiently handle simple operations on simple data. This is not to say that data models built using the relational model cannot be complex. Rather that the elements within the model must be simple as required by the principle of atomization, and are constrained to being one of the basic data types supported by the relational model. The relational model therefore lacks the power essential for composite entities (Raguenaud & al., 2000). The complexity of taxonomic data is such that relational databases require the implementation of composite entity support in the user application. This implies a higher level of development, maintenance, and evolution of such applications when compared with applications built using the object-oriented model.

**Support for role acquisition.** — Relational databases do not support mechanisms that would allow the acquisition of roles *per se*. However, it would be possible to declare multiple views on a relation (table) that would display different attributes according to the role intended for each view. For example, a view storing all specimens that are referred to by another table as a type could describe a type specimen view. However, this technique would not easily support the changes in behaviour that must occur when an entity changes role (specimens would have to migrate from one view to the other). Because the various views would not be mutually exclusive, it would be left to the programmer to manage the various behaviours and spot when a particular entity has switched role (e.g., if a different treatment is suitable for specimens that are types and specimens that are not).

**Support constraints and/or deductive rules.** — Relational databases support a powerful and well-adapted constraint mechanism (e.g., SQL92; Connan & Otten, 1992). However, this mechanism is tailored to the structures that are definable in the database. It therefore suffers from the same limitations as the query language, which is simple manipulation of simple flat structures.

**Support for the manipulation/querying of graphs.** — Even though implementing graphs as described above is possible, it complicates the writing of queries. In addition, recursive queries would require the development of applications designed to manipulate the graphs “intelligently”, which would increase the cost of developing applications and their maintenance.

**Support for recursive behaviour.** — Although some rare exceptions exist [e.g., Oracle (Feuerstein, 1995) that support a form of recursion in PL/SQL, SQL* (Koymen & Cai, 1993)], relational databases are not able to natively handle recursive behaviour. Embedding the necessary specific code in the application or in stored procedures could provide support for this; this would, however, also increase development and maintenance overheads.

### GRAPH DATABASES

Graph databases, e.g., *HICLAS* (Zhong & al., 1996, 1999), have the advantage of offering an inherent ability to represent hierarchies or classifications. The *Prometheus Taxonomic Model* (Pullan & al., 2000) can be easily mapped to a graph database, and initially was implemented using the graph database *Prometheus/G* (Raguenaud & al., 1999). This approach provided us with valuable insights into the workings of graph models. However, in order to overcome the limitations of the graph database model, it would have been necessary to import many of the features that exist in other more abstract models (e.g., object-oriented database models). This was considered an impracticable and redundant approach. The limitations of the graph database model in the context of taxonomy are discussed below.

**Support for the definition of graphs.** — As one would expect, graph databases inherently support the
definition of graph structures. A taxonomic database could therefore use a graph-based storage mechanism to represent classifications.

**Support for composite entities.** — With the exception of Hyperlog (Poulouassilis & Levene, 1994) graph databases lack semantics and abstraction. They usually only capture the existence of an edge between two nodes, not the semantics associated with the relationship. Therefore, it is not possible to distinguish between intra- and extra-relationships, and the circumscription of a composite entity cannot be defined.

**Support for role acquisition.** — Because graph databases generally do not support the ability to describe and manipulate complex entities, they do not provide mechanisms to acquire new roles.

**Support constraints and/or deductive rules.** — With a few exceptions (Telos, Mylopoulos & al., 1990; ConceptBase, Jarke & al., 1995; Progres, Münch & al., 1998), most graph databases do not support constraints. This limits both the ability to support integrity checking and the automatic derivation of new information from the existing data.

**Support for the manipulation/querying of graphs.** — Although intuitively manageable via graphical means, querying in a graph database is made harder by the need to assemble large numbers of related entities in order to express a query. It is also difficult to handle and describe the data returned. For example, even when simple queries are graphically generated, Progres often returns information in text notation.

**Support for recursive behaviour.** — With a few exceptions (e.g., Buneman & al., 1995; Prometheus/G), graph databases do not support recursive querying.

---

**SEMI-STRUCTURED DATABASES**

Semi-structured databases are related to graph databases, and so have the same limitations (such as the in-ability to support the semantics of relationships). In addition, as they are based on a model without schemas, they do not offer the basis to implement an integrity constraint mechanism (the schema is not known a priori).

---

**OBJECT-ORIENTED DATABASES**

Object-oriented databases have been designed to support complex application domains where relational models were inadequate. Object-oriented design supports inheritance, encapsulation, and behaviour. It therefore provides better tools to describe complex data. Nevertheless, object-oriented models still lack certain features that are of importance to taxonomic data.

**Support for the definition of graphs.** — Object-oriented databases can support the definition of simple object graphs. Indeed, objects and their references form a graph. However, only the simplest graphs can be represented, in which edges show only the existence of a link between two nodes and can carry no additional information. For example, weighted, directed graphs are impossible.

**Support for composite entities.** — Object-oriented databases provide the means to describe composite entities. This is achieved through the use of type casting and encapsulation. However, the description of these entities is hampered by the limited ability of such models to describe the semantics of complex entities. For example, in most object-oriented databases it is not possible to distinguish the relationships between the component elements of a composite entity, and the relationship between the whole entity and other entities. A notable exception is that described in Bertino & Guerrini (1998), which uses integrity constraints to support a limited amount of this kind of differentiation.

**Support for role acquisition.** — Role acquisition is partially supported by object-oriented principles. Objects can behave differently depending on their internal state or as a consequence of the state of another object. More advanced role acquisition mechanisms exist (Gottlob & al., 1996; Albano & al., 1991; Kuno & al., 1995). These mechanisms allow objects to behave in different ways depending on the view other objects have on them.

**Support constraints and/or deductive rules.** — Constraints are well supported in object-oriented databases (e.g., Paton & Díaz, 1999). Deductive object-oriented databases (DOODs, e.g., Ceri & Manthey, 1994) also have the ability to deduce new information from existing information, which could be very useful for deriving taxon names.

**Support for the manipulation/querying of graphs.** — Although object-oriented databases can represent graphs of objects using references or using inheritance, their query language does not support manipulation as a graph. Embedding the necessary specific code in the application could provide support for this; this would, however, also increase development and maintenance overheads.

**Support for recursive behaviour.** — Most of object-oriented query languages are not recursive (e.g., OQL). Some models support the definition of transitive closure (e.g., Kifer & al., 1992), but this is a limited form of recursion as no complete patterns can be defined.
Examples of extended object-oriented databases (Extended OODBs) can be found in the literature. There are object-oriented models that provide features emphasising relationships (e.g., OMS, Norrie, 1993; GOOD, Gemis & al., 1993; GraphDB, Güting, 1994; Telos, Mylopoulos & al., 1990; ConceptBase; Jarke & al., 1995), and others that integrate object orientation with graph databases (e.g., GraphDB). Some of these models also support the ability to acquire roles, based on the presence or absence of particular relationships to other objects (e.g., Adam, Díaz & Gray & al., 1990). However, as none of these databases incorporate all of these features, none can fully satisfy the requirements of taxonomic data.

We initially attempted to implement the Prometheus Taxonomic Model using a graph database model (Prometheus/G). We used the experience gained in this process to apply the same principles to extend the object-oriented database model. We extracted the positive features of each type of Extended OODB and combined them into a single model that satisfies all our requirements. We have called this new Extended OODB model the Prometheus Object-Oriented Model (POOM). For a full description of the design and operation of POOM and the related query language POOL (Prometheus Object-Oriented Language) refer to Raguenaud (2001).

Support for the definition of graphs. — POOM, as other extended object-oriented models, directly supports the definition of relationships as first class objects. This provides an extensive range of features that makes the description of graphs more powerful (e.g., information on relationships, querying of relationship objects). This is one of the main enhancements of Extended OODBs over standard OODBs.

Support for composite entities. — POOM provides mechanisms for the explicit definition of graphs without the loss of semantic information about the relationships being described (Raguenaud & al., 2000). We can use this mechanism to incorporate composite objects within classifications. We are able to treat composite objects as single entities within graphs whilst maintaining the semantic relationships of the objects within the composite object. Representing such relationships as first class objects, allows us to store the semantics of the relationship in the relationship object. This is more efficient and more easily maintainable than the storage of the semantics of relationships within the objects that participate in relationships (cf. traditional object-oriented models). Current Extended OODB models that support composite objects lack the ability to represent the subtle differences that exist between different relationships. They are therefore limited in their ability to implement classifications. Extended Object-Oriented models that do emphasise the graph aspect of the database (e.g., GraphDB), however, leave important semantic issues behind. For example, they do not support the description of semantically rich relationships and their interpretation. POOM is therefore an improvement over current Extended OODBs.

Support for role acquisition. — The POOM model has the inherent ability to support the acquisition of roles. This ability is a natural consequence of models that take advantage of the existence of relationships. Indeed, the existence of a relationship associated with an object can provide it with new information, new attributes, or new constraints (Díaz & Gray, 1990; Norrie, 1995).

Support for constraints and/or deductive rules. — Although few extended object-oriented DBMS offer constraint mechanisms, the ability to support relationships and embed constraints in their definition allows a simpler, more manageable, object-oriented definition of some constraints (e.g., Doherty & al., 1993). The ability to describe constraints on relationships allows POOM to provide a system where constraints do not necessarily imply breaking of encapsulation (e.g., because constraints are on relationships, objects do not need to know the internal state of other objects in order to ensure data integrity).

Support for the manipulation/querying of graphs. — As stated above, POOM provides mechanisms for the declaration of relationships between objects. This is an essential element for the description of graphs as sets of nodes and edges, and is inherently supported by the query language POOL. In addition, POOL supports recursion through the definition of regular expressions that allow the repetition of graph traversals or patterns in the graph. It also supports the reconstruction of graphs after a query, which facilitates the manipulation of graphs as a whole, and not simply as the sum of its parts. With the exception of GraphDB, which supports graph specific operations such as finding paths between nodes, other extended object-oriented query languages do not support graph-specific operations.
limitations of existing database models and query languages. However, using POOM to implement the model removes some of these constraints, and we have improved the model to take advantage of this. We believe that this has enabled us to model the taxonomic process more accurately, without changing the basic philosophy of the original taxonomic model.

**Representation of names.** — The term *Nomenclatural Taxon* has been dropped and is replaced with the term *TaxonomicName*. The *WorkingName* and *AscribedName* objects and their relationships to *TaxonomicName* have also been changed (Fig. 4). A new class of *Name* has been elaborated of which *TaxonomicName* and *WorkingName* are subclasses. *AscribedName* is a role that a *TaxonomicName* can acquire through involvement in a *theAscribedName* relationship. Similarly, a *TaxonomicName* can acquire the role of *CalculatedName* through involvement in a *theCircumscription* relationship. Calculated names can only be assigned by the system, whereas ascribed names are specified by the user and indicate the name under which the taxon concept was published. This may differ from the calculated name if a nomenclatural error has occurred. The working name for a taxon allows the user to assign a temporary name before a calculated or ascribed name is available. In making these changes we remove the abstract concept of ascribed name, and clarify the distinction between ascribed names and working names. The *theCircumscription*, *theAscribedName* and *theWorkingName* relationships are all context-dependent, that is, they are only meaningful within the framework of a particular classification. In order to support this, all three of these kinds of name relationships are associated with an *Author* and a *Publication*.

**Representation of Circumscribed Taxa.** — *Circumscribed Taxa* (CTs) are now represented as the relationship *theCircumscription* between a *Name* object and the abstract class *Circumscription* (Raguenaud, 2001). Figure 2 shows the evolution of the representation of CTs from that described in Pullan & al. (2000) to the present database implementation. Figure 2a shows a conceptual representation of a CT. It has a circumscription, and one or more of a calculated name, an ascribed name or a working name. CTs are subclasses of *Circumscription* as are specimens. Figure 2b shows how this pattern has been remodelled using the features offered by POOM. The CTs have been replaced by *theCircumscription* relationships. Although semantically depicted as a relationship, the new representation has the
same attributes as the original CT. The definition of multiple circumscriptions is supported by this relationship. Each classification is represented by a unique combination of Author and Publication. All the CTs for a classification can be extracted by selecting all the Circumscription relationships that have the appropriate Author and Publication combination. All the elements of a circumscription within the classification can be obtained by taking a subset of the extracted relationships that are all related to the same Name. Figure 3 illustrates how this works in practice, and Fig. 4 shows how all the relationships fit together from an object-oriented perspective.

The above transformation of the CT allows the representation of classifications containing any kind of concept, including classifications that cannot be broken down into classification and nomenclatural elements. It also makes it simpler to query the CT side of the database from a name-based perspective. It should be noted that the Circumscription represents the outcome of two distinct processes: the process of creating a circumscription through the grouping together of specimens and taxa of lower rank, and the process of naming the resulting circumscription on the basis of the type specimens and taxa contained within the circumscription. Unless the circumscription of the CT changes, this CalculatedName will remain unchanged and hence a semi-permanent link to the appropriate TaxonomicName can be made. Creating this permanent link greatly reduces the system overheads required for maintaining CalculatedNames.

Enhanced support for different kinds of nomenclatural types. — The original model allowed only one class of nomenclatural type and the kind of type was represented by the value of an attribute of each instance. To simplify the mechanism for assigning different behaviours to the various kinds of type, we now choose to represent them as subclasses of a general “type” relationship class. This mechanism provides for more expressive (they support constraints), more precise (their semantics are unambiguously defined) relationships. Each of these subclasses can have different properties and behaviour (Fig. 5). A Specimen or a TaxonomicName can therefore acquire the role of taxonomic type if they are involved in a type relationship, i.e., if they are targeted by one of the subclasses of the LinkToType relationship.

Author and publication abbreviations. — We have introduced author and publication abbreviations. Our first model lacked this feature, which is important for the display of taxon names and the generation of reports. The mechanism implemented allows us to show that authors or publications may have different abbreviations in different sources (Fig. 6).

Improved efficiency in calculating names. — In order to reduce the system overheads required to support the automatic naming of taxa, a simple rule set has been elaborated that establishes when the name of a circumscription has to be recalculated. These rules are: 1. If a specimen either becomes a type specimen, or the kind of type that a specimen represents is changed (e.g., through lectotypification), the names of all circumscriptions that include that specimen must be recalculated.

Fig. 3. An illustration of three classifications using an overlapping set of specimens and names. The arrows represent instances of theCircumscription forming a relationship between two names or a name and a specimen. The different line types represent instances of theCircumscription with different Author and Publication assignments. A classification is represented by the collection of lines of the same type. For example, the circumscription of species “a” is specimens 2 & 3 according to the dashed line classification, specimens 1, 2 & 3 according to the solid line classification.
2. If a circumscription is changed, its name must be recalculated along with the name of all circumscriptions that are dependent on it.

**SUMMARY**

We have shown that taxonomic data require five main abilities from the underlying database system:

1. The ability to create overlapping, directed hierarchies.
2. Support for composite entities.
3. The ability to acquire roles or change behaviour according to context.
4. Support for constraints and/or deductive rules.
5. Support for manipulation of graphs and recursive behaviour.

The available database technologies cannot fully support these features. Relational systems are very limited in this respect, as they were designed to represent simple flat structures and support simple operations upon them. Graph models are better suited to databasing taxonomic information as they inherently allow the representation of the relationships that exist between concepts. This enables them to be used directly to model hierarchical classifications. However, without considerable elaboration they cannot offer features such as the definition and manipulation of complex entities. Object-oriented models are better still. They provide abstraction, simplicity of design, and some ability to capture the semantics of a model. They are, however, limited in their support for the representation of relationships between entities. Extended object-oriented models went some way to providing the more precise and varied semantics required. However, it was realised that there was still a need for a system that would provide both powerful
Fig. 5. Taxonomic Types. (a) The general model used for the representation of types; (b) expanded version of (a) showing some of the possible type relationships and illustrating how \textit{LinkToType} can be sub-classed and exclusive constraints can be placed on the relationships. The arc marked XOR indicates that a name may have only one Holo-, Lecto-, or Neo-type. “Here “Specimen” is used as defined in Pullan & al. (2000) and applies to herbarium specimens and illustrations.

Fig. 6. A diagram illustrating how publications and authors are handled in the revised \textit{Prometheus model}. An author can have multiple abbreviations through participating in multiple \textit{theAuthorAbbreviation} relationships. Similarly a publication can have many abbreviations through participation in multiple \textit{thePublicationAbbreviation} relationships. Each abbreviation is linked to the data source to which it applies.
modelling features, and graph definition, manipulation and querying facilities, merging the best features of existing database models.

POOM and POOL capture all the design concepts and implementation requirements identified in this paper, and we have designed and built a database management system (Prometheus) using it. POOM includes facilities for the modelling of relationships as first class concepts, offering a better implementation platform for conceptual models. It also allows the representation of particular modelling features that other systems ignore (such as the addition of attributes to relationships). POOL provides the mechanism for effectively querying and manipulating the structures constructed using POOM. This approach has permitted us to develop a system that is able to capture inherently any kind of classification without requiring the use of virtual objects and without changing the semantics of the underlying conceptual model.

A prototype Prometheus Taxonomic Database has been developed in Java using POET (an Object-Oriented Data Base Management System), and is currently being used to undertake a revision of the genus Globba L. (Zingiberaceae). A web interface to a Prometheus database holding multiple classifications of Apium L. (Apiaceae) and other materials can be seen via the Prometheus Project Website [www.prometheusdb.org].

ACKNOWLEDGEMENTS

We wish to acknowledge the constructive comments of Walter Berendsohn and Marc Geoffroy in their review of this paper. The Prometheus project was funded by BBSRC/ESPRC grants 95/BIF10516 & 754/BIF10518.

LITERATURE CITED


Koymen, K., Cai, Q. 1993. SQL*: a recursive SQL.
APPENDIX

Persistent store = Persistent store system: A mechanism by which information may be kept between successive launches of an application. This can be a simple text file.

Database model = Model: The set of modelling features offered by a persistent store system. For example the relational model offers relations as tables. Most object-oriented models offer classes and objects.

Database query language = Query language: A mechanism that allows retrieval of data stored using a database model. Relational databases provide SQL, some object-oriented databases offer OQL.

Types of databases: Databases grouped according to their database model and/or database query language. For example, relational databases (relational model + SQL), object-oriented databases (object-oriented model, sometimes with OQL but not necessarily).

User application: Everything to which a user has access that was not provided by the database system itself. It can include code embedded in the database (e.g., stored SQL procedures, objects, and deductive rules).

---


