• SABDOOD has an object data model with classes, objects with unique object identifiers (oid's), encapsulation, (single) inheritance, polymorphism and aggregation relationships among classes.

• SABDOOD is expressive. Definition and manipulation of complex objects can be easily done in the system. Object attributes can be simple values (integers, reals, etc.) or structured ones (lists, tuples, sets and multisets) defined with their corresponding constructors [3].

• The SABDOOD query language, based on rules in a Prolog-like syntax, is a declarative language for querying and updating databases in a clean, easy fashion. The language supports structured types (lists, tuples, etc.) and objects (using oid's).

• SABDOOD has function symbols (e.g. sum) in its declarative language. It also permits negation if the program is stratified [16].

• Class methods are defined using SPL. Encapsulation is not strict since attributes of a class can be directly accessed by rules [2].

• SABDOOD has predefined predicates for updating a database that can be freely used by the rules. Integration of these predicates with the declarative language is clean, because updating predicates are subgoals of rules.

• SABDOOD programs can be organized as a collection of modules (sets of rules with a given purpose) to help the programming of large applications.

• SABDOOD keeps both the notion of schema and instance of a database, and that of a database and application. First, classes of a database are defined (schema), and then objects are generated at execution time. The database can be used by many applications, with no need for redefining or creating it.

• Last, but not least, SABDOOD provides a single paradigm to define and build an object-oriented and deductive database, and to manipulate it in a declarative way through user-defined applications.

1.1 Related Proposals

SABDOOD is a general purpose database programming system, whose development has been influenced by related works. Some important aspects of SABDOOD are:

• SABDOOD presents a new object data model. Other query languages use data types of existing languages, such as C++ [17]. The idea for using an own type system is to have a single language to create, update and query a database. It is worth noting that other proposals (e.g. Coral++ [17], Logres [10]) are evolutions of previous works, and programs in these languages must be first translated to the host languages (Coral [15] and Algres [11], respectively), which obscures the functional features of DOOD systems.
As a DBMS, SABDOOD provides support for disk-resident data. Related proposals (e.g. Coral++ [17], Logres [10]) only apply the proposed paradigms in main memory.

Methods in SABDOOD are defined using a language based on extended Horn clauses, that is, clauses where a class can appear as a term. In this sense, it is a declarative language and can be understood in terms of Horn logic. Within the language, lists, tuples, sets and multisets can be used.

SABDOOD has a typed language. Types are important for detecting programming errors as well as for efficiency reasons. SABDOOD makes static and dynamic type verifications.

The rest of the paper is structured as follows. In the next section, the SABDOOD data model is described. Section 3 presents various SABDOOD programs illustrating the features of the language and of the data model. The declarative query language is presented in Section 4 and the imperative language in Section 5. Implementation issues are addressed in Section 6. Finally, in Section 7 future lines of work are discussed.

2 Data Model

The data model is based on classes, used when data is stored in secondary storage, and on nested relations used to handle data in main memory.

Relations Relations are structured values. They are sets (or multisets) of tuples that can be explicitly declared with predicates or can be generated by rules. Predefined operators iterate through collection's members, and term constructors allow arbitrary nesting within a predicate.

Classes Data is organized in classes. A class is a collection of attributes, with an associated type that specifies its structure, and a set of methods that specifies its behavior. SABDOOD classes have the key characteristics that an object-oriented system should have [3].

Object Identity Each object has a unique object identifier (oid); oid's are automatically generated and managed by the system transparently.

Attributes Attributes may be simple types (integer, real, character, string), classes (oid of an object) or structured types (lists, tuples, sets and multisets).

Encapsulation In order to avoid an overhead in handling classes, the attributes of a class can be directly used from the rules. It is not strictly necessary to use class methods to access a class attribute.

Complex Objects SABDOOD provides a variety of constructors for complex objects, such as list (denoted by []), tuples (denoted by ()), sets (by {}) and multisets (by <>).