ROCK & ROLL: A Deductive Object-Oriented Database System

Extended Abstract

Alvaro A.A. Fernandes, Maria L. Barja, Norman W. Paton,
M. Howard Williams, Andrew Dinn, Imara P. Diaz, Alia I. Abdelmoty

Department of Computing and Electrical Engineering,
Heriot-Watt University
Riccarton, Edinburgh EH14 4AS, Scotland, UK
e-mail: <alvaro,marisa,norm,howard,andrew,ceeipd,alia>@cee.hw.ac.uk

1 Introduction

This extended abstract presents an approach to the development of a deductive object-oriented database system, describing the key design decisions and their consequences for implementation. The approach is novel, in that it integrates an object-oriented database system manipulated using an imperative programming language (ROCK – Rule Object Computational Kernel) with a logic language for expressing queries and methods (ROLL – Rule Object Logic Language). The integration is made seamless by deriving both the imperative and logic languages from a single formally defined data model, thereby avoiding impedance mismatches when they are integrated. The basic architecture is summarised in figure 1.

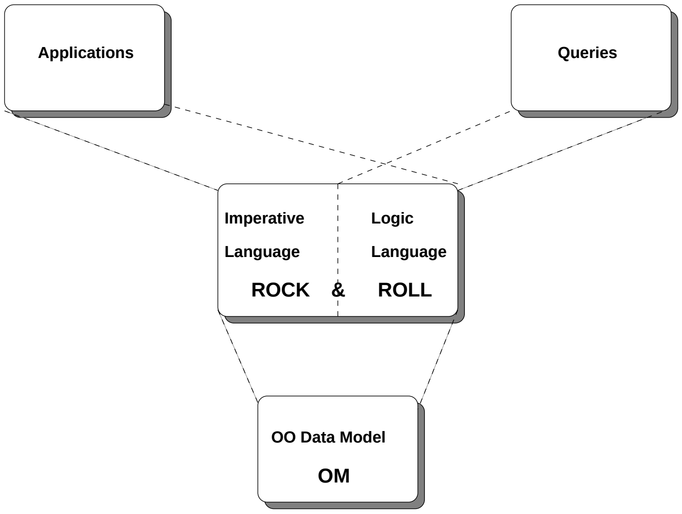

Figure 1: Relationship between the principal components in the architecture.

In the following sections the principal components of the ROCK & ROLL system are introduced, with an indication of what facilities are provided, an illustration of how these can be applied in an example application based upon protein structure data, and a description of the techniques used to provide an implementation on top of the EXODUS extensible database system [CDG+90].

2 Data Model – OM

2.1 Design

Many object-oriented database systems are based upon existing programming languages, and as such support data models which are at a low level, with few of the semantic modelling facilities found in data models such as SDM [HM81] or IFO [AH87]. The data model OM has been most directly influenced by semantic data models, and supports structural abstractions such as specialisation, association, sequentiation and aggregation. As such, the data model OM and its associated diagramming method can be used effectively during database design, thereby easing the transition from conceptual design to implementation.

2.2 Example

This subsection illustrates how the data model of ROCK & ROLL can be used to model protein structure data (the system is also in use with geographic data, as described in [APW+94]). The data used is derived from that described in [GPKF90], an earlier example of the use of an object-oriented database for modelling protein structure data. This application domain poses a number of challenges for a data model, with a requirement to model complex objects with many components; developing an effective approach using the relational model has proved to be difficult [IS89].

The schema of the database is given in figure 2, using the following notation. Secondary object types describe application-specific concepts, such as *residue*s or *strand*s, and are represented in the diagram using rectangles. Primary object types describe scalar values, and are represented in the diagram using ellipses. Labelled directed edges represent modelling features thus: $\bigcirc$ – attributes, $\bigtriangleup$ – specialisation, $\otimes$ – aggregation, $\circledast$ – association, and $\circledcirc$ – sequentiation. For example, *residue* is a specialisation of *sub_component*; a *sub_component* is an association of *atom*s, and has attributes *name*, *position* and *neighbour*.

2.3 Implementation

The data model is supported by an object manager, which has been implemented as a collection of persistent C++ classes within the extensible database system EXODUS [CDG+90]. As such, the facilities of the model are made available to the compilers/interpreters of ROCK and ROLL as abstract data types which allow schema and instance objects to be created, modified, and destroyed.

3 Logic Language – ROLL

3.1 Design

Initial research on deductive languages for databases deliberately restricted the facilities of the logic language used with the database to allow a fully declarative reading of programs, and to enable effective optimisation to be carried out [CGT90]. Recent implementations, however, have sometimes departed from this approach by providing comprehensive facilities for update and control within the logic language [NT89, RSS92]. This strategy complicates the semantics of the logic language, and restricts use of the rule rewriting techniques upon which optimisation depends, but can be avoided, as in Glue-Nail [PDR91], by integrating an imperative programming language with a logic query language.

The approach adopted in ROCK & ROLL follows that of Glue-Nail in that two languages are used, one deductive (ROLL) and the other imperative (ROCK). A number of advantages are offered in ROCK & ROLL compared with Glue-Nail: a richer data model, a more comprehensive imperative

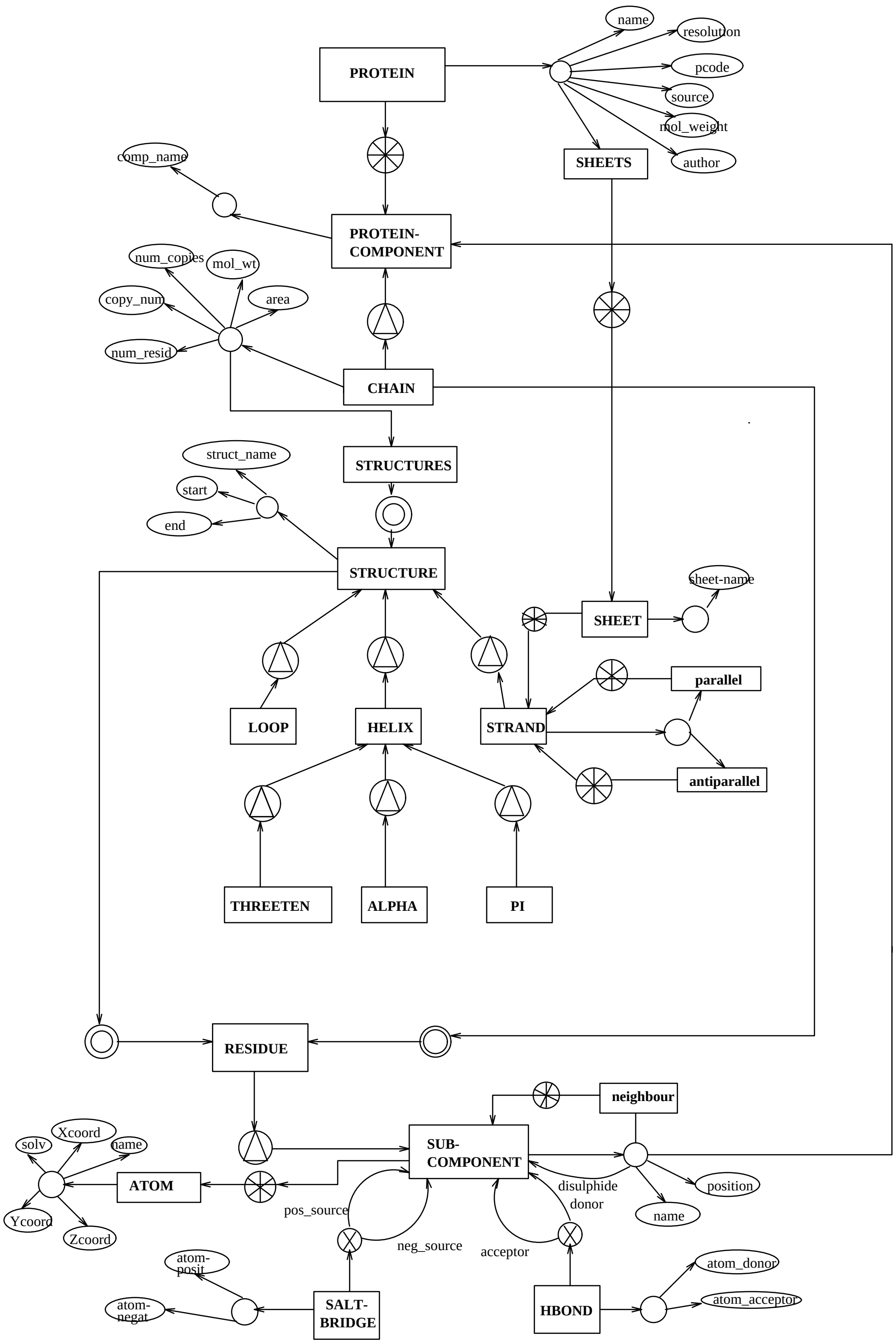

Figure 2: Protein structure database schema in OM.

programming language, object-oriented behaviour modelling facilities (encapsulation, inheritance, overriding, late binding), and static type checking.

From a structural point of view, the logic language ROLL offers all the facilities of Datalog (extended with built in predicates and negation), except that the extensional database over which it operates is structured using the modelling facilities of OM. In contrast with Datalog and related languages, however, the ROLL intensional database has a class-based organisation along inheritance paths, and full support is provided for the behavioural notions associated with the object-oriented paradigm, such as overloading, overriding and late binding. ROLL has been formalised as a mapping onto extended Datalog in the context of the axioms which are used to describe the data model OM [FBPW94].

3.2 Example

ROLL can be used both to express queries and to define methods. A ROLL atom has the form $\beta(\tau_1, \ldots, \tau_{n-1})@\alpha == \tau_n$, which is read as "send the *message* β with the arguments $\tau_1, \ldots, \tau_{n-1}$ to the object α, unifying the result with τ_n". Where a method has no distinguished result, as is the case for all ROLL methods, the corresponding form of atom is $\beta(\tau_1, \ldots, \tau_n)@\alpha$. The following is an example ROLL query over the protein database schema presented in figure 2:

Retrieve the names of all structures which contain alanene residues.

```
[{SN} | get_struct_name@S == SN,
        get_element@S == R,
        get_name@R == "alanene" ]
```

A number of points in the above example bear explanation. The query is enclosed in a set expression which projects out the set of bindings for the variable *SN* obtained by evaluating the ROLL query. Methods that are prefixed by *get_* are generated at compile time to provide access to the values of properties and components of constructions. The types of variables are inferred, using a type inference system, from the way in which the variables are used in the query – for example, the type of *S* is inferred to be *structure* from the fact that it is sent the message *get_struct_name*. The above query can thus be read as "Retrieve all values for *SN* such that the *struct_name* of *S* is *SN*, *R* is a *residue* in the sequence from which *S* is constructed, and the *name* of *R* is *alanene*.

ROLL rules are defined as methods attached to classes. For example, the following method definition specifies the transitive closure of the *parallel* attribute on the class *strand*:

```
class strand
    ...
    isparallel(strand)
    begin
        isparallel(Direct)@Strand :-
            get_parallel@Strand == Parallel,
            get_element@Parallel == Direct.
        isparallel(Indirect)@Strand :-
            get_parallel@Strand == Parallel,
            get_element@Parallel == Direct,
            isparallel(Indirect)@Direct.
    end
end-class
```

3.3 Implementation

In common with other logic languages, a range of techniques exist which can be used to optimise and evaluate ROLL queries and methods. The approach which has been adopted uses a variant on static filtering [KL90] for optimisation, with evaluation being based upon bottom-up traversal of an extended relational algebra tree.

4 Imperative Language – ROCK

4.1 Design

ROCK is characterised as statically typed imperative object-oriented language which supports the definition, creation and manipulation of databases described using the data model presented in section 2. The language can be regarded as the conjunction of a data definition language for schema declarations, and a data manipulation language that provides access and allows operations to be performed on the objects stored in the database. The data definition language provides support for the definition of an object oriented schema for the application, and distinguishes between types and classes. Each type declaration provides the definition of the structure for a corresponding class. A class declaration defines the behaviour of objects which conform to the structure defined by the type associated with the class. The data manipulation language is the vehicle for the development of application code, and provides support for the creation and manipulation of objects. The constructs supported by the data manipulation language are roughly those supported by most imperative programming languages, i.e. assignment, control structures such as *while* and *if then else*, and I/O statements. It also provides a *foreach* construct to iterate over the elements of a set, a list or the instances of a class. The system also provides a number of built-in methods to manipulate constructed objects. Further, for each property in a type definition the system generates a pair of methods whose names are formed from the name of the respective property with the prefixes *get_* and *put_*. These methods allow the object referenced by a property to be retrieved or updated. ROCK also provides facilities for the definition and manipulation of persistent data.

4.2 Example

An example of a ROCK method definition is *read_sheet* in figure 3, which is one of a collection of methods used to populate the protein structure database. The method illustrates a number of standard features – reading data, message sending, updating attributes, and creating objects.

4.3 Implementation

ROCK has also been implemented using the persistent C++ system EXODUS, with both type-checking and evaluation based upon sending messages to an object-oriented internal representation of a program. This implementation strategy leans heavily on the structural operational semantics of the language, as described in [BPW94].

5 Integration

The languages ROCK and ROLL have been fully integrated to allow their combined use in supporting advanced applications. ROLL queries can be embedded in ROCK, and side-effect free ROCK methods can be called from ROLL queries and rules. For example, the following statement assigns to the ROCK variable *v1* the set of all *strand*s which are directly or indirectly parallel to strand *s1*, where *s1* is a ROCK variable embedded in ROLL (the embedding being represented by the *!*):

```
v1 := [{S} | isparallel(S)@!s1];
```

```
class sheet
    ...
    read_sheet()
    begin
        var strd: strand;
        var shname: string;
        var flag: string;

        read shname;
        put_sheet_name(shname)@self;
        write "Sheet Name: ",get_sheet_name@self,nl;
        read flag;
        while (flag = "*") do
        begin
            strd:= new strand();
            read_strand()@strd;
            add_member(strd)@self;
            read flag;
        end
    end
end-class
```

Figure 3: Method definition in ROCK.

The above example illustrates the invocation of the ROLL method *isparallel* with its message recipient bound. However, as a logic language method, it can also be called with an unbound message recipient thus:

```
v1 := [{S} | isparallel(!s1)@S];
```

More complex queries, such as that presented in section 3 can also be embedded in ROCK. For example, the following code fragment declares the variable *v2* as an association of strings, and uses the ROCK loop construct *foreach* to print each of the strings returned as a result of the embedded ROLL query:

```
var v2 := [{SN} | get_struct_name@S == SN,
                  get_element@S == R,
                  get_name@R == "alanene" ]
foreach n in v2 do write n, nl;
```

A more comprehensive discussion of the integration of ROCK & ROLL is given in [BPF+94].

6 Conclusion

This extended abstract has briefly presented the facilities of the deductive object-oriented database system ROCK & ROLL. The resulting system has the following characteristics:

1. It is fully formally specified [FBPW94].
2. It is fully implemented.

3. It supports an object-oriented data model with comprehensive semantic data modelling facilities.
4. It supports a logic language for the specification of queries and methods which has a straightforward semantics based upon that of Datalog, and which is free from extra-logical facilities.
5. It supports an imperative object-oriented programming language with comprehensive facilities for creating and manipulating objects described by the data model.
6. It allows the logic and imperative languages to be used together in a single program in a way which minimises impedance mismatches, as both languages are derived from the same data model. Such an integration allows users to select the most appropriate language for different parts of a complex task.

A significant claim for the resulting system is that the strengths of each of the paradigms represented in ROCK & ROLL have been used to overcome the weaknesses of the other, to yield an effective database programming system which is well suited to advanced, data-intensive applications.

Acknowledgements The work that resulted in this paper has been funded by the UK Science and Engineering Research Council through the IEATP programme, and their support is duly acknowledged. We would also like to thank Dr. Keith G. Jeffery of Rutherford Appleton Laboratory for useful discussions on the subject of this paper, and Dr. J.M.P. Quinn representing ICL and Mr Neil Smith of Ordnance Survey as the industrial partners in the project.

References

[AH87] S. Abiteboul and R. Hull. IFO: A Formal Semantic Database Model. *ACM Transactions on Database Systems*, 12(4):525–565, December 1987.

[APW+94] A.I. Abdelmoty, N.W. Paton, M.H. Williams, A.A.A. Fernandes, M.L. Barja, and A. Dinn. Geographic Data Handling in a Deductive Object-Oriented Database. In *Proc. Databases and Expert Systems Applications (DEXA)*. Springer-Verlag, 1994.

[BPF+94] M.L. Barja, N.W. Paton, A.A.A. Fernandes, M.H. Williams, and A. Dinn. An Effective Deductive Object-Oriented Database Through Language Integration. In *Proc. 20th International Conference on Very Large Data Bases*, 1994.

[BPW94] M.L. Barja, N.W. Paton, and M.H. Williams. Semantics Based Implementation of a Deductive Object-Oriented Database Programming Language. 1994. to be published in J. Programming Languages.

[CDG+90] M. Carey, D. DeWitt, G. Graefe, D. Haight, J. Richardson, D. Schuh, E. Shekita, and S. Vandenberg. The EXODUS Extensible DBMS Project: An Overview. In S. Zdonik and D. Maier, editors, *Readings in Object-Oriented Databases*, CA 94303-9953, 1990. Morgan Kaufman Publishers, Inc.

[CGT90] S. Ceri, G. Gottlob, and L. Tanca. *Logic Programming and Databases*. Springer-Verlag, Berlin, 1990.

[FBPW94] A.A.A. Fernandes, M.L. Barja, N.W. Paton, and M.H. Williams. A Formal Specification of ROCK & ROLL: A Deductive Object-Oriented Database System. 1994. Submitted for publication.

[GPKF90] P.M.D. Gray, N.W. Paton, G.J.L. Kemp, and J.E Fothergill. An Object-Oriented Database for Protein Structure Analysis. *Protein Engineering*, 4(3):235–243, 1990.

[HM81] Michael Hammer and Dennis McLeod. Database Description with SDM: A Semantic Database Model. *ACM Transactions on Database Systems*, 6(3):351–386, September 1981.

[IS89] S.A. Islam and M.J.E. Sternberg. A relational database of protein structures designed for flexible enquiries about conformation. *Protein Engineering*, 2(6):431–442, 1989.

[KL90] M. Kifer and E. Lozinskii. On Compile-Time Query Optimization In Deductive Databases By Means Of Static Filtering. *TODS*, 15(3):385–426, 1990.

[NT89] S.A. Naqvi and S. Tsur. *A Logical Language for Data and Knowledge Bases*. Computer Science Press, Rockville, MD, 1989.

[PDR91] G. Phipps, M.A. Derr, and K.A. Ross. Glue-Nail: A Deductive Database System. In James Clifford and Roger King, editors, *Proc. ACM SIGMOD International Conference on the Management of Data*, pages 308–317. ACM Press, 1991.

[RSS92] R. Ramakrishnan, D. Srivastava, and S. Sudarshan. CORAL-Control, Relations and Logic. In Li-Yan Yuan, editor, *Proceedings of the 18th International Conference on Very Large Databases*, pages 239–250. Morgan Kaufman, 1992.