Object-Oriented Database Programming Languages Founded on an Axiomatic Theory of Objects

Alvaro A.A. Fernandes, M. Howard Williams, Norman W. Paton,
Maria L. Barja, Andrew Dinn
Department of Computing and Electrical Engineering
Heriot-Watt University
Riccarton, Edinburgh EH14 4AS, Scotland, UK

Position paper
ECOOP'94 Workshop on Logical Foundations of Object-Oriented Programming

Introduction This paper is based on our experiences in designing, formalizing and implementing the deductive object-oriented database (DOOD) system ROCK & ROLL [5, 7, 13, 14, 15, 16]. Our overall goal has been to bring together the deductive and the object-oriented approaches in a clean, uniform and practical way for database systems. We have achieved this by first developing a first-order theory that captures features of the object-oriented paradigm in a standard logical framework, and then developing a pair of languages, ROCK (Rule-Object Computational Kernel) and ROLL (Rule-Object Logic Language), that cater for the declarative and the imperative styles in database programming. In this position paper, we concentrate on the form taken by the logical characterization of the object-oriented data model (OODM) that underpins these two language components constituting the system as seen by users. Note that although a friendlier, block-structured surface syntax is available to users of ROCK & ROLL (see [5], for instance), in this paper we use logical form for all our examples as this is closer to the central argument being developed.

ROCK & ROLL The most distinctive feature of ROCK & ROLL as a database system is its mismatch-free integration of a persistent imperative object-oriented programming language (ROCK) and a deductive object-oriented query language (ROLL). This feature makes ROCK & ROLL a practical system with truly scalable functionality beyond that of most alternative proposals for DOODs [3, 8, 9, 18, 19, 25]. ROCK & ROLL is fully implemented (in E [24], a persistent C++ over EXODUS) and is proving in tests to be a better platform for certain classes of application than either purely deductive or purely object-oriented databases. The pre-release version of the ROCK & ROLL system is being tested with a geographic application since it has been shown that geographic information systems (GISs) have a need for both object-orientation and deduction [1, 2] and hence represent a particular challenge for DOOD systems.

Declarative *or* Imperative Consider the large amounts of structurally complex data, the necessity of providing multiple behaviourally-defined abstractions of the same object, and the algorithmic complexity of the computational geometry required to derive and manipulate spatial objects. These factors combine to make efficiency an absolute requirement in database platforms for GISs. This requirement creates a dilemma for a practical DOOD system: on the one hand, efficient, ad-hoc associative querying over very large databases requires state-of-the-art query optimization techniques, and hence the declarative nature of the query language cannot be renounced at low cost; on the other hand, most manipulations of spatial data rely on algorithms in which imperative expression of the execution flow cannot be renounced if space and time complexities are to be kept within practical bounds.

To Control *and* Not To Control The analysis above indicates that in different circumstances explicit control may or may not be desirable. There is a need for the querying component to be kept purely declarative, so as to take advantage of every possible opportunity for optimization. There is also the need for the manipulation component to be free to rely on explicit control of both the flow of execution and the processes that populate the database, so as to produce the most efficient formulations of the crucial manipulations at the geometric level. The integration of a purely-declarative query language with an imperative manipulation language is one solution to the dilemma above, but poses problems of its own. These take the form of impedance mismatches, of which two are most visible: on the type system and on the evaluation strategy.

The Central Role of the OODM By independently defining a model of structural and behavioural data that fixes a type system and an evaluation strategy for two otherwise independent language components the impedance mismatches above are solved, whilst keeping orthogonal the question of whether the style of expression adopted in each language component is declarative or imperative. Thus, it falls on the underlying model to define the common ground over which a deductive and an imperative language can be integrated without incurring impedance mismatches. The mismatch-free nature of the integration of two distinct languages is then the direct consequence of the fact that they are both being derived from an independently and formally defined, logic-based OODM.

Mismatch-Free Integration No type-related impedance mismatch arises between ROCK and ROLL because every object that can be denoted by expressions of one language can be denoted by expressions of the other, under a type system that is defined by the underlying model, and, as a consequence, is shared by, and independent of, both the language components. In ROCK & ROLL, static type checking with type inference is provided with no user-discernible language boundary. Similarly, no evaluation-related impedance mismatch arises because the user-discernible aspects of the computation model (viz. encapsulation, overloading, overriding and late-binding) are disciplined at the level of the underlying model, and, as a consequence, are shared by, and independent of, both the language components. In ROCK & ROLL, safe methods (i.e. those without global side effects) coded in one language can be invoked in and mixed with code written in the other with no user-discernible language boundary. Programmers are aware that two distinct languages are involved in the code they produce only to the extent that ROCK and ROLL are *meant* to complement one another and provide alternative styles of implementing solutions. This shows up in few and justifiable points (e.g. the declaration of ROCK interfaces are obliged to distinguish the type of the result; ROLL interfaces, taking advantage of flexible invocation patterns of logic programs in general, do not).

Logical Foundations for DOODs This scenario, however, characterizes an impasse in terms of logical foundations, because the desirable computational properties of the logic that underlies database systems is only achieved through radical specialization [10]. This logic of databases is very brittle and most attempts to capture side-effects and explicit control directly by extending the logical machinery jeopardize desirable properties or block the way towards more comprehensive optimization. The alternative of going back to foundations is as exciting as it is nontrivial. Even if one re-emerges with a more comprehensive logical machinery, it is reasonable to assume that mechanizing the new logic to operate over a very large persistent store of complex structural and behavioural data will take considerable time and effort. In the ROCK & ROLL project, our position with respect to this impasse has been quite pragmatic: the logical underpinning of the new framework concentrates on those aspects where practical gains are most guaranteed. In the current stage of database research, this translates to query optimization and evaluation. Correspondingly, the ROCK & ROLL database system is endowed with logical foundations that make its query optimization and query evaluation strategies as simple, practical and efficient as those of Datalog [10]. At the same time, the logic-based data model smooths the path towards delegating to a non-logic-based language issues such as side-effects and explicit control. Note that, albeit in a different style, the imperative language is fully formalized in [6], using structural operational semantics [22], but is logically underpinned only to the extent that the states involved possess a well-defined logical characterization.

An OODM as a Class of Axiomatic Theories The OODM underlying ROCK & ROLL is characterized as a class of first-order axiomatic theories. Our goal in the remainder of this position paper is to overview the particular logical form taken by the OODM underlying ROCK & ROLL and how this form simplifies the task of providing end-users with fully-fledged object-oriented functionality for declarative querying and imperative manipulation of databases. Although our results are of interest as a practical approach to the problem of providing logical foundations for DOOD systems, they do not address the dynamic aspects of the object-oriented paradigm. Because we are concerned with databases only, we do not use logic to model how objects interact with one another, but rather how application programs and their users interact with objects in a persistent store. We adopt, therefore, a limited but practical perspective in which the required logical underpinning need not model state transitions — all deductive processes can safely assume a single determinate closed state. Our approach is nevertheless of relevance to object-oriented programming insofar as our axiomatization is comprehensive and extensive for most of the common static features of major object-oriented programming languages.

The Structure of the Axiomatization Essentially, four sets of axioms[1] can be distinguished in every theory in the class identified with the OODM:

- Axioms in SCH are ground atomic formulas asserting schema-level application-specific information, including the definition of class-specific operation interfaces. For instance,

  specialization-of(employee, person) ←
  described-by(person, name) ←
  income-tax-due : employee(tax) ←
  tax-band : company-employee(salary, percentage) ←
  tax-band : self-employed(salary, percentage) ←

- Axioms in INS are ground atomic formulas asserting instance-level application-specific information, including the definition of operation implementations. For instance,

  instance-of(!4, employee) ←
  property(!4, name, john) ←

  where !4 denotes, in the example above, the unique, lifelong identifier of the object in class 'employee' whose current 'name' is 'john'. As an example of code implementing an operation interface to retrieve the income tax deduced from an employee's pay, one could have:

  income-tax-due:employee(Emp, Tax) ←
  property(Emp, salary, Salary),
  tax-band(Emp, Salary, Band),
  Tax = Band * Salary

  Note that interfaces are uniquely associated with classes and implementations have a class parameter so that overloading/overriding can be taken into account. In the example above, 'tax-band' is itself an application-specific operation with two implementations. It is overloaded depending on whether the recipient is a 'company-employee' or is 'self-employed'. Below there is a brief discussion of how the ensuing ambiguity is resolved through late-binding.

- Axioms in C characterize application-independent implicit information conveyed by the modelling relationships with which users analyze and design applications. They assume the form of universal closures taken to be implicitly conjoined. Classification, attribution, specialization, association (yielding unordered collections), sequentiation (yielding ordered collections) and aggregation are the modelling relationships directly catered for. For instance, the transitivity of the is-a (or a-kind-of) relationship is captured by $\forall(is\_a(x,y) \wedge is\_a(y,z) \rightarrow is\_a(x,z))$, whilst the propagation of classification through is-a links is defined by: $\forall(instance\_of(x,y) \rightarrow (is\_a(y,z) \rightarrow instance\_of(x,z)))$. Also among the axioms in C are those propagating properties and operations through is-a links.

- Axioms in E characterize application-independent well-formedness properties of modelling relationships, and, therefore, determine legal schemas and the legal states induced by these. They assume the form of negations of existential closures taken to be implicitly conjoined. For instance, the is-a relationship is not well-formed if it is symmetrical, i.e. if the following axiom is false:

[1] The complete list of axioms and axiom-schemes characterizing the OODM is available in [16].

$\neg\exists(\neg(x = y) \wedge is\_a(x, y) \wedge is\_a(y, x))$. Among the axioms in E are those imposing a discipline on how structure and behaviours are inherited, possibly from multiple superclasses, under overloading and overriding conditions. In particular, axioms in E guarantee that the declarations in SCH are such that every overloaded message identifier can be late-bound to a unique method that overrides all other candidates, according to the most-specific type of the recipient. Also, axioms in E guarantee that each interface in SCH (a triple consisting of the operation name, the operation signature and the class in which the operation is declared) has exactly one corresponding implementation in INS.

Declared, Completed and Constrained Object Theories The set of formulas $B = SCH \cup INS$ is called a *declared (object) theory* and characterizes the declared information content of an application domain. The set of formulas $D = B \cup C$ is called the *completed (object) theory* and, through logical consequence, characterizes the complete information content of an application domain iff D is such that every axiom in E is true in every model of D. In this case D is said to be *constrained* and the complete information content characterized by it is said to be *valid*.

Expressiveness of Object Theories Note that this axiomatic characterization suffices to directly capture more semantics of application domains than is possible in the classic relational model of data or in its nested extensions, viz. the atoms in SCH and the axioms in $C \cup E$ capture directly in the database knowledge about the application that in relational systems must remain external to the database. Furthermore, the basic discipline of behavioural inheritance and all the static conditions required for overloading, overriding and late-binding to take place are axiomatized as well. Models of constrained completed theories not only have much more mathematical structure than models of pure relational theories, but they can also be used by an object-oriented style of behavioural modelling in a way that pure relational structures cannot. With respect to conventional object-oriented databases (OODBs) the way is open for a declarative logic-based language to express queries, derived data and integrity constraints. The ROCK & ROLL OODM captures all the relevant mandatory features of [4], where most other proposals for DOODs capture fewer structural aspects and almost no behavioural ones. Nevertheless, the ROCK & ROLL OODM is wholly within first-order logic. In fact, our axiomatization is so designed as to make it possible to cast every constrained completed theory as a set of Horn clauses.

Desirable Consequences of the Axiomatization Because every object theory D can be cast in Horn-clausal form, it follows that under the usual representational assumptions [17] underlying databases in general, one can ignore all but the least Herbrand model of D. An OODB can then be identified with the least Herbrand model of D as in Reiter's logical reconstruction of relational database theory [23]. In other words, the declarative semantics of object theories are reduced to those of logic programs [12]. Note that if, as is customary, the task of enforcing the axioms in E (which, in clausal form, contain Skolem function-symbols) is delegated to a component other than the query evaluator, we can go a step further and identify the syntactic and semantic characterization of the language of D with those of Datalog. This simplifies even further the proof- and model-theoretic characterization of constrained completed object theories. More importantly, it establishes that in implementing a query optimizer and query evaluator for ROLL one can proceed along very well-known lines [10]. In fact, most results in the theory of deductive relational database carry over almost intact.

Reduction to Known Cases Our approach can now be understood as reductionist (as are, in different ways, [8, 11, 26]) in the sense that the semantics of object theories are defined by a mapping onto Datalog and thus remain in the firm territory of deductive database theories. In broad terms, since $MET \subset INS$ consists of sets of class-parameterized clauses implementing operation interfaces, the axioms in $C \cup MET$ can be characterized as the intensional part of a deductive database. The logical consequences of $D = B \cup C$ that are not in $B \setminus MET$ follow from the application semantics expressed in the rules in $C \cup MET$. It is then straightforward to extend D into a fully-fledged deductive database DD by simply adding a set of application-specific deductive rules R in addition to those in MET. The resulting set of clauses $DD = D \cup R$ models as many forms of application-specific implicit information in a uniform way as deductive relational databases (DRDBs) do. It also provides a precise characterization of query evaluation with very desirable termination properties and optimization opportunities [10], viz. that of Datalog. From a purely formal point of view DD is syntactically and semantically equivalent to a DRDB [10, 17]. However, due to the schema-level information in SCH, the behaviours captured in INS and the properties captured in C and E, any DD is a stronger theory that any DRDB, precisely in the sense that the theory of Abelian groups is stronger than the theory of groups. It remains to position ROCK & ROLL in terms of encapsulation and late-binding, which are features related to interaction modes with the objects in the database.

Late-Binding of Methods To model late binding ROCK & ROLL rewrites a language L in which disciplined ambiguity is allowed into another language L' in which any ambiguity has been resolved deterministically and uniquely. Technically, therefore, two languages are at play: in one, which we call ROLL, atoms can be overloaded when they appear in queries or in the bodies of methods; in the other, which we call CROLL (for Compiled ROLL), every such overloaded atom must have been disambiguated on the basis of the most specific type of the recipient previous to resolution against the clause base. A most specific type is guaranteed to exist by well-formedness axioms in E that constrain the interface declarations in SCH. The standard resolution step in the proof theory of deductive languages must then be preceded by this disambiguation. The overall effect is that a ROLL goal (e.g. 'tax-band(Emp, Salary, Band)') is mapped into a CROLL goal with an explicit class parameter obtained from the type of the recipient 'Emp' (e.g. 'tax-band : self-employed(Emp, Salary, Band)'). Every method invocation is therefore cast in what is technically a distinct language, i.e. ROLL, from that of object theories, i.e. CROLL. Note that explicit typing is allowed, so that the default binding rule can be overridden by the programmer, and, as previously mentioned, static type checking and type inference are implemented uniformly for both ROCK and ROLL and flow bidirectionally across them.

Encapsulation of States This rewriting approach opens the way for a straightforward and intuitive account of encapsulation. ROLL is defined to have a restricted set of predicate symbols, containing only those occurring in the interfaces declared in SCH. This models the fact that CROLL is an implementor's language which only "exports" operation names that do not break encapsulation. Note, for instance, that in the implementation of the example method 'income-tax-due' given above the CROLL predicate symbol 'property' is not a ROLL predicate symbol, whereas 'tax-band' is a predicate symbol, in this case an overloaded one, of both ROLL and CROLL. The overall effect is that ROLL users cannot break encapsulation and directly access the details of the state of an object is represented.

Practical Architectural Advantages The axiomatic approach used in ROCK & ROLL has significant practical advantages in that it suggests a tripartite system architecture to combine the three main components with intuitive interfaces among them. In the current implementation the OODM is implemented as an object store that embodies the semantics of the axiom sets C and E and serves the extensional data in SCH and INS to the two interpreters in the form of persistent C++ objects. The ROCK interpreter provides user-friendly syntax for data definition and manipulation, management functions for persistent environments, and an extensive set of primitive types with built-in operators that match the semantic constructs defined in the OODM. The ROCK interpreter is also responsible for type checking and type inferencing over integrated ROCK & ROLL code. Finally, the ROCK interpreter serves the result of evaluating method invocations that originate in ROLL. The ROLL interpreter performs query optimization and evaluation, possibly serving query results to the ROCK interpreter. Query optimization uses static filtering over query graphs and evaluation is, currently, bottom-up using processing trees whose nodes correspond to relational algebra operators extended with a fixpoint facility. A top-down approach based on an extension of the WAM has already been designed and is scheduled for implementation.

Extension Paths The logical characterization described above lends itself naturally to extensions that are based on Horn-clausal logic. We have designed and prototyped a framework to incorporate active behaviour on top of the deductive engine of ROLL using the Kowalski-Sergot event calculus as the foundation of a logic of events that is then used to define a language of event-condition-action rules with an execution model that requires minimal modifications to the current ROLL interpreter. This extension is intended to make ROCK & ROLL a suitable platform for the investigation of active rules and for the construction of an active rule toolkit concentrating on facilities for active-rule analysis and optimization.

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 thank our colleague Alia I. Abdelmoty for her many contributions to the work described in this paper. 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

[1] Alia I. Abdelmoty, M. Howard Williams, and Norman W. Paton. Deduction and Deductive Databases for Geographic Data Handling. In David Abel and Beng C. Ooi, editors, *Advances in Spatial Databases: Third International Symposium, SSD '93*, LNCS 692, pages 443–464, Singapore, June 1993. Springer-Verlag. ISBN 3-540-56869-7, 0-387-56869-7.

[2] Alia I. Abdelmoty, M. Howard Williams, and Michael Quinn. A Rule-Based Approach to Computerized Map Reading. *Information and Software Technology*, 35(10):587–602, October 1993.

[3] Serge Abiteboul. Towards a Deductive Object-Oriented Database Language. *Data & Knowledge Engineering*, 5:263–287, 1990. Preliminary Version in [20], pages 453–472.

[4] Malcolm Atkinson, François Bancilhon, David DeWitt, Klaus Dittrich, David Maier, and Stanley B. Zdonik. The Object-Oriented Database System Manifesto. In *[20]*, pages 223–240. 1990.

[5] Maria Luisa Barja, Norman W. Paton, Alvaro A.A. Fernandes, M.Howard Williams, and Andrew Dinn. An Effective Deductive Object-Oriented Database Through Language Integration. To appear in *VLDB '94 - Proceedings of the 20th International Conference on Very Large Data Bases*, Morgan Kaufmann, Santiago, Chile ., 1994.

[6] Maria Luisa Barja, Norman W. Paton, and M.Howard Williams. Design of an Object-Oriented Database Programming Language for a DOOD. Technical Report TR92016, Department of Computing and Electrical Engineering,Heriot-Watt University, December 1992.

[7] Maria Luisa Barja, Norman W. Paton, M.Howard Williams, and Alvaro A.A. Fernandes. A Database Programming Language for a Deductive Object-Oriented Database. To appear in *Journal of Information Science and Technology.*, 1994.

[8] Elisa Bertino and Danilo Montesi. Towards a Logical-Object Oriented Programming Language for Databases. In Alain Pirotte, Claude Delobel, and Georg Gottlob, editors, *Advances in Database Technology - EDBT'92, 3rd International Conference on Extending Database Technology*, LNCS 580, pages 168–183, Vienna, Austria, March 1992. Springer-Verlag. ISBN 3-540-55270-7, ISBN 0-387-55270-7.

[9] Filippo Cacace, Stefano Ceri, S. Crespi-Reghizzi, Letizia Tanca, and Roberto Zicari. Integrating Object-Oriented Data Modeling with a Rule-based Programming Paradigm. In *Proceedings of the 1990 ACM SIGMOD International Conference on the Management of Data*, pages 225–236, Atlantic City, NJ, May 1990. ACM Press. SIGMOD Record, 19(2), June 1990.

[10] Stefano Ceri, Georg Gottlob, and Letizia Tanca. *Logic Programming and Databases.* Springer-Verlag, Berlin, 1990. ISBN 3-540-51728-6, ISBN 0-387-51728-6.

[11] Mukesh Dalal and Dipayan Gangopadhyay. OOLP: A Translation Approach to Object-Oriented Logic Programming. In *[20]*, pages 593–606. 1990.

[12] Maarten H. Van Emden and Robert A. Kowalski. The Semantics of Predicate Logic as a Programming Language. *Journal of the ACM*, 23(4):733–742, October 1976.

[13] Alvaro A.A. Fernandes, Maria L. Barja, Norman W. Paton, and M. Howard Williams. A Deductive Object-Oriented Database for Data Intensive Application Development. In *Advances in Databases: Proceedings of the 11th British National Conference on Databases (BNCOD 11)*, LNCS 696, pages 176–198, Keele, UK, July 1993. Springer-Verlag. ISBN 3-540-56921-9, 0-387-56921-9.

[14] Alvaro A.A. Fernandes, Norman W. Paton, and M. Howard Williams. A Logical Query Language for an Object-Oriented Data Model. In *[21]*, pages 234–250. 1994.

[15] Alvaro A.A. Fernandes, M. Howard Williams, and Norman W. Paton. A Comparative Formalization of Deductive Object-Oriented Databases, 1994. Submitted for publication.

[16] Alvaro A.A. Fernandes, M. Howard Williams, and Norman W. Paton. Deductive Object-Oriented Databases as Horn Logic-Programs, 1994. Submitted for publication.

[17] Hervé Gallaire, Jack Minker, and Jean-Marie Nicolas. Logic and Databases: A Deductive Approach. *ACM Computing Surveys*, 16(2):153–185, June 1984.

[18] Matthias Jarke, Stefan Eherer, Rainer Gallersdoerfer, Manfred A. Jeusfeld, and Martin Staudt. ConceptBase - A Deductive Object Base Manager. Technical Report 93-14, Aachener Informatik-Berichte/RWTH Aachen, October 1993.

[19] Michael Kifer, Georg Lausen, and James Wu. Logical Foundations of Object-Oriented and Frame-Based Languages. Technical Report 93/06, Department of Computer Science, State University of New York at Stony Brook (SUNY), April 1993. Revised June 1993.

[20] Won Kim, Jean-Marie Nicolas, and Shojiro Nishio, editors. *Deductive and Object-Oriented Databases (First International Conference DOOD'89, Kyoto)*. Elsevier Science Press (North-Holland), Amsterdam, 1990. ISBN 0-444-88433-5.

[21] Norman W. Paton and M. Howard Williams, editors. *Rules in Database Systems: Proceedings of the 1st International Workshop, Edinburgh, Scotland (30 August–1 September 1993)*. Workshops in Computing Series. Springer-Verlag, London, 1994. ISBN 3-540-19846-6, 0-387-19846-6.

[22] G.D. Plotkin. A structural approach to operational semantics. Technical Report DAIMI FN-19, Aarhus University, Dept. of Computer Science, Aarhus, Denmark, 1981.

[23] Raymond Reiter. Towards a Logical Reconstruction of Relational Database Theory. In Michael L. Brodie, John Mylopoulos, and Joachim W. Schmidt, editors, *On Conceptual Modelling: Perspectives from Artificial Intelligence, Databases, and Programming Languages*, pages 191–233. Springer-Verlag (Topics in Information System Series), New York, NY, 1984.

[24] Joel E. Richardson and Michael J. Carey. Implementing Persistence in E. In John Rosenberg and David Koch, editors, *Persistent Object Systems: Proceedings of the Third International Workshop*, pages 175–199. Springer-Verlag, London, 1989.

[25] Divesh Srivastava, Raghu Ramakrishnan, Praveen Seshadri, and S. Sudarshan. Coral++: Adding Object-Orientation to a Logic Database Language. In Rakesh Agrawal, Seán Baker, and David Bell, editors, *Proceedings of the Nineteenth International Conference on Very Large Data Bases*, pages 158–170, Dublin, August 1993. Morgan Kaufmann Publishers. ISBN 1-55860-152-X.

[26] Carlo Zaniolo. Object Identity and Inheritance in Deductive Databases – An Evolutionary Approach. In *[20]*, pages 7–24. 1990.