Chapter 2.1.2
The DBPL Project:
Advances in Modular Database Programming

Joachim W. Schmidt and Florian Matthes

Technical University Hamburg-Harburg
Harburger Schloßstraße 20
D-21071 Hamburg, Germany

Summary This paper is a synopsis of [23].

1. Introduction

In the DBPL project we tackled the problem of supporting data-intensive applications in a single framework, clean and simple in its conceptual foundation and free of technical mismatches. Our development of specific language extensions required by data-intensive applications was guided by a strict adherence to the language design principles of simplicity, orthogonality and abstraction. The project started in the mid 70ies with Pascal/R [16, 17, 18, 6] and Modula/R [10, 12] where we first suggested some high-level language constructs for data of type relation. This research culminated in the delivery of a mature implementation of the DBPL system [23, 22, 15, 14, 19], a database programming language based on Modula-2.

2. DBPL Language Concepts

In the full paper [23] we describe the rationale, the conceptual basis for the language design [21] and the specific language elements in detail. Conceptually, we based the DBPL language on Modula-2 with built-in extensions for data-intensive applications:

- a parametric bulk type constructor for "keyed sets" or relations;
- a module concept which supports sharing across programs and implies persistence;
- a procedure concept with transactional semantics, i.e., serializability and recovery;
- abstraction from bulk iteration through associative access expressions.

In particular, we eliminated the competence and impedance mismatch between programming languages and database management systems by providing

orthogonal persistence, a uniform treatment of volatile and persistent data,
type completeness, a uniform treatment of large quantities of objects with a simple structure and small quantities of objects with a complex structure, as well as
strong static typing, a uniform (static) compatibility check between the declaration and the utilization of each name.

In implementing the DBPL system we adopted a rather strict approach by aiming for

- full orthogonality in our relationally extended type space, leading especially to nested relations;
- type-complete persistence, i.e. longevity of data from Booleans to relations;
- functional abstraction for relational expressions including recursion.

Without going into details it should be noted that naming (of statements, expressions etc.) naturally leads to the concept of recursion (see Fig. 2.1). The semantics of a recursive query expression in DBPL is not defined operationally (as is common practice for procedures) but as a least fixed point of recursive set equation [2, 5].

```
TYPE NumPairType = RELATION OF RECORD
                                Super, Sub : PartNumType END;

CONSTRUCTOR TransitiveMadeFrom: NumPairType;
BEGIN
  {p.Name, mf.Num} OF
     EACH p IN PartRel, EACH mf IN p.MadeFrom:
       p.State = comp,
  {mf1.Super, mf2.Sub} OF
    EACH mf1 IN {TransitiveMadeFrom},
    EACH mf2 IN {TransitiveMadeFrom}:
      mf1.Sub = mf2.Super
END TransitiveMadeFrom;
```

Fig. 2.1. Recursive query (access expression) in DBPL

3. DBPL System Architecture

In the full paper we report on experience gained during DBPL system implementation and refinement in

- removing conventional compiler implementation restrictions [20],
- extending the expressiveness of the bulk query language,
- generalizing the storage model, and
- reducing system performance bottlenecks resulting from mismatches between independently developed contributing technologies [4].

A repeating pattern in the development of the DBPL system was the need to generalize existing database solutions to meet the requirements imposed by the orthogonal DBPL language model.

DBPL was not the end of our efforts in supporting database programming. We embedded it into a network of DBPL-based spin-off projects ranging from distributed DBPL [8, 11] to DBPL-oriented programming environments (DAIDA) [3, 7, 24]. The Open DBPL/SQL gateway [13] and the bidirectional call interfaces between DBPL and other languages demonstrated that it requires a surprisingly small effort to embed a "closed" database programming language with a well-designed internal architecture into "open" system environments.

4. On Project Termination

Over the years, we encountered several data modeling concepts (inheritance, object identifiers, ordered bulk data structures, triggers, ...) and database system requirements (communication between concurrent processes, exception handling, ...) that did not fit nicely into the original DBPL language framework (see, for example, [1, 9]). We regard the monomorphic type system of Modula-2, the lack of first-class procedures, and the strict separation between built-in and user-defined system functionality in the DBPL system architecture as hard, built-in limitations which prevent, without violating our design principles, DBPL extensions beyond the realms of set- and predicate-based data models and first-order programming languages.

The DBPL project results in a mature product for modular database application programming and a follow-up project Tycoon (see Chapters 1.1.1 and 2.1.4) based on polymorphic types and other higher-order concepts. Thus the Tycoon project carries forward in a more general linguistic and architectural framework the experience gained with bulk data typing, iteration abstraction, and modularization in DBPL.

Acknowledgement This research was supported by ESPRIT Basic Research, #892 (DAIDA), #3070 (FIDE), #6309 (FIDE_2).

References

1. S. Böttcher, M. Jarke, and J.W. Schmidt. Adaptive predicate managers in database systems. In *Proceedings of the Twelfth International Conference on Very Large Databases, Kyoto, Japan*, 1986.
2. J. Eder, A. Rudloff, F. Matthes, and J.W. Schmidt. Data construction with recursive set expressions in DBPL. In *Proceedings of the Kiev East/West Workshop on Next Generation Database Technology*, volume 504 of *Lecture Notes in Computer Science*, April 1991.
3. M. Jarke. *Database Application Engineering with DAIDA*. Springer-Verlag, 1993.
4. M. Jarke and J. Koch. Range nesting: A fast method to evaluate quantified queries. In *Proceedings of the ACM-SIGMOD International Conference on Management of Data*, pages 196–206, May 1983.
5. M. Jarke, V. Linnemann, and J.W. Schmidt. Data constructors: On the integration of rules and relations. In *Proceedings of the Eleventh International Conference on Very Large Databases, Stockholm*, August 1985.
6. M. Jarke and J.W. Schmidt. Query processing strategies in the Pascal/R relational database management system. In *Proceedings of the ACM-SIGMOD International Conference on Management of Data, Orlando, Florida*, 1982.
7. M. Jeusfeld, M. Mertikas, I. Wetzel, Jarke. M., and J.W. Schmidt. Database application development as an object modelling activity. In *Proceedings of the Sixteenth International Conference on Very Large Databases, Brisbane, Australia*, August 1990.
8. W. Johannsen, L. Ge, W. Lamersdorf, K. Reinhard, and J.W. Schmidt. Database application support in open systems: Language support and implementation. In *Proceedings of the IEEE Fourth International Conference on Data Engineering, Los Angeles, California*, February 1988.

9. W. Johannsen, W. Lamersdorf, K. Reinhard, and J.W. Schmidt. The DURESS project: Extending databases into an open systems architecture. In *Proceedings of the First Conference on Extending Database Technology, EDBT'88*, volume 303 of *Lecture Notes in Computer Science*, pages 616–620. Springer-Verlag, 1988.
10. J. Koch, M. Mall, P. Putfarken, M. Reimer, J.W. Schmidt, and C.A. Zehnder. Modula/R report, lilith version. Technical report, Department Informatik, ETH Zürich, Switzerland, February 1983.
11. W. Lamersdorf, H. Eckhardt, W. Effelsberg, K. Reinhard, and J.W. Schmidt. Database programming for distributed office systems. In *Proc. Int. Conf. on Office Automation*, Washington, 1987.
12. M. Mall, M. Reimer, and J.W. Schmidt. Data selection, sharing and access control in a relational scenario. In M.L. Brodie, J.L. Myopoulos, and J.W. Schmidt, editors, *On Conceptual Modelling*. Springer-Verlag, 1984.
13. F. Matthes, A. Rudloff, J.W. Schmidt, and K. Subieta. A gateway from DBPL to Ingres. In T. Risch W. Litwin, editor, *Applications of Databases, First International Conference, ADB-94*, volume 819 of *Lecture Notes in Computer Science*, pages 365–380, Vadstena, Sweden, June 1994. Springer-Verlag.
14. F. Matthes and J.W. Schmidt. The type system of DBPL. In *Proceedings of the Second International Workshop on Database Programming Languages, Portland, Oregon*, pages 255–260, June 1989.
15. F. Matthes and J.W. Schmidt. DBPL: The system and its environment. In M. Jarke, editor, *Database Application Engineering with DAIDA*, pages 319–348. Springer-Verlag, 1993.
16. J.W. Schmidt. Some high level language constructs for data of type relation. In *Proceedings of the ACM-SIGMOD International Conference on Management of Data, Toronto, Canada*, August 1977. (Also appeared in ACM TODS, 2(3), September, 1977 and A. Wasserman (editor), IEEE Tutorial on Programming Language Design, and M. Stonebreaker (editor), Readings in Database Systems, Morgan Kaufmann Publishers, 1988 and 1993).
17. J.W. Schmidt. Type concepts for database definition. In B. Shneiderman, editor, *Databases: Improving Usability and Responsiveness*. Academic Press, New York and London, 1978.
18. J.W. Schmidt and M. Mall. Pascal/R report. Bericht 66, Fachbereich Informatik, Universität Hamburg, Germany, January 1980.
19. J.W. Schmidt and M. Mall. Abstraction mechanisms for database programming. In *Proc. SIGPLAN Symp. on Programming Language Issues in Software Systems*, San Francisco, June 1983.
20. J.W. Schmidt and F. Matthes. Naming schemes and name space management in the DBPL persistent storage system. In *Proceedings of the Fourth International Workshop on Persistent Object Systems, Martha's Vineyard, Massachusetts*. Morgan Kaufmann Publishers, January 1991.
21. J.W. Schmidt and F. Matthes. The rationale behind DBPL. In *3rd Symposium on Mathematical Fundamentals of Database and Knowledge Base Systems*, volume 495 of *Lecture Notes in Computer Science*. Springer-Verlag, May 1991.
22. J.W. Schmidt and F. Matthes. Modular and rule-based database programming in DBPL. In M. Jarke, editor, *Database Application Engineering with DAIDA*, pages 85–124. Springer-Verlag, 1993.
23. J.W. Schmidt and F. Matthes. The DBPL project: Advances in modular database programming. *Information Systems*, 19(2):121–140, 1994.
24. J.W. Schmidt, I. Wetzel, A. Borgida, and J. Mylopoulos. Database programming by formal refinement of conceptual designs. *IEEE – Data Engineering*, September 1989.