Part 2
Persistent Support Systems: Introduction and Overview

Florian Matthes

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

The goal of the persistent support systems developed in FIDE is to improve the construction, maintenance and operation of *persistent application systems* (PAS). A PAS is a software system that gives its users a flexible, problem-oriented and safe access to large sets of long-lived objects of application-specific types.

Due to technological and commercial developments there is an ever-increasing demand for persistent application systems that have to handle *non-standard* object types, such as texts (electronic mail, text retrieval, "intelligent" text manipulation), two- and three-dimensional graphics (CAD and geographic information systems), digitized raster images (picture and font databases, fax manager, character recognition software), voice data, or even image sequences (presentation software, scene analysis). Moreover, persistent application systems have to be implemented also in PC-based, interactive, networked, distributed, or semi-autonomous system environments. As a consequence of their longevity, PAS have to survive even under drastic changes of their underlying hardware and software platforms. As it is widely recognised today [4, 8, 12, 1, 11], classical database technology provides insufficient system support for these new object types and changing system environments.

In addition to the classical problems of large-scale application development, developers of a PAS are faced with three PAS-specific requirements:

Generic Programming and Data Modelling: Persistent application systems are characterised by highly repetitive algorithmic and structural patterns (set-oriented queries, integrity constraints, generic data structures, etc.). Therefore, many commercial tools utilise generator techniques to derive automatically stereotyped program fragments (form definitions, format conversions, sorting routines, iterators, etc.) from parameterised, high-level descriptions. But by no means all of their operations fit these regular patterns, and therefore, it is necessary to facilitate their use from general purpose languages, such as those presented in Part 1 of this book, as well.

Persistent Storage: The primary task in a persistent application system is the manipulation of the state of long-lived data objects shared by multiple, possibly concurrently executing applications. In advanced persistent application systems it is advantageous to be able to store, share

and modify also program objects ("methods", "scripts") and programs in the process of being executed ("workflows", "activities", "processes") as first-class data objects.

External Server Integration: The buzzword "open architectures" is of particular relevance for persistent application systems since it is often necessary to interact with external devices (screens, printers, keyboards, pointing devices, scanners, CD-ROMs, modems, fax machines, etc.) and external servers (window libraries, RPC libraries, directory services, etc.) for data acquisition, data presentation, data transfer and data integration purposes.

In the following, we first sketch how these requirements are met partially in third- and fourth-generation languages, and we then outline FIDE's contributions to PAS construction in comparison with these more traditional approaches.

Low-Level Server Access from Third-Generation Languages

The majority of today's information systems are implemented in third-generation languages (COBOL, FORTRAN, C, Pascal). A severe limitation of these languages is their inability to capture correctly the genericity inherent even in simple persistent object models. For example, the relational data model supports the definition of unary, binary, ternary, ... relations over various domains and it includes generic operations over relation variables. These operations cannot be captured adequately by any of the commercial programming languages mentioned above [3].

As a consequence of this conceptual limitation, developers of database programming interfaces like Embedded SQL [7] restrict themselves to the least common denominator of all 3GLs and handle any information interchange via strings, addresses and untyped byte arrays that are transferred at the smallest possible granularity (attribute-wise) between application programs and the database systems. A similar situation can be observed at the interface between programming languages and (graphical) user interfaces.

The resulting POS scenario using 3GLs is depicted in Fig. 1: The overall functionality of a particular POS is provided by a coordinated manipulation of numerous heterogeneous objects on different media (D: database objects, P: program objects, S: screen objects, etc.). Objects on different servers are subject to server-specific naming, typing, lifetime, and binding rules (indicated by different line styles in Fig. 1) while inter-server communication takes place at a low level of abstraction using simple, machine-oriented protocols.

A crucial advantage of such a loose server coupling is the *openness* of the application development environment. If required, new servers can be integrated freely as libraries into existing persistent object systems. Furthermore,

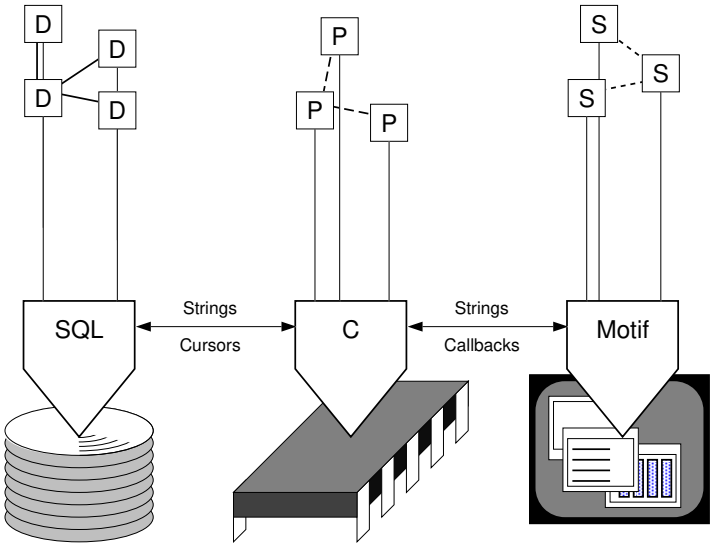

Fig. 1. Narrow and unsafe server interfaces in third-generation languages

call interfaces provide a minimal common ground for data exchange between independently developed services and they also constitute a starting point for data abstraction and component exchangeability.

Server Integration in Fourth-Generation Languages

Fourth-generation languages recognise the need for a uniform, safe and application-oriented development environment across a variety of servers and above the minimal interoperation level of strings and cursors. As depicted in Fig. 2, a well-designed 4GL provides high-level naming, typing and binding mechanisms for objects on different media (on disk, in memory, and on screen) and thereby eliminates a substantial amount of the low-level, repetitive data-conversion and data-transfer code found in 3GL applications. 4GLs are tailored to the specific requirements of traditional, commercial database applications:

Functional requirements: record- and set-oriented data modeling, data manipulation, data storage and data display; generic operations (search, iterate, insert, delete, update, format, display, validate, ...) applicable uniformly to record and set objects of different component structures.

Operational requirements: persistent storage of data, application code, integrity constraints and user-interface definitions; efficient manipulation

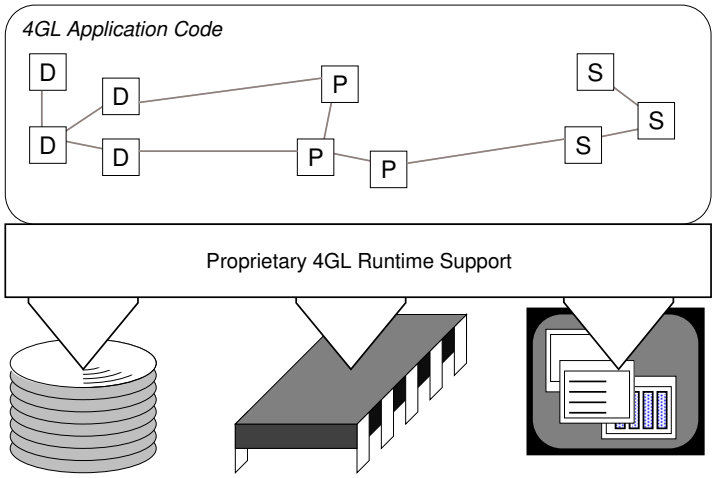

Fig. 2. Server integration in proprietary fourth-generation languages

of data on secondary storage; integration into transactional multi-user systems.

Modal requirements: interactive user-interfaces, access to precompiled queries and update transactions; support for incremental system evolution (e.g. schema changes); provision for *ad-hoc* queries.

Examples of relational 4GLs include Ingres-4GL [6], PL-SQL [10], Visual Basic for Microsoft Access and Informix-4GL [5]. Furthermore, several object-oriented database systems are shipped with 4GL languages to simplify application development (e.g., O_2SQL and CO_2 for O_2 [2]). Several client-server programming languages like Visual Basic or Power Builder can also be classified as 4GLs. Practical experience shows that 4GL constitute a significant advance towards achieving faster software development, improved long-term software maintainability, better software portability, and even enhanced execution efficiency for specialised applications (e.g., queries) when compared with hand-coded third-generation language solutions such as written in C or COBOL [9].

There are, however, some severe limitations to current 4GL languages that hamper their success in today's open, continually evolving system environments: 4GLs are deeply integrated into proprietary DBMSs and it is therefore impossible to move 4GL applications from one DBMS to another. This situation is in sharp contrast to 3GL applications using standardised programming and database languages (e.g., C and OpenSQL). Moreover, applications developed in 4GLs do not scale well due to a lack of appropriate typing and modularisation support such as that found in Ada, C++, Modula-2 or Eiffel. On the system level, 4GL often incur an unnecessary system overhead since it

is impossible to "downscale" the extensive 4GL runtime support (form management, event management, query support, report generation, transaction management, ...) to just the functionality needed by a particular application. When a problem does not match the patterns of computation captured by the 4GL models, the cost of coding the solution can rise very rapidly.

Support Systems for Persistent Higher-Order Languages

Most of the above 4GL limitations reflect the deficiencies of the commercially available persistent system technology. This technology simply does not support the abstractions required for a more flexible, systematic 4GL service integration.

To overcome these limitations and to achieve 4GL openness and extensibility, FIDE undertook the integrated effort of not only developing higher-order languages with appropriate abstraction capabilities (see Part 1), but also of designing persistent system architectures (Chapter 2.1), realising object store managers (Chapter 2.2) and experimenting with novel implementation techniques (Chapter 2.3).

The FIDE system support architectures and their implementation techniques are influenced heavily by *language concepts* which are described in more detail in Part 1. For example, orthogonal persistence requires a uniform and highly efficient management of volatile and persistent data, higher-orderness requires a uniform persistent storage of data, code and threads, genericity calls for polymorphic value representations, and reflection leads to a tight integration of the PAS development and execution environment.

Fig. 3 sketches the architecture of a PAS implemented in a persistent higher-order language. Similar to a 4GL, the relevant application objects (persistent database objects, volatile program objects, virtual screen objects, etc.) are subject to uniform naming, typing and binding rules, indicated by uniform line styles connecting the corresponding symbols in Fig. 3. In contrast to a 4GL, most of these object types are not built into the base language but are provided by an open set of libraries that are an integral part of the programming environment. Some of the libraries are simply interfaces to external services (database systems, language compilers, GUI toolkits), other libraries provide functionality that is implemented in the persistent higher-order language itself (see Chapter 3.1.4). Furthermore, Fig. 3 indicates that access to some of the libraries may be supported by tailored syntactic forms provided by the persistent languages or by application-specific syntax extensions.

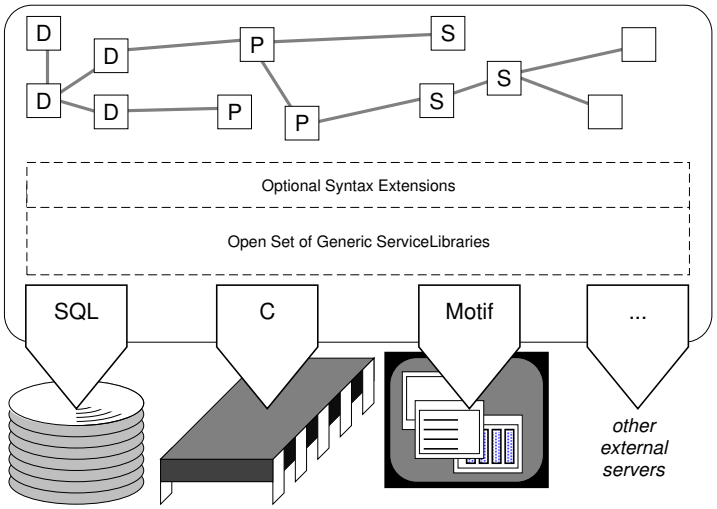

Fig. 3. Scalable application frameworks in persistent higher-order languages

References

1. M. Atkinson, F. Bancilhon, D. De Witt, K. Dittrich, D. Maier, and S. Zdonik. The object-oriented database system manifesto. In *Deductive and Object-oriented Databases*. Elsevier Science Publishers, Amsterdam, Netherlands, 1990.
2. F. Bancilhon, C. Delobel, and P. Kanellakis. *Building an Object-Oriented Database System: The Story of O_2*. Morgan Kaufmann Publishers, 1992.
3. P. Bunemann, J. Hirschberg, and D. Root. A Codasyl interface to Pascal and Ada. In *Proceedings of the Second British National Conference on Databases*. Cambridge University Press, 1982.
4. R.G.G. Cattell. Next-generation database systems. *Communications of the ACM*, 34(10), October 1991.
5. Informix Software Corp. Informix-4GL reference manual. Technical report, Informix Software Corp., 1986.
6. Ingres Corporation. INGRES ABF/4GL reference manual for the UNIX and VMS operating systems. Technical Report INGRES Release 6.3, Ingres Corporation, 1080 Marina Village Parkway, Almeda, California, November 1989.
7. Ingres Corporation. INGRES embedded SQL companion guide for C. Technical Report INGRES UNIX Release 6.3, Ingres Corporation, 1080 Marina Village Parkway, Almeda, CA 94501, December 1990.
8. F. Manola, S. Heiler, D. Georgakopoulos, M. Hornick, and M. Brodie. Distributed object management. *International Journal of Intelligent and Cooperative Information Systems*, 1(1), March 1992.
9. V.M. Matos and P.J. Jalics. An experimental analysis of the performance of 4GL tools on PCs. *Communications of the ACM*, 32(11):1340–1352, 1989.
10. Oracle Corporation. PL/SQL user's guide and reference, version 1.0. Technical Report Part No. 800-V1.0, Oracle Corporation, June 1991.

11. M. Stonebraker, L.A. Rowe, B. Lindsay, J. Gray, M. Carey, M. Brodie, and P. Bernstein. Third-generation data base system manifesto. *ACM SIGMOD Record*, 19, September 1990.
12. S.B. Zdonik and D. Maier. *Readings in Object Oriented Database Management Systems.* Morgan Kaufmann Publishers, 1989.