10
DBPL: The System and its Environment

Florian Matthes and Joachim W. Schmidt

ABSTRACT  *The DBPL system and its environment support the modular implementation of advanced data-intensive applications based on integrated database and programming language technology. It provides state-of-the-art system support at application compile-, link- and run-time as well as tools for incremental application evolution.*

10.1 System Support for Data-Intensive Applications

Databases have proven to be the keystones for the majority of application systems with a wider functionality, utilization and availability. Therefore, the development of integrated language and system support for the definition, execution and maintenance of data-intensive applications gained substantial interest over the past years [BJM+89, BMSW89, DT88, JMW+90, AB87]. Essentially, this support concentrates on three major goals:

- improved data integrity;
- enhanced program efficiency;
- increased user productivity.

As expected, substantial progress towards these goals requires contributions from several key areas of computer science, in particular from

- language and compiler technology;
- optimization strategies in searching, computing and scheduling; and
- formal specification schemes for relevant properties of data and programs.

The database programming language project DBPL approaches the above goals essentially by the

- linguistic quality of the DBPL interfaces for data definition and manipulation;
- distinctiveness of the DBPL system implementation; and
- richness of the DBPL environment.
It turns out to be crucial for data-intensive applications to distribute the supported tasks appropriately over time. While application design time profits from the availability of powerful abstraction mechanisms for data and transaction definition, compile time support is essential for (partial) correctness and consistency, proper error reporting, interface checking and certain classes of optimizations. Run time support is vital for global efficiency within and among transactions as well as for long-term data integrity. Finally, the extremely long lifetime of most data-intensive applications requires permanent support for incremental data and program modification and partial redesign.

While the DBPL language is discussed in chapter 4 this chapter concentrates in section 10.2 on the presentation of DBPL’s compile time support and in section 10.3 on the DBPL run time system. Section 10.4 describes how advanced incremental compilation technology can be exploited for long-term program and system evolution in a tightly coupled interactive programming environment. Further details of the system components are described in DBPL and DAIDA technical reports (e.g. [SEM88b, NS87b] and chapter 12). The chapter closes with an outlook on current research in next-generation DBPL environments.

As of today, the DBPL system exists in two fully source code compatible implementations, VAX/VMS DBPL and Sun DBPL. Both are written entirely in Modula-2 and consist of a compilation and a run time environment. The run time system is highly portable and runs under VAX/VMS 6.1, SunOS 4.1, IBM AIX 3.1 and IBM OS/2. All operating system dependend code is factored out into four modules for main memory management, block-oriented file input and output, exception handling and error message handling. The DBPL compilers generate native code for VAX, respectively Sparc, Motorola and Intel architectures. Both DBPL systems can be integrated deeply into the DAIDA environment and into commercially available software development environments (NSE [Cou89], SCCS, CMS).

Figure 10.1 depicts the overall DBPL system architecture. The left hand side of Fig. 10.1 sketches how a DBPL program module is first translated by the DBPL compiler then linked with other compiled DBPL modules yielding executable object code. This object code interacts through the interface module DBPLRTS with the various layers of the DBPL run time system (PSMS, CTMS, CPMS, CRDS, SMS; see Sec. 10.3). Database objects are stored in files accessed through operating system calls issued by the layer SMS.

Fig. 10.1 also outlines the interaction between two DBPL applications running against shared databases. This sharing is achieved by importing the same database module(s) into two different programs. Concurrent access to database objects requires a synchronization that is achieved by explicit message passing at three layers of the DBPL system (CTMS, CRDS, SMS) via LMS services. A centralized scheduling process guarantees serializable transaction execution for DBPL