Type-Safe FIDE / SQL Server Interfaces

Final FIDE Review Meeting, Hamburg

Florian Matthes, Andreas Rudloff, Joachim W. Schmidt, Kazimierz Subieta
Hamburg University

September 1992

Objectives of the Demo

The purpose of this demonstration is to show how existing commercial relational database management systems can be deeply integrated into type-safe FIDE language environments. This integration is of high relevance for the commercial exploitation of FIDE technology since it provides the basis for a smooth migration path from todays system solutions to new FIDE services. On the other hand, the integration process itself poses challenging demands on FIDE languages and their supporting system architectures.

The demonstration covers two conceptually complementary approaches to server integration:

DBPL SQL Gateway: A built-in, transparent approach to server integration

Tycoon Library: An add-on, extensible approach to server integration

The functionality of the DBPL gateway is demonstrated using a DBPL application that maintains a large on-line module and identifier database which is identical to the stand-alone DBPL application demonstrated at the previous FIDE review.

The underlying SQL database is also used for the demonstration of the Tycoon Library.

The DBPL SQL Gateway: An Overview

During the last year, the DBPL system available on Sun/SunOS and VAX/VMS platforms was extended by a set-oriented gateway to external relational databases (Ingres 6.3). It provides a fully transparent access to external (pre-existing or dynamically created) SQL databases in addition to type-complete DBPL-specific complex object databases.

The gateway preserves the linguistic qualities of DBPL like

- strong typing
- static binding
- user-defined data types
- application modularization
- generalized iteration abstractions
- fixed-point queries
- transactional procedures

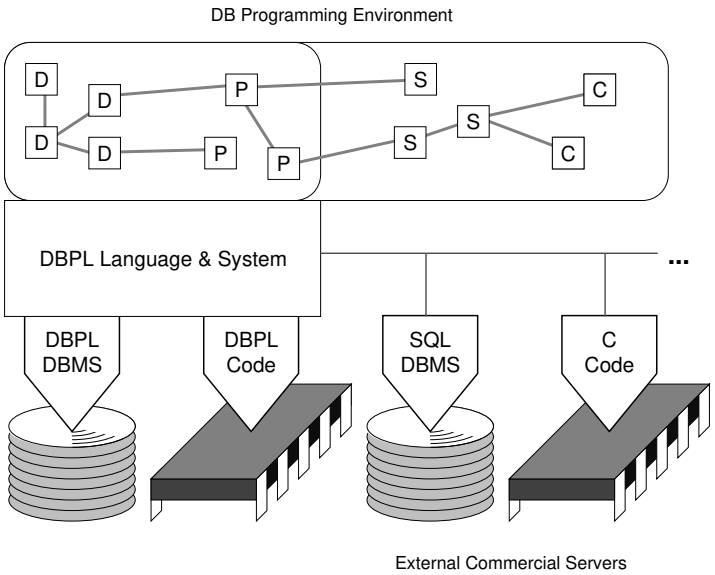
while exploiting the services of todays set-oriented storage servers

- extensive access optimization
- access protection
- concurrency control
- crash recovery
- ad-hoc query interfaces, form interfaces, ...

The gateway is responsible for

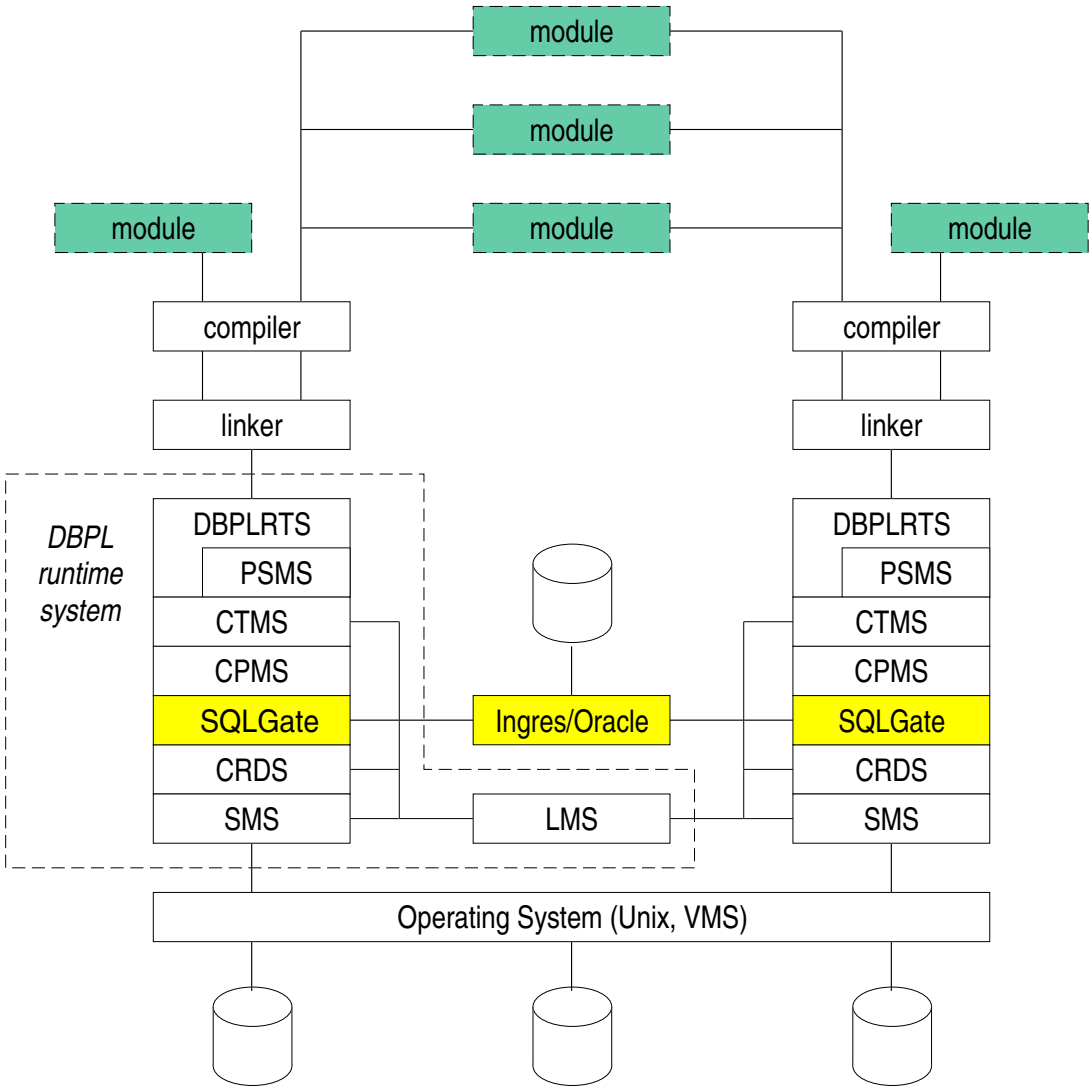
- data type conversions
- query transformations (e.g. universal quantification, parameterization)
- host variable and cursor handling (e.g. in recursive procedures)
- evaluation of "mixed" queries (involving DBPL and SQL objects)
- multi-server transaction management (integration of DBPL and SQL concurrency control and recovery)

A Built-In Approach to Server Integration



- Seamless integration of *external* and *internal* services
- Localized DBPL runtime system extensions
- No need for DBPL language extensions
- Type orthogonality restrictions on external databases
  - Persistence only for relation variables
  - No support for non-flat relations
  - Restrictions on the maximum tuple size

Implementation Overview



Extensions to the predicate management system (CPMS):

- 1600 lines dynamic SQL/C code
- 5400 lines Modula-2 code
- 1000 lines revised Modula-2 code
- $\Rightarrow$ 10% of the total DBPL system code
- 12 man-months
- SQL Sever: Ingres 6.3 (Oracle 6.0)
- Available on Sun-3, Sun-4, VAX (and IBM RISC) platforms

Example Application

Task Maintenance of an online module cross reference database that stores information about

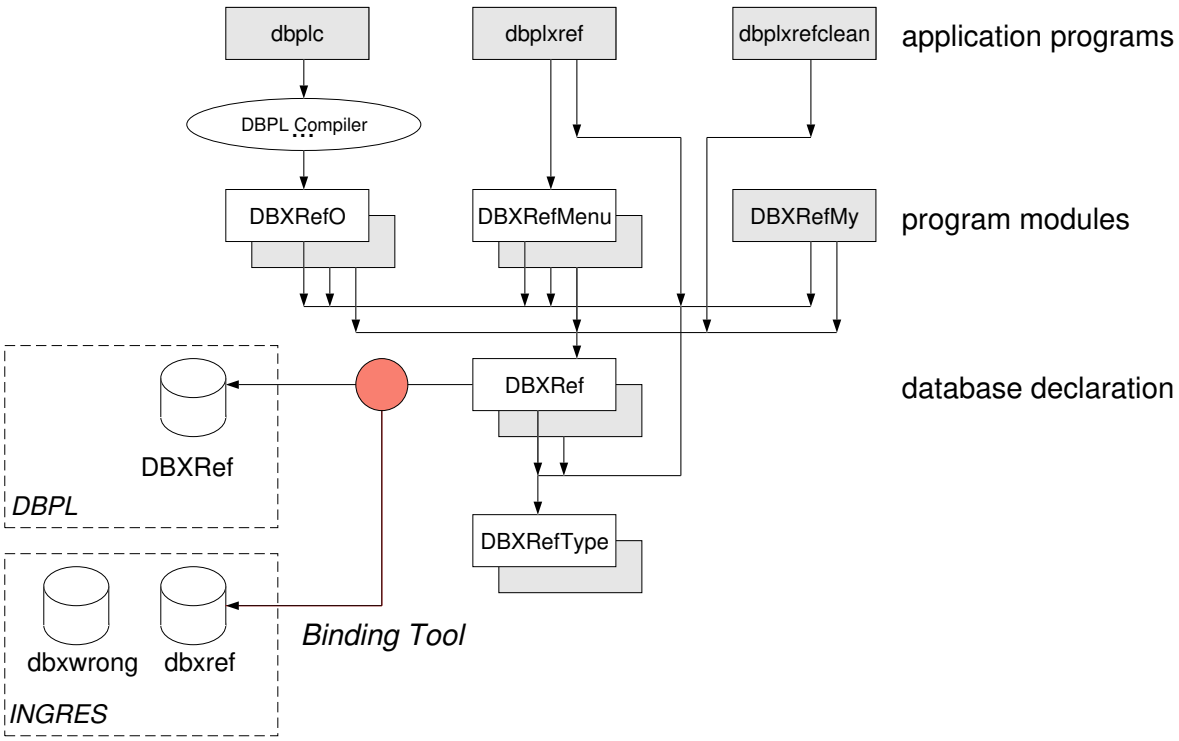
- compiled DBPL module definitions, module implementations and application programs;
- identifiers declared in module definitions (transactions, procedures, types, database variables, etc.);
- module import dependencies;
- identifier dependencies (e.g., variables depend on their types, procedures depend on their parameter types).

Test Data 5 tables

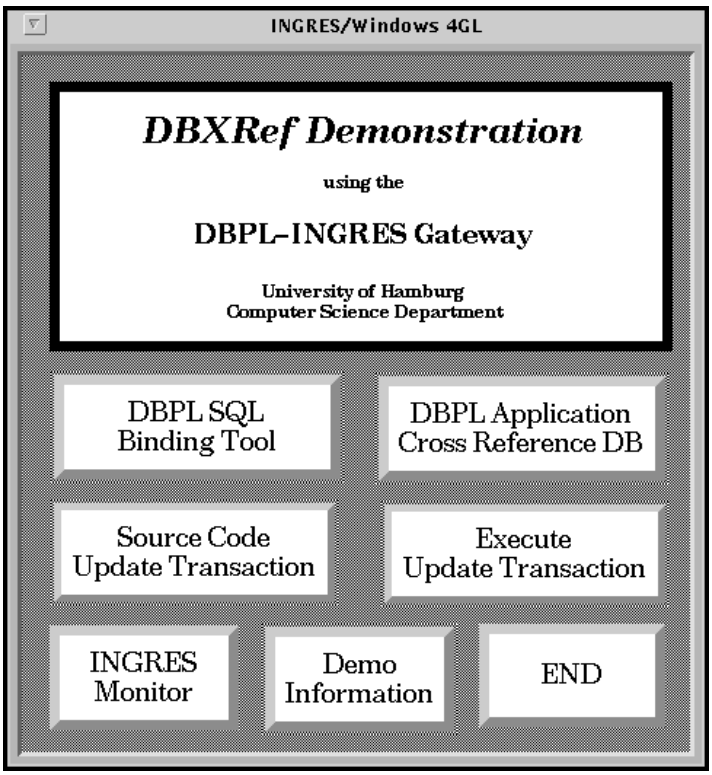
- 223 modules
- 3509 identifiers
- 5135 direct declaration dependencies
- 43954 identifier occurences

Approach A modular DBPL application program consisting of

- a global SQL database: module *DBXRef* importing *DBXRefTypes*
- a procedural interface to the DBPL compiler: *DBXRefO*
- a menu-driven text window interface for browsing: *DBXRefMenu*
- two main programs: *dbplxref*, *dbplxrefclean*

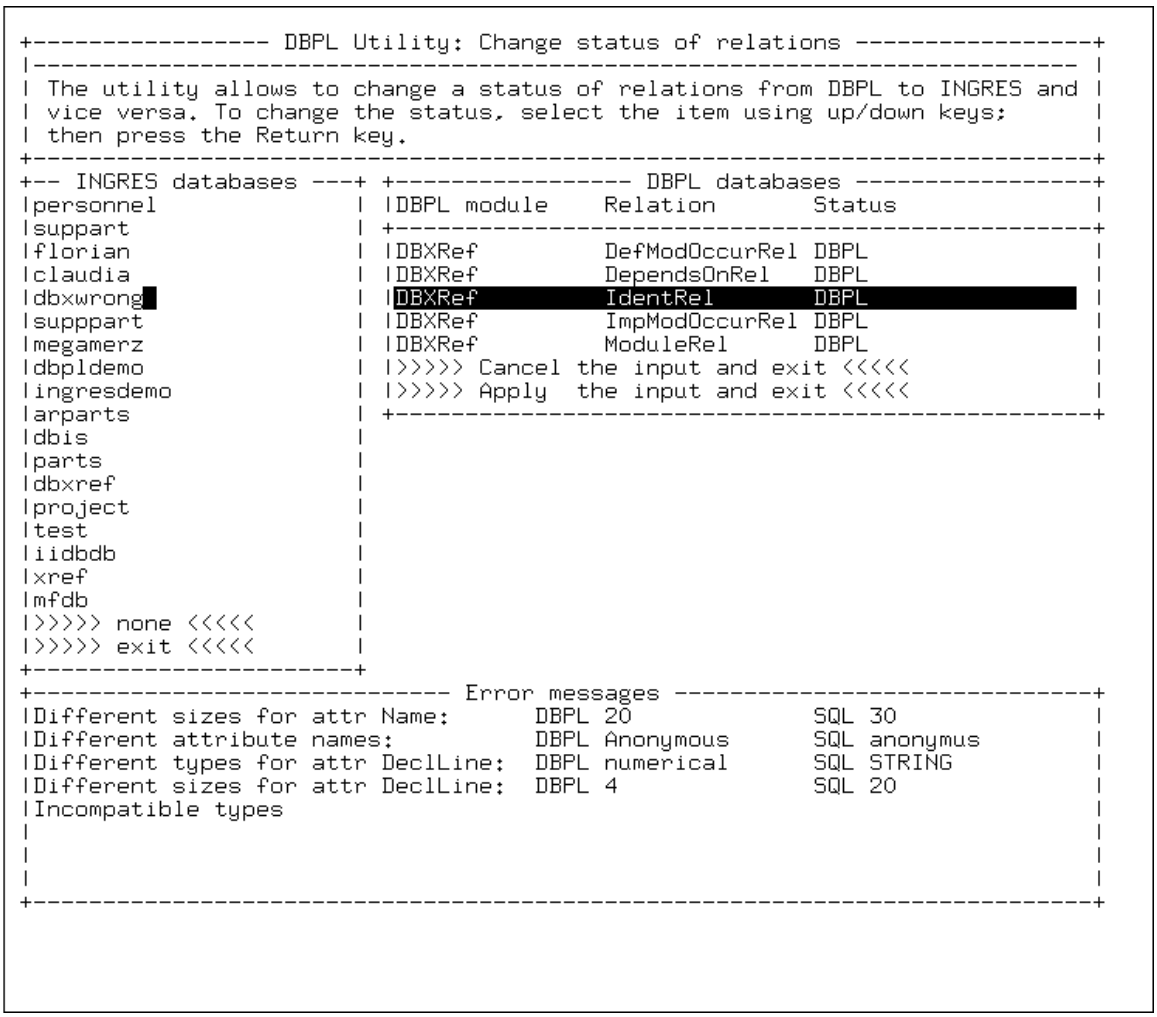


DBPL / SQL Demo Menu



- Integration of DBPL and Ingres tools (Ingres 4GL)
- Interactive users
- DBPL applications
- Standard ESQL applications

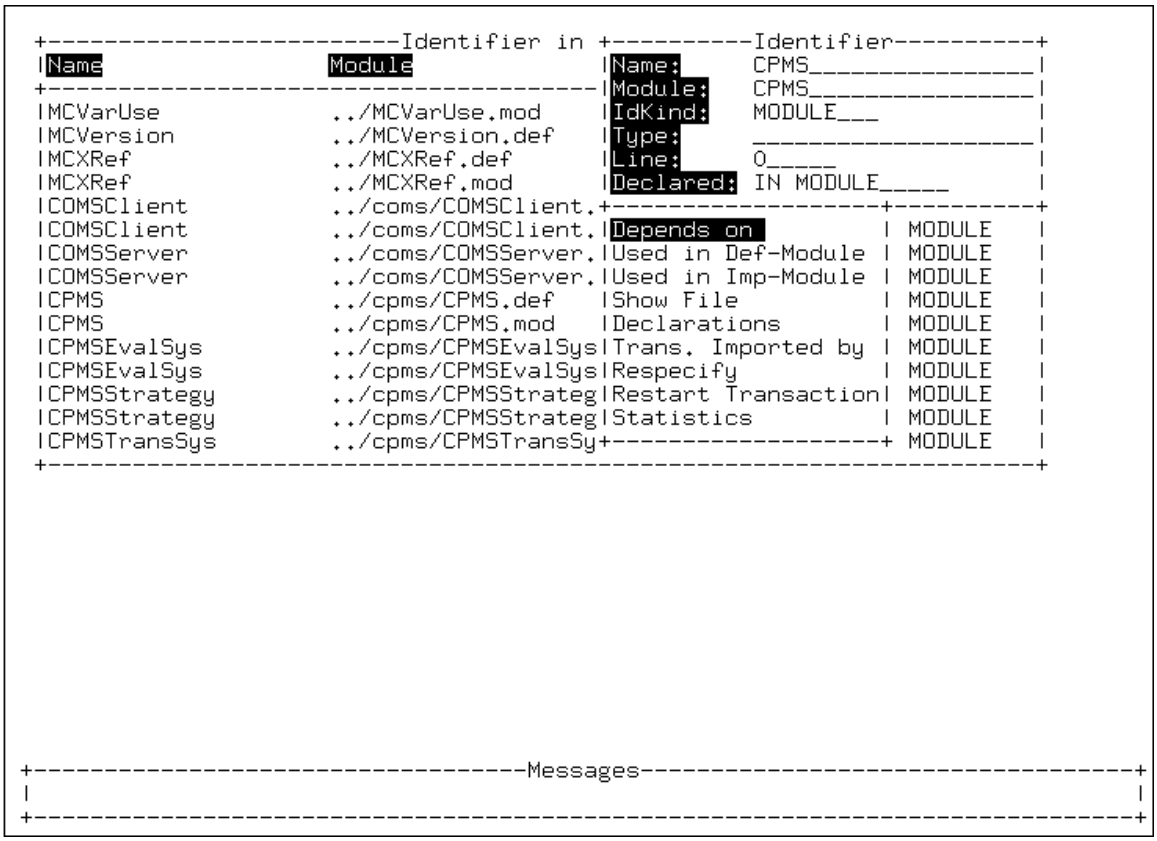
Dynamic Binding of SQL Databases



- Strong type compatibility check (e.g. null values) between
  - DBPL database schema (*DATABASE DEFINITION MODULE*)
  - Ingres database schema
- Data type conversion (e.g. enumeration types, subrange types)
- Detailed data format control (e.g. string sizes)
- Helpful error messages

Accessing SQL Databases from Compiled DBPL Programs

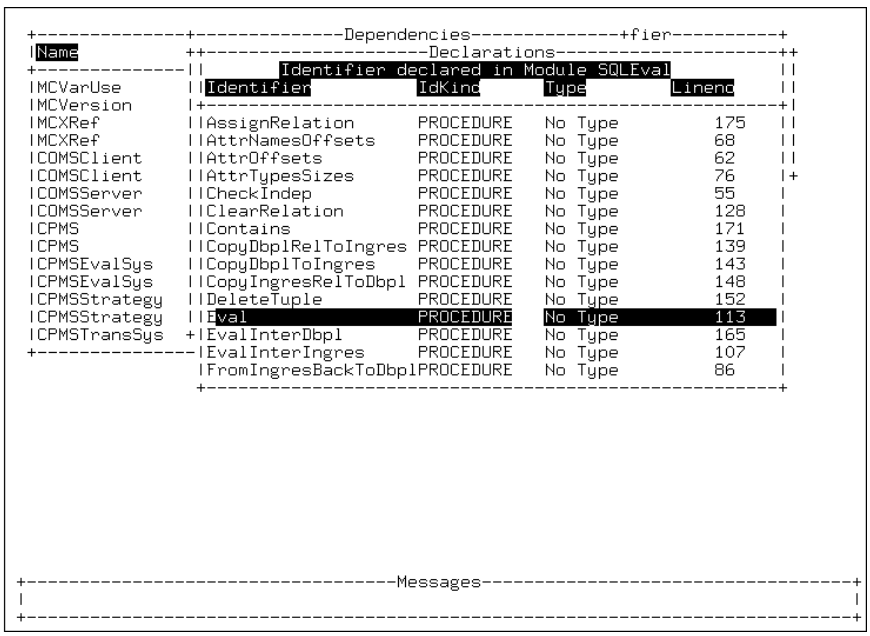
- Display all module identifiers in the database
- Select module CPMS
  (an implementation module of the DBPL runtime system, see page 4)

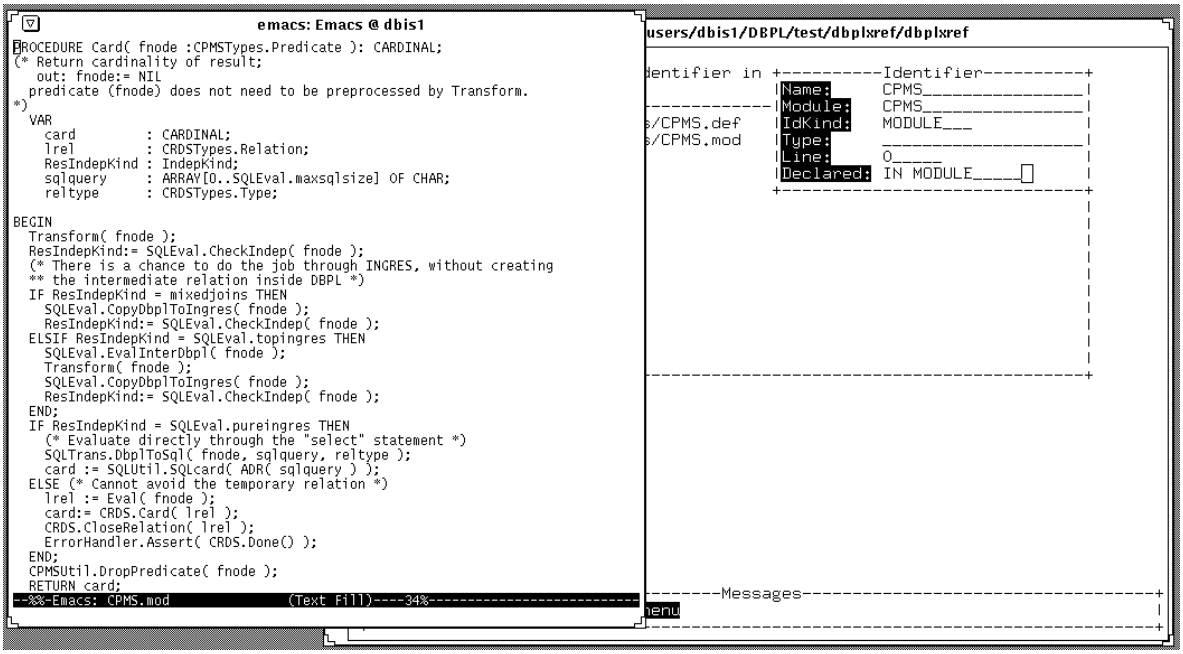


- *Canned* parameterized queries
- Expressions involving external SQL and temporary DBPL relations
- Stack of intermediate results
- Generic DBPL form interface

Navigation through the Cross Reference Database

- List the modules imported by module *CPMS*
- List all declarations exported by module *SQLEval*
- Display the source code of a declaration





Implementation Aspects

- Associative DBPL queries transparently translated into SQL queries:

```
DepRel := DepRelT{{i.Name, id.Name, id.Kind, i.ID, id.ID} OF
            EACH i IN IdentRel,
              EACH id IN IdentRel,
                EACH dep IN DependsOnRel:
                    (dep.IdentID = loc.Id) AND
                    (i.ID = dep.IdentID) AND
                    (id.ID = dep.DependsOnID)};
```

- Query execution plan based on case analysis:

  Pure DBPL, pure Ingres, top DBPL, top Ingres, mixed join, badly mixed

- Global optimizer, local DBPL query optimizer, local SQL query optimizer

- Selectors, constructors, recursive queries implemented on top of set expressions

  $\Rightarrow$ Recursive SQL queries

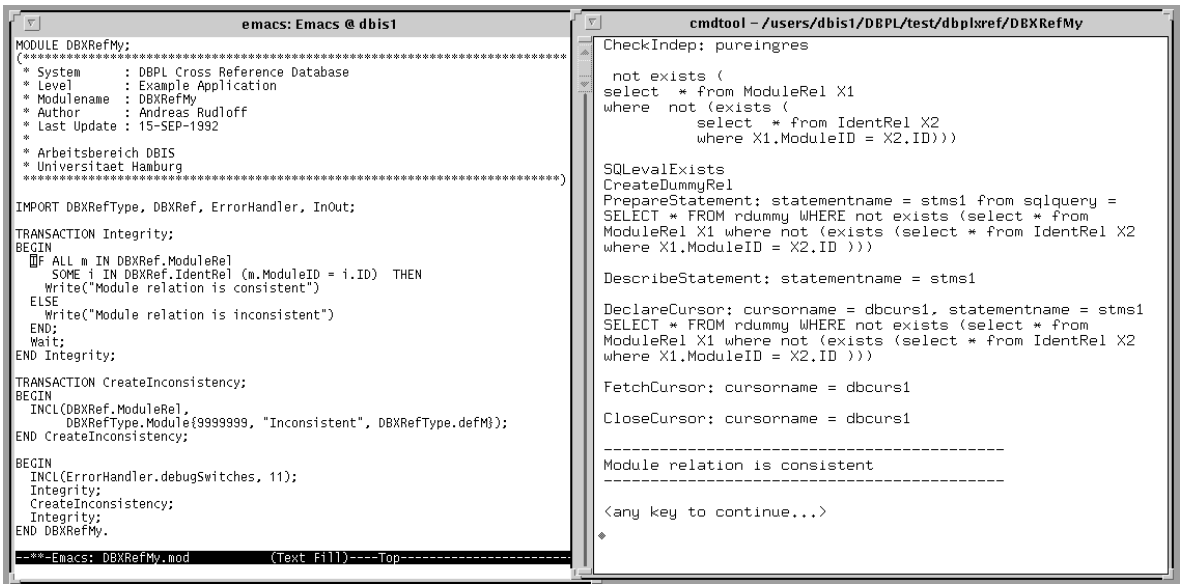
```
CONSTRUCTOR NextModules WITH (root:ID): IdRelT;
BEGIN
  root,
  d.IdentID OF EACH d IN DependsOnRel:
    SOME s IN IdRelT{NextModules(root)} (d.DependsOnID = s)
END NextModule;
```

For a given parameter (e.g. *30*), the application of the constructor defines the set of all keys of modules that transitively depend on the module with the key *30*.

```
modules:= IdRelT{NextModule(30)};
```

Query Translation and Concurrency Control

The following stand-alone DBPL program is executed against the cross-reference database. All data exchanged between the DBPL runtime system and the SQL server is displayed on the screen.



- Translation of universal quantifiers
- Non-set expressions
- Update statements
- Concurrency control, recovery

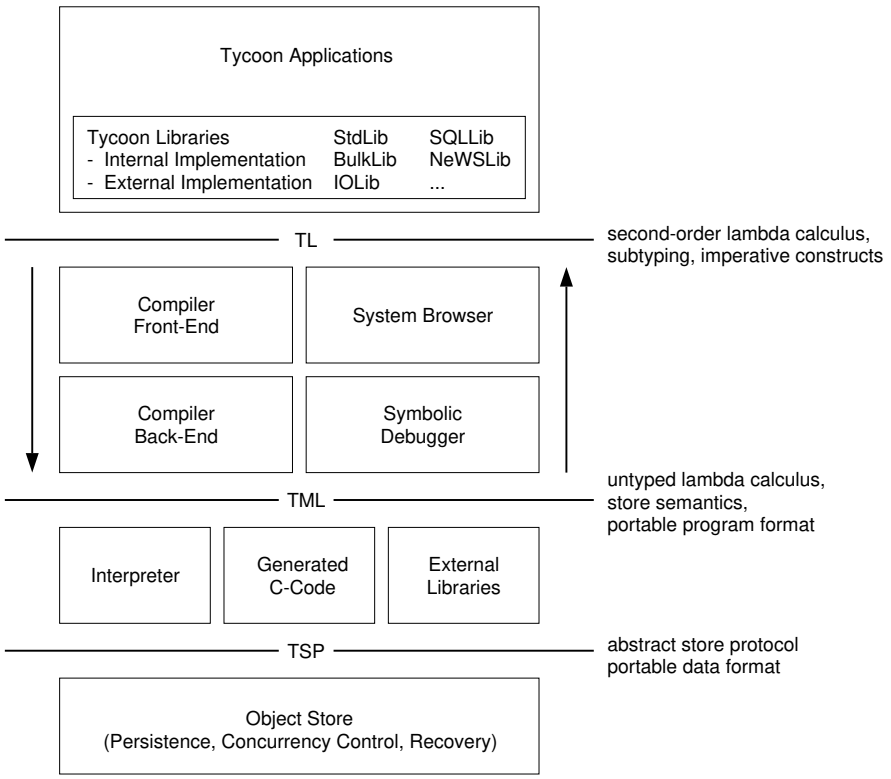
An Add-On Approach to Server Integration

Tycoon (TYped Communicating Objects in Open eNvironments):

A uniform linguistic and architectural framework for the integration of *generic* database services:

- Bulk data storage
- Data manipulation
- Graphical display
- Network communication
- ...

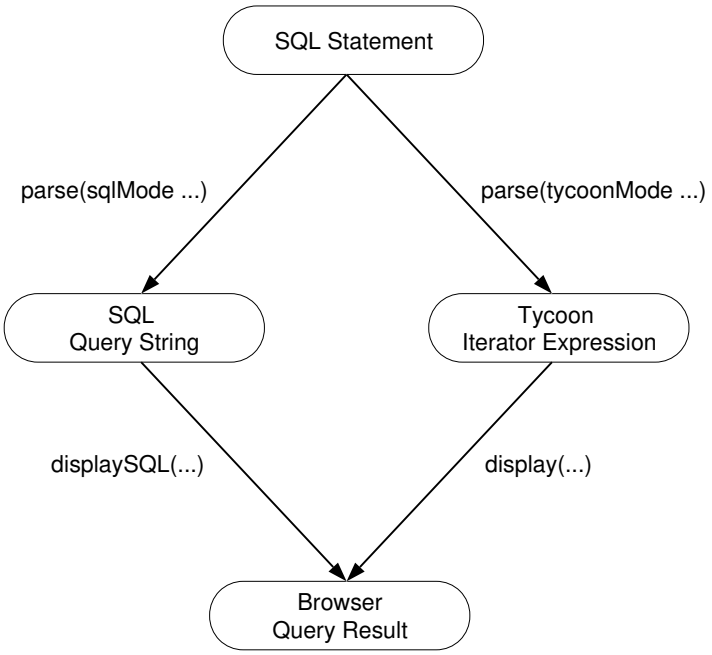
- Provide extensive library support for data-intensive applications
- Exploit existing commercial servers
- High-level application and system programming framework (TL)
- Scalable system architecture



An Exercise in Interoperability

Pre-existing, independently developed languages and abstractions

- SQL syntax
- Iterators in the Tycoon libraries
- SQL server (cursors, statements, buffer descriptors, ...)
- Tycoon bulk data types
- NeWS window server



Subasks

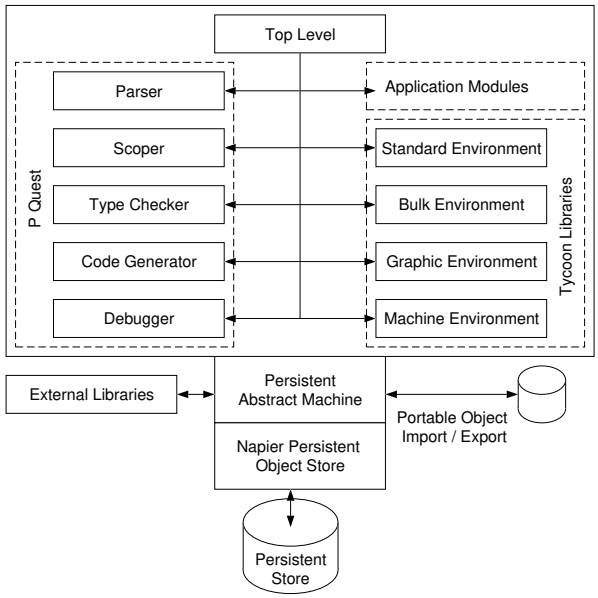
- Scanning, parsing, error reporting
- Expression analysis: scoping, semantic checking
- Expression translation
- Data browsing

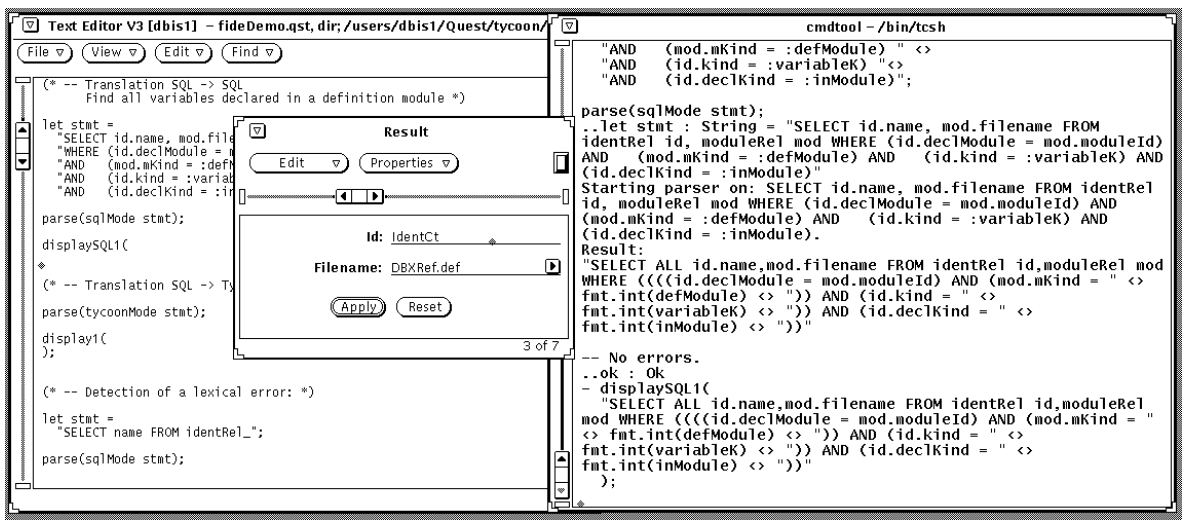
General DBPL issues

- Syntactic support
- Genericity
- Reusability of system components
- Ease of implementation

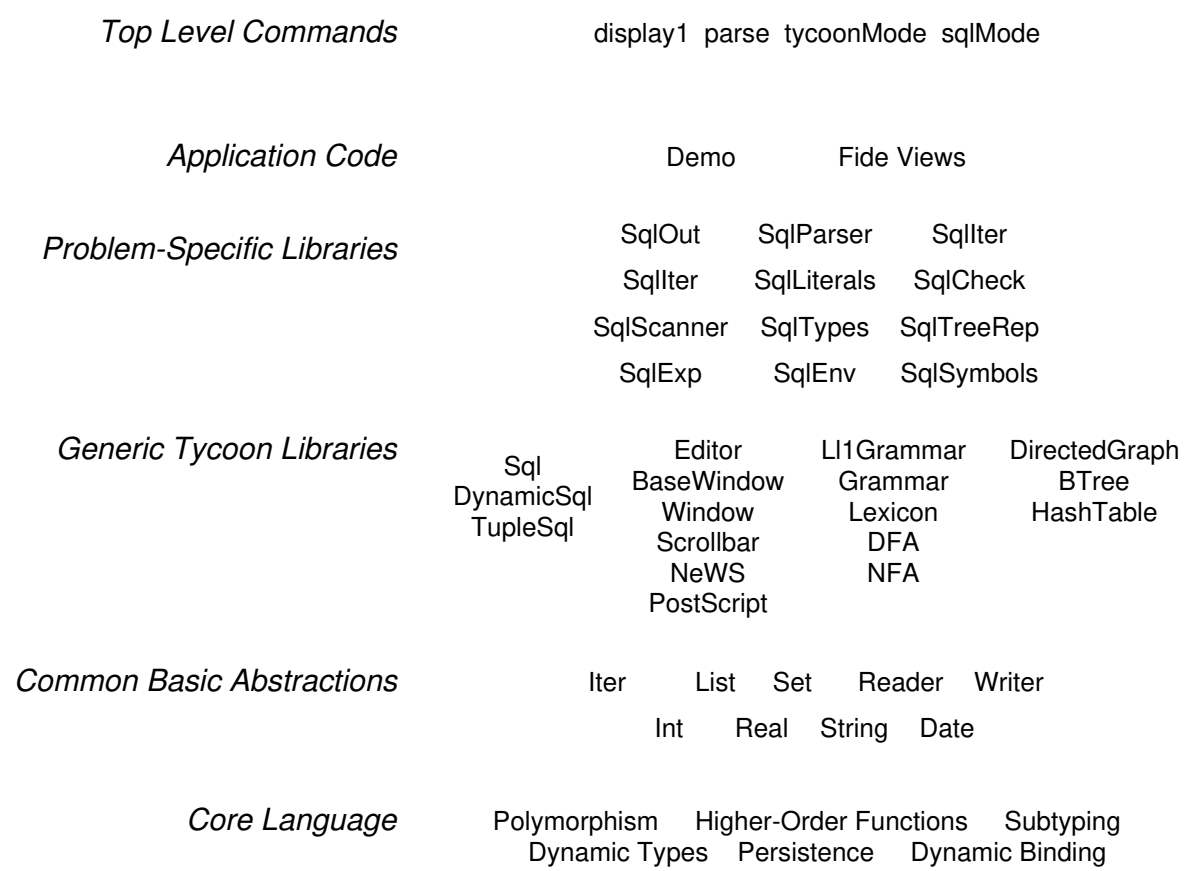
Demonstration Environment

- Napier persistent object store (garbage collection, stability)
- P-Quest language processors
- Generic Tycoon libraries (> 200 modules)
- Ingres 6.3
- NeWS/X11 window server



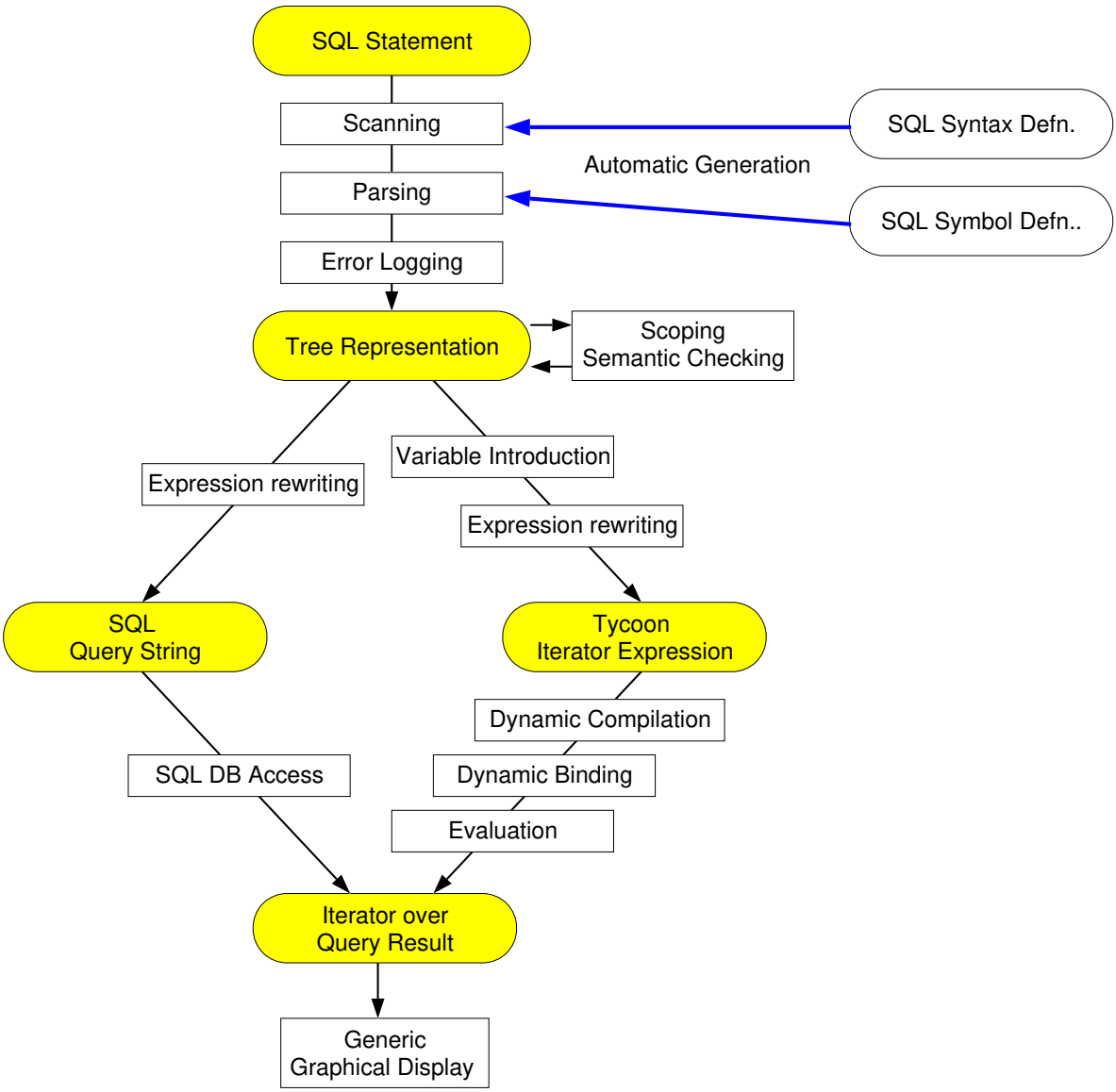


Special-Purpose vs. Generic Solutions



- Smooth transition from system to application programming
- Extensibility at various levels of abstraction
- Syntactic extensibility to support "language specializations"
  - Application-domain-specific abstractions (e.g. grammars, forms)
  - Application-specific abstractions (e.g. query operators, database objects)
- High demands on language genericity and binding flexibility

The Modular Implementation



FIDE Technology

- Reflection
- Language processing in persistent environments
- Bulk type generalisation
- Type-safe external interfaces