Overview of the ODRA project

Krzysztof Stencel
Polish-Japanese Institute of Information Technology
Warsaw, Poland
ODRA = Object Database for Rapid Application development
ODRA

A homogeneous development environment, consisting of three, highly integrated elements:

• object-oriented DBMS, completely different from ODMG-like architectures

• object-oriented query/programming language (based on the Stack-Based Approach and SBQL), with queries treated as expressions

• middleware based on updatable views and ideas known from federated databases
The syntactic tree is the subject of several operations, in particular, strong type checking, optimization by rewriting, optimization by indices and finally, compilation to a bytecode.

Figure 1 Architecture of ODRA

- The strong type checker \cite{[10, 11]} takes a query/program syntactic tree and checks if it conforms to the declared types. Types are recorded within a client local metabase and within the metabase of persistent objects that is kept on the server. The metabases contain information from declarations of volatile object types (that are a part of source programs) and from a database schema. The module that organizes the metabases is not shown. The strong static type checker simulates actual execution of a query during compile time. The type checker has several other functions. In particular, it changes the query syntactic tree by introducing new nodes for automatic dereferences, automatic coercions, for typing literals, for resolving elliptic queries and for dynamic type checks (if static checks are impossible). The type checker introduces additional information to the nodes of the query syntactic tree that is necessary further for query optimization.
SBQL in ODRA

SBQL is a prototype query language that is used to explain the semantics of the Stack Based Approach.

- SBQL in Odra has been extended to a database application programming language
- declarative, high-level, object-oriented programming
- queries as expressions
- typical programming language (modules, procedures, classes, etc.) and database (indexes, triggers, etc.) mechanisms
- semi-strong static type checking
- compile-time (e.g. query rewriting) and runtime (e.g. indexes) optimizers
- updatable views
Scenarios of application

A) non-distributed application
B) 3-tier client-server
C) federated database
A simple, distributed application

module client {
    dblink aps appuser/apppasswd/appuser.appserver@my.appserver.pl;
    
    main() {
        print aps.count_employees("Smith");
    }
}

module appserver {
    dblink dbs1 dbuser1/apppasswd/dbuser1.dbserver@my.dbserver1.pl;
    dblink dbs2 dbuser2/dbpasswd/dbuser2.dbserver@my.dbserver2.pl;
    
    count_employees(n : string) : integer {
        return count (dbs1.emp union dbs2.emp) where ename = n;
    }
}

module dbserver {
    emp : record { ename : string; salary : integer; job : string; } [0..*]
}
eGovBus architecture

e-Gov Bus - Advanced e-Government Information Service Bus
(European Commission 6-th Framework Programme, IST-26727)
Thank You!

More information on SBQL and our projects:
http://www.sbql.pl