Overview of the Project ODRA

Radosław Adamus[1,3], Marcin Daczkowski[1], Piotr Habela[1], Krzysztof Kaczmarski[1,4], Tomasz Kowalski[1,3], Michał Lentner[1], Tomasz Pieciukiewicz[1], Krzysztof Stencel[1,2], Kazimierz Subieta[1,3], Mariusz Trzaska[1], Tomasz Wardziak[1], Jacek Wiślicki[1,3]

[1] Polish-Japanese Institute of Information Technology, Warsaw, Poland
[2] Institute of Informatics, Warsaw University, Warsaw, Poland
[3] Computer Engineering Department, Technical University, Łódź, Poland
[4] Faculty of Mathematics and Information Science, Warsaw University of Technology, Warsaw, Poland
{radamus, tkowals, jacenty}@kis.p.lodz.pl, merdacz@gmail.com, {habela, m.lentner, pietia, stencel, subieta, mtrzaska, wardziak}@pjwstk.edu.pl, k.kaczmarski@mini.pw.edu.pl

Abstract. ODRA (Object Database for Rapid Application development) is an object-oriented database management system provided as an integrated programming tool for building various business and administration applications, including centralized and distributed systems, Web applications, service buses, virtual repositories, P2P networks, and so on. The paper presents general architecture of ODRA, its object model, back-end interoperability facilities, front-end programming interfaces and integrated development environment. ODRA is based on the database query and programming language SBQL (Stack-Based Query Language) and virtual updateable views defined and used in SBQL. ODRA is the basis for two European projects: eGov Bus, aiming at the development of interoperability service bus for public administration, and VIDE, aiming at the development of a visual and textual programming language based on the OMG MDA and OMG specifications of Executable UML and a query language OCL.

1 Introduction

ODRA (Object Database for Rapid Application development) [1, 2, 3] is a prototype object-oriented database management system based on SBA (Stack-Based Architecture) [4, 5, 6, 7, 8, 9]. ODRA is developed at the Polish-Japanese Institute of Information Technology (Warsaw, Poland) since March 2004. The project started with no financial support, as preparation for applications aiming at domestic and European Commission research grants. Currently it is supported by two 6-th FP European projects (eGov Bus, IST-4-026727-ST, and VIDE, IST 033606 STP) and by Polish Ministry of Science and Higher Education. In the project ODRA participated or participates more than 20 researchers and developers (plus many students). The project have joined researchers from several Polish academic institutions. Besides the pragmatic goal of building a new modern software, the important meaning of the project is education and training. Due to the project many its participants have become experts in databases, query languages, programming languages, compiler construction, distributed architectures, query optimization, strong typing, interoperability, object-oriented technologies, Web technologies, OMG standards, etc. In the software research and development own practical experience is the best teacher.

The main motivation for the ODRA project is to develop new paradigms of database application development. This goal is going to be reached mainly by increasing the level of abstraction at which the programmer works. ODRA introduces a new universal declarative query and programming language SBQL (Stack-Based Query Language) [1, 4, 5, 6, 7], together with distributed, database-oriented and object-oriented execution environment. Such an approach provides functionality common to the variety of popular technologies (such as relational/object databases, several types of middleware, general purpose programming

languages and their execution environments) in a single universal, easy to learn, interoperable and effective to use application programming environment.

ODRA consists of three closely integrated components:

- Object Database Management System (ODMS)
- Compiler and interpreter for object-oriented query programming language SBQL
- Middleware with distributed communication facilities based on the distributed databases technologies.

The system is additionally equipped with a set of tools for integrating heterogeneous legacy data sources. The continuously extended toolset includes importers (filters) and/or wrappers to XML, RDF, relational data, web services, etc.

Each installation of ODRA can work as a client and as a server; multiple-client – multiple-server architectures are possible. ODRA makes it possible to create multi-layered architectures, where some client is considered a server for lower-level clients. ODRA has all chances to achieve high availability and high scalability because it is a main memory database system with memory mapping files and makes no limitations concerning the number of servers working in parallel. In ODRA many advanced optimization methods that improve the overall performance without compromising universality and genericity of programming interfaces have been implemented.

The rest of the paper is organized as follows. Section 2 describes the architecture of ODRA. Section 3 explains the ODRA object model. Section 4 outlines the ODRA query language SBQL. Section 5 presents ODRA virtual updatable views. Section 6 and 7 describe ODRA interoperability facilities. Section 8 introduces ODRA IDE. Section 9 gives an example of the ODRA-based distributed application architecture. Section 10 concludes.

2 Architecture of ODRA

There are several views on the internal architecture of ODRA. Understanding internal organization of the system can be helpful in understanding of functionalities and modes of using ODRA in applications. Fig.1 presents a view on the architecture, which involves data structures (figures with dashed lines) and program modules (grey boxes). The architecture takes into account the subdivision of the storage and processing between client and server, strong typing and query optimization (by rewriting and by indices). The subdivision on client and server is only for easier explanation; actually, each ODRA installation can work as a client and as a server. Many clients can be connected to a server and a client can be connected to many servers. Below we present a short description of architectural elements from Fig.1.

The client side:

- A source code of a query/program is created within the **Integrated Development Environment**, which includes an editor, a debugger, storage of source programs, storage of compiled programs, etc.
- A **lexer** and **parser** takes a query/program source as input, makes syntactic analysis and returns an Abstract Syntactic Tree (AST) of the query or program..
- A **query/program AST** is a data structure which keeps the abstract syntax in a well-structured form, allowing for easy manipulation (e.g. inserting new nodes or sub-trees, moving some sub-tree to another part of the tree, removing some sub-trees, etc.). Each node of the tree contains a free space for writing various query optimization information.

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.

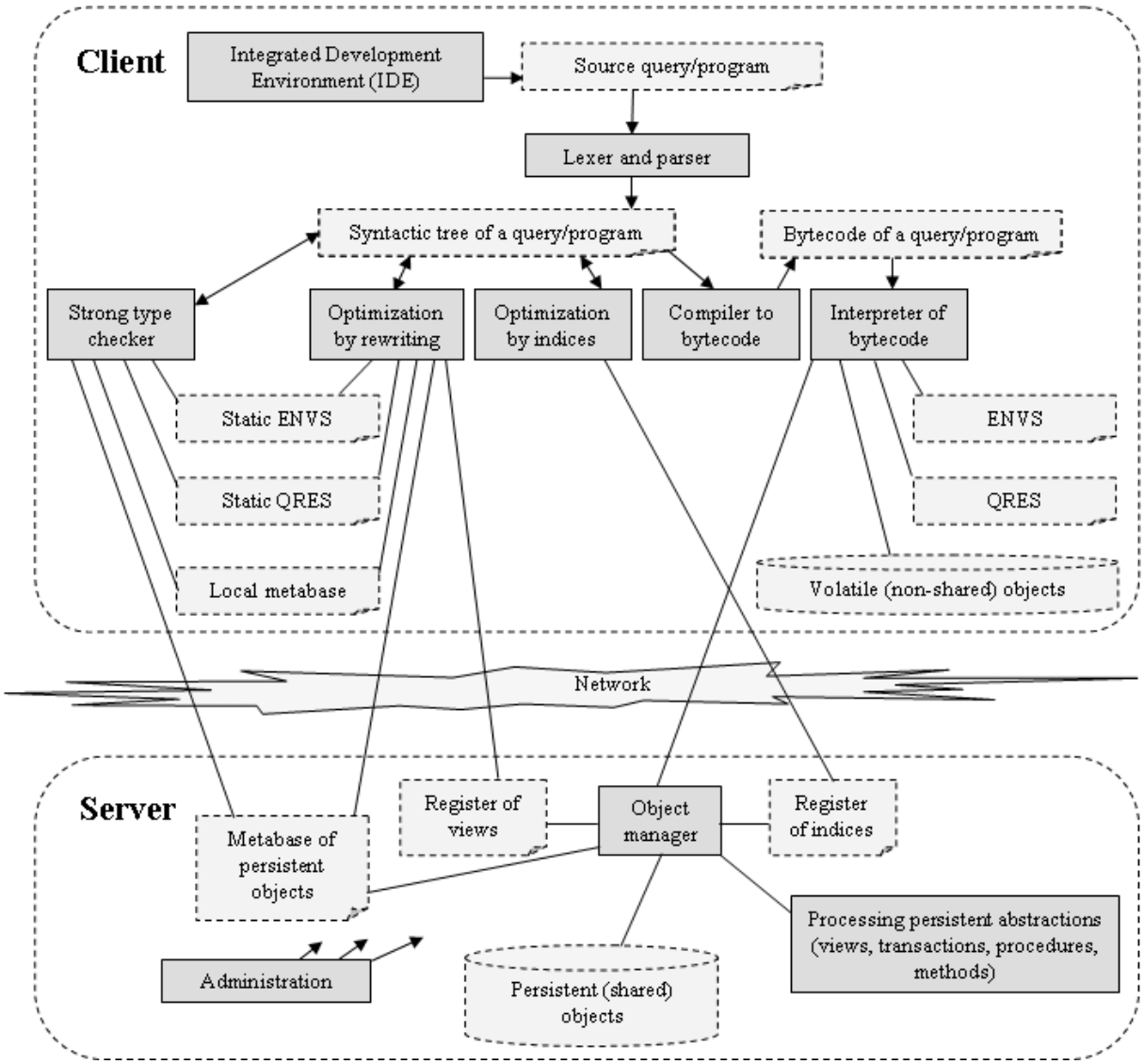

Figure 1 Architecture of ODRA

- The **strong type checker** [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 type checker uses two stacks, static ENVS (keeping signatures of runtime environments) and static QRES (keeping signatures of query results). 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.

- **Static ENVS** - static environment stack. It is a compile time counterpart of the environment stack (call stack) known from almost all programming languages.
- **Static QRES** - static result stack. It is a compile time counterpart of the result stack (arithmetic stack) known from almost all programming languages.**Local metabase.** It is a data structure containing information of types and specifications introduced in source programs.
- **Optimization by rewriting** - this is a program module that changes the syntactic tree that is already annotated by the strong type checker. There are several rewriting methods that are developed for SBA, in particular:
  - Performing calculations on literals.
  - Changing the order of execution of algebraic operators.
  - Application of the query modification technique, which changes invocations of views into view bodies. To this end, the optimization module refers to the *register of views* that is kept on the server.
  - Removing dead sub-queries, i.e. sub-queries that do not influence the final query result.
  - Factoring out independent sub-queries: sub-queries whose results are not changed within some loop are factored out outside the loop.
  - Shifting conditions as close as possible to the proper operator, e.g. shifting selection condition before a join.
  - Methods based on the distributivity property of some query operators.
- **Optimization by indices**. This is a program module that changes the syntactic tree that is already annotated by the strong type checker. Changes concerns some sub-trees that can be substituted by invocation of indices. To this end, the optimization module refers to the **register of indices** that is kept on the server. Changes depend on the kind of an index. The module will be extended to deal with cached queries.
- **Compiler to bytecode.** This module takes the strongly checked and optimized syntactic tree of a query/program and produces a bytecode that can be executed by the interpreter. In the prototype implementation we developed our own bytecode format called Juliet. In the future we consider the possibility to generate directly the Java bytecode but it needs further research.
- **Interpreter of bytecode**. During runtime it takes instructions of a bytecode and triggers proper routines. To this end it uses two run-time stacks, ENVS (environment stack) and QRES (query result stack). The interpreter refers to volatile objects that are kept on a client and to any resources that are available on the server, in particular persistent (shared) objects. All the server resources are available through the **object manager**.

The server side:

- **Persistent (shared) objects** - this is a part of the object store commonly known as a database.
- **Object manager** - this is a low-level API that performs everything on persistent objects that is needed.
- **Metabase of persistent objects** - this is a compiled database schema plus some additional information, e.g. necessary for optimization.

- **Processing persistent abstractions** (views, transactions, procedures, methods, etc.) - essentially, this module contains all basic elements of the client side and extends them by additional functionalities.
- **Register of indices** and **register of views** are data structures that contain and externalize the information of created indices and created views. The information is used by the client for query optimization. Internally, this information is fulfilled by the administration module.
- **Administration module** - makes all operations that are necessary on the side of the server, e.g. introducing a new index, removing an index, introducing a new view, changing the database schema, logins and authorization of users, etc.

3 ODRA Object-Oriented Data Model

The ODRA data model is similar to the UML object model. Because in general UML is designed for modeling rather than for programming several changes have been made to the UML object model that do not undermine seamless transition from a UML class diagram to an ODRA database schema. The ODRA data model covers also XML (except some of its minor features that are not supported). In the same way it covers a lot of other models, including the RDF model, the Topic Maps model, etc. The ODRA object model covers also the relational model as a particular case; this feature is essential for making wrappers to external sources stored in relational databases. Below we present a short description of the main data model elements.

Objects. The basic concept of the ODRA database model is *object*. It is an encapsulated data structure storing some consistent bulk of information that can be manipulated as a whole. A database designer and programmers can create database and programming objects according to their own needs and concepts. Objects can be organized as hierarchical data structures, with attributes, sub-attributes, etc.; the number of object hierarchy levels is unlimited. Any component of an object is considered an object too.

Any objects has an *external name* (or more names) that can be used by the programmer to identify (to bind) the object from a source query or program. External names need not be unique. A name (such as *Account, Invoice, DateOfBirth*, *SocialSecNbr*, etc.) usually bears some conceptual meaning in the business domain. Any object has also an *internal identifier* that is used internally as a reference. Internal identifiers are unique for the given environment of objects. Internal identifiers are non-printable and have no meaning in the business domain. The programmer never uses internal identifiers explicitly.

Collections. Objects within a collection have the same name; the name is the only indicator that they belong to the same collection. Usually objects from a collection have the same type, but this requirement is relaxed for some kinds of heterogeneous collections. Collections can be nested within objects with no limits (e.g. in this way it is possible to represent repeating attributes).

Links. Objects can be connected by pointer links. Pointer links represent the notion that is known from UML as *association*. Pointer links support only binary associations; associations with higher arity and/or with association classes are to be represented as objects and some set of binary associations. This is a minor limitation in comparison to UML class diagrams, introduced to simplify the programming interface. Pointer links can be organized into bidirectional pointers enabling navigation in both directions. If a bidirectional link connects

objects A and B, then it is understood as a pointer from A to B and a pointer from B to A. Such bidirectional links behave consistently as twin interrelated pointers: updating of one of them causes immediate and automatic updating of its twin.

Modules. In ODRA the basic unit of database organization is a module. As in popular object-oriented languages, a module is a separate system component. An ODRA module groups a set of database objects and compiled programs and can be a base for reuse and separation of programmers' workspaces. From the technical point of view and of the assumed object relativism principle modules can be perceived as special purpose complex objects that store data and metadata.

Types, classes and schemata. A class is a programming abstraction that stores invariant properties of objects, in particular, its type, some behavior (methods, operations) and (optionally) an object name. A class has some number of member objects. During processing of a member object the programmer can use all properties stored within its class. The model introduces atomic types (**integer, real, string, date**, **boolean**) that are known from other programming languages. Further atomic types are considered, but not implemented yet. The programmer can also define his/her own complex types. Collection types are specified by cardinality numbers, for instance, [0..*], [1..*], [0..1], etc.

Inheritance and polymorphism. As in the UML object model, classes inherit properties of their superclasses. Multiple inheritance is allowed, but name conflicts are not automatically resolved. The methods from a class hierarchy can be overridden. An abstract method can be instantiated differently in different specialized classes (due to late binding); this feature is known as *polymorphism*.

Persistence and object-oriented principles. The model follows the orthogonal persistence principle, i.e. a member of any class can be persistent or volatile. Shared server objects are considered persistent, however, non-shared objects of a particular applications can be persistent too. The model follows the classical compositionality, substitutability and open-close principles assumed by majority of object-oriented programming languages. Shared (server) objects are the subject of transactional ACID semantics based on the 2PL algorithm.

Distinction between proper data and metadata (ontology) is not the property of the ODRA database model. The distinction can be important on the business model level, but from the point of view of ODRA both kinds of resources are treated uniformly.

4 ODRA Query Language SBQL

SBQL (Stack-Based Query Language) is a powerful query and programming language addressing the object model described above. SBQL is precise with respect to the specification of semantics. SBQL has also been carefully designed from the pragmatic (practical) point of view. The pragmatic quality of SBQL is achieved by orthogonality of introduced data/object constructors, orthogonality of all the language constructs, object relativism, orthogonal persistence, typing safety, introducing all the classical and some new programming abstractions (procedures, functions, modules, types, classes, methods, views, etc.) and following commonly accepted programming languages' and software engineering principles. The principles include:

- *orthogonality* (keep unrelated features unrelated),
- *compositionality* (avoid big syntactic and semantic patterns, as well as far context dependencies in a program code),

- *universality* (the language should cover the assumed domain),
- *generality* (using language features for many purposes),
- *parsimony* (avoid redundant features),
- *clean formal semantics*,
- *openness* (use external systems and specialized tools),
- *no semantic anomalies* (no exceptional features and irregular treatment),
- *no semantic reefs* (programmer's understanding and machine processing coincide),
- *correspondence* (the methods of binding do not depend on a context),
- *conceptual closure* (introducing a feature A enforces next features that appear from the combination of A with already existing features),
- *safety* (typechecking, assertions, constraints),
- *semantic relativity* (identical properties of parent and nested entities),
- *conceptual continuation* (bigger tasks are to be smooth extensions of smaller tasks).

SBQL queries can be embedded within statements that can change the database or program state. We follow the state-of-the-art known from majority of programming languages. Typical imperative constructs are *creating* a new object, *deleting* an object, *assigning* new value to an object (updating) and *inserting* an object into another object. We also introduce typical control and loop statements such as *if…then…else…, while* loops, *for* and *for each* iterators, and others. Some peculiarities are implied by queries that may return collections; thus there are possibilities to generalize imperative constructs according to this new feature.

SBQL in ODRA project introduces also procedures, functions and methods. All procedural abstractions of SBQL can be invoked from any procedural abstractions with no limitations and can be recursive. SBQL programming abstractions deal with parameters being any queries; thus corresponding parameter passing methods are generalized to take collections into account. We have implemented the *strict-call-by-value* method which makes it possible to achieve the effects of *call-by-value*, *call-by-reference*, and more. Transactions are also considered procedural abstractions, syntactically and semantically very similar to procedures. Nested transactions are supported.

SBQL is a strongly typed language. Each database and program entity has to be associated with a type. However, types do not constraint semi-structured nature of the data. In particular, types allow for optional elements (similar to null values known from relational systems, but with different semantics) and collections with arbitrary cardinality constraints. Strong typing of SBQL is a prerequisite for developing powerful query optimization methods based on query rewriting and on indices.

For ODRA we have implemented a generic gateway to Java libraries. This facility allows one to use calls to Java programs within SBQL programs. The facility is especially useful to extend SBQL with GUI, with string operators, with J2EE capabilities, etc.

5 Virtual Updatable Views

Virtual views (known from SQL) are frequently considered as a tool for adapting heterogeneous data to some common schema assumed by the business model of an application. Classical SQL views do the mapping from stored data into virtual data. However,

some applications may require updating of virtual data; hence there is a need for a reverse mapping: updates of virtual data are to be mapped into updates of stored data. This leads to the well-known *view updating problem*: updates of virtual data can be accomplished by updating of stored data on many ways, but the system cannot decide which of them is to be chosen. In typical solutions these updates are made by side effects of view invocations (as presented, e.g., in the Oracle solution). Due to the view updating problem, many kinds of view updates are limited or forbidden.

In the ODRA project (basing on previous research) another point of view has been introduced. In general, the method is based on overloading generic updating operations (create, delete, update, insert, etc.) acting on virtual objects by invocation of procedures that are written by the view definer. The procedures are an inherent part of the view definition. The procedures have full algorithmic power, thus there are no limitations concerning the mapping of view updates into updates of stored data. SBQL updatable views allow one to achieve full transparency of virtual objects: they cannot be distinguished from stored objects by any programming option. This feature is very important for distributed and heterogeneous databases.

SBQL views present a new method that attempts to achieve two qualities: high-level view definition, as in SQL views, and full algorithmic power (including updating) as e.g. in OMG CORBA adapters or wrappers.

6 Back-End Interoperability Facilities

One of the important ODRA project tasks was to develop a wide range of the back-end interoperability facilities to enable an access to legacy applications and data. The most important are: wrapper to relational databases, wrapper to web services and XML importer/exporter.

6.1 Wrappers to external relational databases

The need for a relational data wrapper arises especially in the context of virtual repositories where client applications work on the common (canonical) database schema. The common model is the result of negotiations and tradeoffs between business or administrative partners having incompatible (heterogeneous) data and services. This makes development of an object-relational wrapper more constrained than in a classical Object-Relational Mapping (ORM) case known e.g. from Java or .Net oriented wrappers (such as JDO, EJB, TopLink and Hibernate). The wrapper should deliver virtual objects and services according to the predefined object-oriented canonical schema. There could be little freedom or could be inconvenient to change the canonical schema due to limitations of ORM capabilities. Moreover, the mapping to an UML-like object model is much more complex problem than mapping to the Java or C# object model. The mapping should also support both directions: mapping of relational data into virtual objects, and mapping of updating operations on virtual objects into updates of relational tables through SQL. Hence, the architecture and algorithms for ORM aiming at advance business or public administration applications must be developed with proper universality of the mappings in minds.

One of the most important features of the ODRA wrapper is the possibility to utilize an SQL optimizer. In all known RDBMS-s the optimizer and its particular structures (e.g. indices) are transparent to the SQL users. A naive implementation of the wrapper causes that it generates primitive SQL queries such as *select * from R*, and then, processes the results of such queries by SQL cursors. Hence the SQL optimizer has no chances to work. Our experience has shown

that direct (static, i.e. compile time) translation of object-oriented queries into SQL is unfeasible even for a typical case.

The solution of this problem that was implemented in ODRA is based on the object-oriented query language SBQL, virtual object-oriented views defined in SBQL, query modification technique, and an architecture that will be able to detect in a query syntactic tree some patterns that can be directly mapped as optimizable SQL queries. The patterns match typical optimization methods that are used by the SQL query optimizer, in particular, rewriting, indices and fast joins. The idea is fully implemented within the ODRA prototype, including not only retrieval, but also some updating statements. The currently supported RDBMS are: Axion, Cloudscape, DB2, DB2/AS400, Derby, Firebird, Hypersonic, Informix, InstantDB, Interbase, MS Access, MS SQL, MySQL, Oracle, Postgres, SapDB, Sybase and Weblogic.

6.2 Wrapper to Web Services

The ODRA system includes the Web Services facilities that allow for consuming existing web services through ODRA Web Services proxy mechanism. The Web Services Proxies mechanism allows include volatile and external data into a database. Interaction with remote services is achieved through strongly typed stubs. They are realized as regular database objects. That makes them transparent for database – any of its mechanisms can be applied to them. For example they can profit from query, type checking and authorization database features. That makes remote and local resources interaction very similar from the developer's point of view.

The ODRA includes also support for stubless, dynamic invocation of web services. Realized as a part of an external method call interface, it allows the user to create a document that is sent to a web service during runtime, creating any valid XML structure. Returned results may be consumed like results of any other SBQL query.

Currently ODRA proxies support subset of WS-I BP 1.1 compliant web services.

6.3 XML importer/exporter

The XML importer/exporter is implemented in such a way that no information contained in the original XML file is lost in its ODRA representation. XML exporter is implemented as a generic utility having an SBQL query as a parameter. XML tags are deduced from ODRA objects returned by a query and/or from auxiliary names that are used within the query. Because SBQL queries have full algorithmic power, the XML importer/exporter has no limitations concerning transformation of XML files into XML (or other) files. The XML importer/exporter can also invoke SBQL views as well as any SBQL functions.

7 Front-End ODRA Application Programming Interfaces.

The ODRA system provides several methods to access its resources from external applications. All the methods use SBQL.

- Java Object Base Connectivity (JOBC). This interface follows the style of the JDBC interface to relational databases. The differences concern input (SBQL rather than SQL) and output (serialized ODRA objects rather than serialized relational tables);
- Web Services facilities. This interface makes it possible to create Web Services on top of the ODRA resources. There are three kinds of web services endpoints in ODRA: generic, procedure and class based. Each of them works in the context of a special web service user account. They are all designed to meet WS-I BP 1.1 compliance.

- JSP interface. This interface makes it possible to generate dynamic web pages utilizing standard JSP and some web server. Dynamic elements on HTML pages are determined through SBQL.

8 ODRA Integrated Development Environment

ODRA system is equipped with an IDE that allows for maintain (edit, compile, store, test, debug, administer) of the ODRA applications. The IDE is based on the jEdit editor [12]. Currently an ODRA IDE is being extended with the functionality of fully-fledged administration, performance tuning and optimization module, which will include granting access privileges for users and for particular virtual resources, creating/removing indices, import/export of files (e.g. XML), determining modes of execution and output, etc.

9 Example ODRA Based Application Architecture

ODRA is a combination of object-oriented database management system with own query and programming language, virtual updateable views, stored procedures, stored classes and methods and with many interoperability modules that can be used in different configurations, depending on needs of a particular business application. There are a lot of various software architectures that can be developed on the ODRA system.

Because the ODRA system is especially devoted to distributed applications Figure 2 presents some architectural variant for a Virtual Repository that can be developed. The picture presents some possible configuration of developed software units.

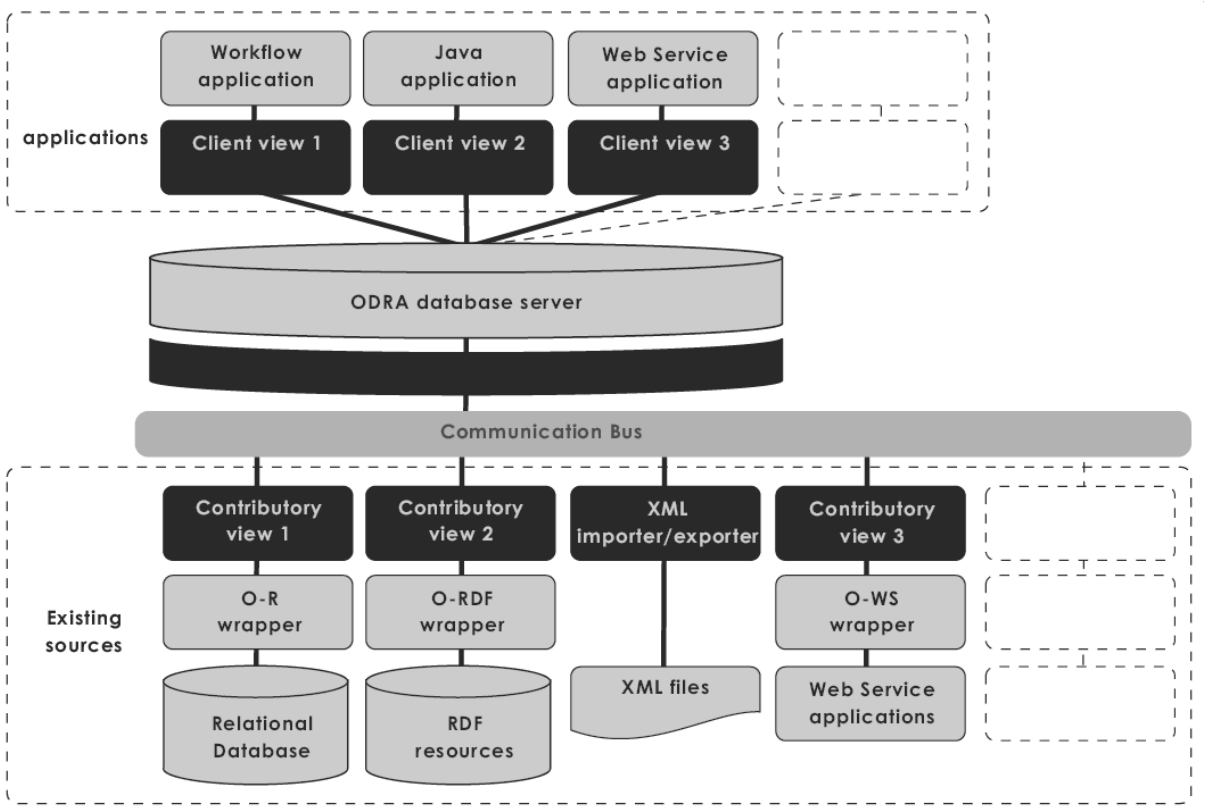

Figure 2 Virtual Repository architecture of applications based on ODRA

A central part of the architecture consists of ODRA, an object-oriented DBMS. Existing resources (bottom of the figure) are extended by wrappers and contributory views (or importers/exporters) that convert data/services proprietary to particular existing applications into the format acceptable for ODRA. The application developers can install as many ODRA servers as necessary, addressing the same distributed sources. The integration view on an ODRA server allows for the virtual integration of data and services supplied by distributed sources, supporting data and function abstractions. The virtual repository front-end will provide various APIs to access virtually integrated data, including workflow applications, Java applications, Web services applications, and others. A particular user works with his/her client view that is a tailored part of the entire virtual repository schema.

Among many other functions, the virtual repository architecture allows for transparent access to external information resources and for unlimited transformations of complex document structures.

10 Conclusion and Current Developments

In this paper we have described the features of the prototype system ODRA. The main goal of the design was to achieve high level of interoperability and flexibility on the fields of creating future ODRA based applications with a stress put on distributed architectures. The design is based on the Stack-Based Architecture (SBA) and Stack-Based Query Language (SBQL) and supports universal, declarative programming language, together with distributed, database-oriented and object-oriented execution environment.

The ODRA is still under development. Currently the system is being extended with the transaction mechanism that is based on the assumption that the transaction is represented as a regular object in the database that can be managed with a special administration utility. Our goal is to give more freedom for distributed transactions that is assumed in 2PC and 3PC protocols. The network protocols are being extended with the possibility to create P2P scenarios applications. Additionally we plan to extend the object model with dynamic roles to express the dynamic inheritance and many other features.

In the VIDE project ODRA is used as a main engine for implementing the OMG Executable UML 2.1 [13] and OMG OCL 2.0 [14] specifications, within OMG MDA (Model Driven Architecture). This work is in progress. OCL is already implemented on top of SBQL as a database query language. OCL is being integrated with UML Action Semantics; unfortunately inconsistencies of OMG specifications cause integration difficulties. The programming language VIDE joining AS and OCL is prepared as an executable interface on the PIM (Platform Independent Model) level; then, the specification is to be compiled according to the EMF (Eclipse Modeling Framework) format. A metamodel stored in EMF is then the subject of model compilers that deliver PSM (Platform Specific Model) codes. More details on this project will be the subject of other papers, currently in preparation.

References

[1] Various information on SBQL, including recent presentations and implementations: http://www.sbql.pl/various

[2] ODRA discussion forum (in Polish): http://iolab.pjwstk.edu.pl/forum/

[3] M.Lentner, K.Subieta: ODRA: A Next Generation Object-Oriented Environment for Rapid Database Application Development. Springer LNCS 4690, 130-140

[4] SBA and SBQL Web pages: http://www.sbql.pl/

[5] Papers and reports: http://www.sbql.pl/articles

[6] PhD theses: http://www.sbql.pl/phds

[7] K.Subieta. *Theory and Construction of Object-Oriented Query Languages* (in Polish), PJIIT - Publishing House, 2004, 522 pages

[8] K.Subieta, C.Beeri, F.Matthes, J.W.Schmidt. *A Stack-Based Approach to Query Languages*. Proc. 2nd East-West Database Workshop, 1994, Springer Workshops in Computing, 1995, 159-180

[9] K.Subieta, Y.Kambayashi, J.Leszczyłowski. *Procedures in Object-Oriented Query Languages*. Proc. 21-st VLDB Conf., Zurich, 1995, pp.182-193

[10] K.Stencel: *Semi-strong Type Checking in Database Programming Languages* (in Polish), PJIIT - Publishing House, 2006, 207 pages

[11] M.Lentner, K.Stencel, K.Subieta: Semi-strong Static Type Checking of Object-Oriented Query Languages. SOFSEM 2006: 399-408

[12] jEdit project website: http://www.jedit.org

[13] OMG: Unified Modeling Language Specification (Superstructure and Infrastructure) Version 2.1.2. November 2007 http://www.omg.org/spec/UML/2.1.2

[14] OMG: Object Constraint Language OMG Available Specification Version 2.0. May 2006. http://www.omg.org/cgi-bin/doc?omg/06-05-01