Binding of programming languages with subject mediators for scientific problems solving

Alexey Vovchenko[1],
Supervisor: Leonid Kalinichenko[1],
[1] Institute of Informatics Problems of the Russian Academy of Science
itsnein@gmail.com

Abstract. Definition of an adequate architecture of a procedural programming language (PL) binding to the declarative language used for specification of mediators discussed in the paper. A set of features to be used for characterization and evaluation of different approaches of PL bindings to information resource management systems is proposed. It is shown how a set of supported features should be selected to solve impedance mismatch problems.

Keywords: Subject mediators, problems solving, impedance mismatch, language binding.

1 Introduction

This paper[1] has been prepared in frame of the PhD research that is being performed in accordance with the conception of scientific problems formulation in the subject mediation environment [1]. The subject mediators are defined in terms of a subject domain independently of existing information resources (databases, services, processes). Heterogeneous distributed information resources relevant to a problem are registered in a mediator in a form of bidirectional mappings between resource classes and mediator classes. Such mappings are expressed in a declarative way by means of GLAV views [2]. Subject mediator specifications and views are defined by frame based, object-oriented language [3] combined with the typed first order logic used for expression of formulae, assertions and rule-based programs over the mediator classes and functions. Problem solving support in the mediation environment is provided by its various components – mediators, information resources (IR), programmable wrappers, transformers defined by the mappings between resource and mediator classes, application programs over the mediators. The PhD research is devoted to the issues of efficient planning and organization of problem solving in the mediation environment designed as a dispersed composition of the interoperable heterogeneous components. The PhD research may be divided into three separate issues. The first one is the problem of dispersed organization of the problem solving itself. The second one is the issue of information resource wrappers semi-automatic generation. The last one is the main topic of this paper and consists in the definition of an adequate architecture of a procedural programming language (PL) binding with the declarative language used for specification of mediators. The paper is structured as follows. In

[1] This research has been done under the support of the RFBR (projects 10-07-00342-a) and the Program for fundamental research of the Presidium of RAS 15P (project 4.2).

section 2 the PhD research outline is presented. Characterization of binding issues is considered in section 3. Brief characterization of impedance mismatch problem and its solving approaches are considered in section 4. In section 5 the related works are considered. In conclusion the contribution of the work is summarized.

2 PhD research outline

The issues of dispersed organization of problem solving in the mediation environment arise from the fact that various components of the environment (such as the mediator declarative program, mappings between resource classes and mediator classes expressed in the view definitions, programs in procedural language, resource wrappers taking into account capabilities of the resources, resources themselves) can be assigned for implementation of various parts of the problem solving algorithm. Main task of the PhD research consists in selection of an efficient variant of dispersed implementation of the algorithm. A set of possible variants forms a dispersed implementation model (DIM). Each state of the model is characterized by assignment of the mediation environment components for all parts of the algorithm. Two accompanying tasks are also studied in frame of the PhD work. One of them is an approach for semi-automatic development of the resource wrappers with an architecture that supports adaptive planning and permits efficient capability-based operation execution. Another one is associated with the development of an adequate architecture of a procedural programming language (PL) binding with the declarative mediator specification language.

In spite of the numerous research and development in the areas of interoperable architectures, PL bindings with databases and existing standards (e.g., ODMG 3.0 standard [4], C++ binding with Oracle (OCCI) [5], Java interface to a DB (JDBC) [6], standard SQLJ for embedded SQL in Java [7], Sun Java Data Objects (JDO) standard [8], Microsoft LINQ [9]), up to now there was no attempt to make systematic analysis of approaches to the design and development of various bindings. Such state of the art creates difficulties for comparison of capabilities of various binding approaches as well as for creation of bindings for new languages and systems. This paper is aimed at the definition of the set of features for characterization and evaluation of various bindings of PL with the systems for management of various information resources. Development of such systematization is motivated by the need for the well-grounded definition of an adequate architecture of binding of the subject mediator support facilities with the programming languages. Proposing such systematization, we shall narrow the discussion with the class of object-oriented languages.

3 Characterization of binding issues in terms of PL and databases

Though this paper discusses a binding between PL and subject mediation support facilities, in this section we consider the binding issue in the context of PL and databases. Due to the overwhelming number of researches, standards and implementations of bindings in database area, such selection of context is reasonable.

At the same time the generality of considerations is preserved since subject mediator environment can be treated as a virtual resource management system.

Mapping of DDL types to PL types, including type functions and invariants, should be commutative. Preserving of type relations (e.g., subtyping) is also assumed. Respect of the commutativity requirement leads to preserving of information and operations of a source data model (determined by DDL) in a target data model (determined by PL). The commutativity is reached if the mapping diagram for DDL is commutative [10]. In particular, type mapping commutativity justification is based on a proof that the type operations of source model are refined by target model [11].

Another important binding issue is the mapping of a query language (QL) into PL. A query containment property [12] should be satisfied for such mapping. To make static type checking support possible, PL should be extended with query language constructs. For dynamic checking considering a query just as a string is enough.

Database object manipulation in PL can be carried out either by means of modification of persistent type values, or by means of special DML operations. For static type checking support, PL should be extended with DML constructs. This is not required in dynamics, therefore the DML operations can be passed to DBMS just in a string form.

Providing of a distinction between persistent and transient types (and their instances) in PL is also an important binding issue. Database DDL classes mapping to PL collections requires semantic modification of such collections in PL giving to them meaning of persistence. Modification of PL itself is actually required for that. Static type checking requires query result collections to be generic (Set <Type>).

Under such assumptions, a set of possible PL to DBMS binding architectures can be characterized by the following orthogonal features.

DDL Mapping (LM):

Type2Type (T2T): Type (class) specification in DDL is mapped to a type (persistent collection) specification in PL

```
//Specification in DDL (SQL)
CREATE TYPE emp UNDER person AS (EMP_ID INTEGER, SALARY REAL)
           INSTANIABLE NOT FINAL REF ( EMP_ID )
INSTANCE METHOD GIVE_RAISE(AMOUNT REAL) RETURNS REAL;
CREATE TABLE empls OF emp;
//Type specification in PL(Java)
Class Emp extends Person implements PersistentObject {
      private int emp_id;
      private float salary;
      public float give_raise(float amount);}
DBCollection<Emp> empls = new DBCollection<Emp>();
```

Type2TypePattern (T2P): Type (class) specification in DDL is presented as value (object) in PL. This value is formed according to the respective type specification

```
//Specification in DDL (SQL)
CREATE TABLE customer (Name char(50), Birth_Date date)
//Type specification in PL(Java)
Class Attribute {String attName; String attType;}
Class DBTable {String tableName; List<Attribute> atts;}
//PL Object
table = {"customer", atts = {
```

```
{"Name","char(50)"}, {"Birth_Date","date"}}}
```

PL-Type2DDLType (PL2T): Type specification in PL(Java)

```
//Specification in C#
[Table(Name="People")]
public class Person {
[Column(DbType="nvarchar(32)",Id=true)] public string Name;
[Column] public int Age; }
//In DataBase the following table is created:
create table People (Name nvarchar(32) primary key, Age int)
```

Query Mapping (QM):

Query2String (Q2S): Query is represented as a string in PL

```
//QL – SQL, Programming Language - Java
OQLQuery query = impl.newOQLQuery();
query.create("select t.assists.taughtBy from t in TA where
t.salary > $1 and t in $2 ");
```

Query2QueryPattern (Q2P): Query is represented as a parameterized object in PL

```
//QL - Declarative JDOQL, Programming Language - Java
Query q = pm.newQuery(org.jpox.Person.class, "lastName ==
\"Jones\" && age < age_limit");
q.declareParameters("double age_limit");
List results = (List)q.execute(20.0);
```

Query2PL-Query (Q2PLQ): PL is extended with QL constructs

```
//QL – SQLJ, Programming language - Java
#sql ordIdIter = { SELECT OrderId FROM otn_deliverydetail };
while (ordIdIter.next()) {
      id = ordIdIter.orderid();
      gui.addToList(id);}
```

Completeness of Query Support (QS):

ClassCompositionQuery (CCQ): Query language is mapped completely. User is provided with a possibility to express any query in PL expressible in the native QL.

OneClasslQuery (OCQ): Query language is strictly limited. User is provided with a possibility to retrieve only a collection of objects of one particular type (compositions of classes are not possible) satisfying some condition.

Object Manipulation by DML (OM):

DMLoperator2String (DML2S): DML operator is represented as a string in PL

DMLoperator2PL-operator (DML2O): PL is extended with DML constructs

Object Manipulation by Object Persistence:

PersistentObjects (PO): Persistent data is supported, modifying of persistent objects in PL causes changes in DB.

TransientObjects (TO): Only transient data is supported, modifying of objects in PL is not reflected in DB.

Generic Collections (C):

Generic-SetType (GST): Generic collections with a type as parameter (Set<Type>) are supported.

Strict-SetType(SST): Generic collections are not supported.

Usability of the binding features introduced will be shown applying them to the definition of a set of features sufficient for support of static or dynamic type checking. For a static type checking support it is enough to realize the following set of features: Type2Type, Query2PL-Query, DMLoperator2PL-operator or PersistentObjects, Generic-SetType. For dynamic (runtime) type checking support it is enough to realize the following set of features: Type2TypePattern, Query2String, DMLoperator2String, Strict-SetType.

The table of characterization of the known binding approaches by the features offered is presented below. The binding approach proposed by the author for PL to subject mediators binding is included in the table as the row denoted by «Synthesis».

Table 1. Characterization of well-known approaches

	LM	**QM**	**OM**	**QS**	**C**
ODMG 3.0	T2T	Q2S	TO	CCQ	SST
JDO	PL2T	Q2S, Q2P	PO	OCQ	GST
JDBC	T2P	Q2S	DML2S, TO	CCQ	SST
SQLJ	T2P	Q2PLQ	DML2O, PO	CCQ	GST
OCCI	T2T	Q2S	DML2S, TO	OCQ	SST
LINQ	PL2T	Q2PLQ	DML2O, TO, PO	CCQ	GST
Synthesis	T2T, T2P	Q2S, Q2PLQ	TO, PO	CCQ	GST

4 Brief characterization of impedance mismatch between DDL/DML and PL

To characterize the impedance mismatch problem in terms of the mapping features proposed we use a list of the basic respective problems extracted from the Kazimierz Subieta paper [13]:

- **Syntax (S)**: Programmer must work in two languages simultaneously. Same concepts may mean different things(for example, "=" in Java means assignment, while in SQL "=" means comparison).
- **Typing (T)**: Types in PL and types in QL may differ. Commutative mapping between types may be impossible or cause serious overhead. PL introduces static (compile time) type checking, but QL is based on dynamic type checking.
- **Binding phases and mechanisms (B)**: QL is based on dynamic (run-time) binding for names occurred in a query, while PL is based on static (compile and linking time) binding.
- **Name spaces and scope rules (N)**: PL and QL have different name spaces, for example from PL we can't use names occurring in queries and v/v. But usually program variables may parameterize queries and result of a query may be used as PL variable.
- **Collections (C)**: Semantics of DB collections and PL collections are different. DB collections are processed by queries and result is stored not in DB but in PL collection with its own syntax and semantics.
- **Persistence (P)**: QL processes persistent data (stored on a disc), while PL processes transient data (stored in memory). Objects in PL received from DB commute with objects in DB. So changes in PL objects must be reflected to DB.

- **Queries and expressions (Q):** Some queries and expressions syntactically may be similar but they have different semantics. For example, in QL 2+2 is a query, but it is also an expression of PL. A query cannot be a parameter to a procedure, but an expression can.
- **References (Ref):** To update, insert or delete data in DB some references to stored data are required. QL returns not references but collections (data). So DML references support (or some another facilities) are required in PL to manipulate data (e.g. Persistent Objects).
- **Refactoring (R):** Refactoring names can't be automatically applied to queries if they are represented as strings.

Table below shows which features of the binding characterization introduced are required to solve the impedance mismatch problems.

Table 2. Covering of the impedance mismatch problems by the features introduced

	S	**T**	**B**	**N**	**C**	**P**	**Q**	**Ref**	**R**
Type2Type (T2T)		Y	Y(w)		Y(w)				
Type2TypePattern (T2P)									
PL-Type2DDLType (PL2T)									
Query2PL-Query (Q2PLQ)	Y		Y(w)	Y			Y		Y
Query2String (Q2S)									
Query2QueryPattern (Q2P)	Y								Y
DMLoperator2PL-operator (DML2O)								Y	
DMLoperator2String (DML2S)								Y	
PersistentObjects (PO)						Y		Y	
TransientObjects (TO)									
Generic-SetType (GST)					Y(w)				
Strict-SetTypet (SST)									

Yes – means that a binding approach supporting this feature completely solves impedance mismatch problem. Yes(w) – means that impedance mismatch problem is solved if both features are supported. As it is shown in the table, to solve all impedance mismatch problems listed above a approach should respect the following set of features: Type2Type, Query2PL-Query, Persistent Objects, Generic-SetType.

5 Related work

The problem of binding programming languages with declarative languages (databases, subject mediators) is not new. Several well-known approaches are presented in Table 1. It's clearly seen that none of the projects solves all impedance mismatch problems.

ODMG 3.0 solves only Typing problem. OCCI solves Typing and Reference problems. JDBC solves only Reference problem. SQLJ does not solve Typing, Binding and Collection. JDO solves Syntax, Persistent, Reference and Refactoring problems. LINQ as SQLJ does not solve Typing, Binding and Collection problems.

Several papers were devoted to the comparison of various binding projects. For example, in [14] a qualitative evaluation criteria are presented that are quite similar to the characterization presented in section 3. The characteristics related to the impedance mismatch are a subset of the features considered in section3.

There are also several approaches which claimed that impedance mismatch problem is completely solved.

Main idea of the Sather approach [15] consists in using of the basic data structures of programming language to represent entities of the relational data model. It is assumed that by using one and the same type system for programming language and database, the impedance mismatch problem will be solved. This approach solves Syntax, Persistent, Reference and Refactoring problems.

In ARARAT project [16] authors eliminates impedance mismatch problem by offering template library whose objective is type safe generation of SQL. This template library is statically generated by database schema. Template library contains classes for query design, not for database type representation. Persistent objects and generic collections are not supported. As Query2QueryPattern approach ARARAT solves Syntax and Refactoring Impedance mismatch problems.

In Stack Base Approach (SBA) [17] all impedance mismatch problems are eliminating by means of inventing new self-contained query/programming language. Such imperative object-oriented programming language, in which there is no distinction between PL expressions and queries called SBQL. SBQL rejects any type checking mechanisms that originate from type theory, so the problem of static type checking is still actual.

In PhD thesis [18] author investigates the integration problem of programming and query languages for distributed object databases. Author uses the SBA for developing the database programming and query language called iDBPQL. Author extends SBQL approach by means of persistent objects, transactions, distribution, support of large amount of data (as the stacks are main memory based), type system support.

SBA and its implementations (SBQL, iDBPQL) constitute a promising technique providing a new foundation of QL-centric programming language instead of applying conventional programming language paradigms. It solves most (SBQL) or all (iDBPQL) impedance mismatch problems. In contrast to the approach presented in this paper, SBA invents a new query/programming language instead of solving impedance mismatch problem in terms of standard PL and QL (e.g. SQL).

6 Conclusion

A set of features to be used for characterization and evaluation of different approaches of PL bindings to information resource management systems is proposed. In terms of the features introduced, a characterization of well-known approaches to PL with DBMS binding is given. It is shown how a set of supported features should be selected to solve impedance mismatch problems. An adequate set of features was selected for an advanced approach for binding of a procedural PL with a subject mediators declarative specification language (row of Table 1 denoted by “Synthesis”). The set of features selected provides for complete representation of information model of subject mediators in PL as well as for overcoming of the most of impedance mismatch problems. The problems considered in the paper arise in the context of PhD work on the efficient organization of dispersed implementation of applications using various components of the mediation environment.

References

1. Kalinichenko L.A., Briukhov D.O., Martynov D.O., Skvortsov N.A., Stupnikov S.A. Mediation Framework for Enterprise Information System Infrastructures. Proc. of the ICEIS 2007. Volume Databases and Information Systems Integration. -- P. 246--251.
2. Friedman M., Levy A., Millstein T. Navigational plans for data integration // National Conference on Artificial Intelligence (AAAI) Proceedings, 1999.
3. Kalinichenko L.A., Stupnikov S.A., Martynov D.O. SYNTHESIS: a Language for Canonical Information Modeling and Mediator Definition for Problem Solving in Heterogeneous Information Resource Environments. Moscow: IPI RAN, 2007. - 171 p.
4. R.G.G. Cattell, Douglas K. Barry, et. al. The Object Data Standard: ODMG 3.0. Morgan Kaufmann Publishers, San Francisco, California
5. OCCI User Guide, http://download.oracle.com/docs/cd/B28359_01/appdev.111/b28390/toc.htm
6. Oracle Java SE Technologies - Database, http://www.oracle.com/technetwork/java/javase/tech/index-jsp-136101.html
7. Jim Melton, (ISO-ANSI Working Draft) Object Language Bindings (SQL/OLB), American National Standard, Information technology — Database languages — SQL — Part 10: Object Language Bindings (SQL/OLB), August 2003
8. JDO documentation, http://java.sun.com/jdo/
9. LINQ to SQL User Guide, http://msdn.microsoft.com/ru-ru/library/bb386976.aspx
10. Kalinichenko L.A. Methods and tools for equivalent data model mapping construction. Proc. of the International Conference on Extending Database Technology EDBT'90. LNCS 416. -- Berlin-Heidelberg: Springer-Verlag, 1990. -- P. 92-119.
11. Kalinichenko L.A., Stupnikov S.A. Constructing of Mappings of Heterogeneous Information Models into the Canonical Models of Integrated Information Systems. Proc. of the ADBIS 2008. -- Pori: Tampere University of Technology, 2008. -- P. 106-122.
12. Todd Millstein, Alon Halevy, Marc Friedman, Query containment for data integration systems, Journal of Computer and System Sciences, Vol.66, Issue 1, 02/2003, Pages 20-39
13. Kazimierz Subieta, Impedance mismatch, http://www.ipipan.waw.pl/~subieta/SBA_SBQL/Topics/ImpedanceMismatch.html
14. William R. Cook, Ali H. Ibrahim. Integrating Programming Languages & Databases: What's the Problem? www.cs.utexas.edu/~wcook/Drafts/2005/PLDBProblem.pdf
15. Jian Chen , Qiming Huang. Eliminating the Impedance Mismatch Between Relational Systems and Object-Oriented Programming Languages. in Proce. the 6th International Hong Kong Database Workshop, 1995.
16. Joseph (Yossi) Gil, Keren Lenz. Eliminating Impedance Mismatch in C++. VLDB, 2007.
17. Michał Lentner, Kazimierz Subieta. ODRA: A Next Generation Object-Oriented Environment for Rapid Database Application Development. ADBIS 2007, p. 130-140.
18. Markus Kirchberg. Integration of Database Programming and Query Languages for Distributed Object Bases. PhD thesis for the degree of Doctor of Philosophy in Information Systems at Massey University, 2007.