IRO-DB
An object-oriented approach towards federated and interoperable DBMS\(^1\)

Ralph Busse, Peter Fankhauser, Gerald Huck, Wolfgang Klas

Integrated Publication and Information Systems Institute GMD–IPSI
Dolivostr. 15, D–64293 Darmstadt, Germany

Abstract

Todays application scenarios need more and more access to information stored and distributed among multiple database management systems which have various underlying data models and which model even the same real world aspects differently with respect to structure and granularity. Therefore, a system is needed which addresses these problems, providing the means to integrate heterogeneous data sources from the design perspective and the data perspective. In this paper we present IRO–DB, an ESPRIT III project, which allows for the integration of heterogeneous object oriented DBMS and also relational DBMS. Using a common data model standard and providing for the necessary tools to realize the communication and access to the heterogeneous databases involved via interoperable/integrated schema definitions, IRO–DB is able to overcome todays deficiencies at a large degree.

1 Motivation

Todays application scenarios need more and more access to information stored and distributed among multiple database management systems. The existing databases are heterogeneous, i.e. they model information with differences in data model, naming, scaling, granularity, structure and semantics, and they are autonomous, i.e. they can evolve over time and may not be modified to fit the needs of new applications with respect to existing applications based on them.

To overcome these problems, applications need access to the distributed databases in a homogenized fashion, allowing meaningful access and exchange of their data. These goals can be achieved by making the databases interoperable, providing the means to model relationships between distributed data, to enrich data structurally and semantically, and to overcome inconsistencies.

\(^1\) This work is initiated and partially supported by the European IRO–DB ESPRIT III project (P8629). The project is developed in cooperation with GMD, GOPAS, IBERMATICA, EDS, GRAPHAEL, INTRASOFT O2 Technology and laboratoire PRiSM.
The IRO–DB project (Interoperable Relational and Object–Oriented Databases – ESPRIT–III P8629) proposes a client–server system architecture based on three layers to achieve interoperability between preexisting databases: The Local Layer provides the necessary mappings from individual local database schemas modelled in the native database’s data model into a common data model and is also responsible for the correct mapping of the stored data. The Communication Layer provides homogeneous remote access to single local databases by means of the common data model. The interoperable layer provides transparent, integrated access to several local databases. The common data model chosen as the basis for all these layers is ODMG/ODL (Object Database Management Group/Object Definition Language), an upcoming standard to be released as part of the work of OMG’s (Object Management Group) database initiative to extend their Interface Definition Language (IDL) by object oriented database concepts.

The goals addressed within IRO–DB can be summarized to:

- Support of a standardized Object–Oriented (OO) Data Model for modeling
  - relational and OO schemas of local distributed DBMS and
  - interoperable schemas
- Support of a standardized OO Query Language for local and interoperable schemas
- Transparent access to distributed data
- Tools for assisting in the integration process
- A standardized Application Programming Interface (API)

The remainder of this paper is organized as follows. Within chapter 2 we briefly introduce the general IRO–DB architecture, identifying the components of the system. Chapter 3 describes in more detail the functionalities to be realized, according to the different layers and their components. Chapter 4 focuses on the database integration approach as one major contribution to IRO–DB, describing the provided tool and used schema integration methodology. The remaining chapters 5, 6 and 7 deal with conclusions, give acknowledgements and provide for some related literature.

2 IRO–DB Architecture

Within this section, we present a suitable architecture for IRO–DB with the main functional components. The proposed architecture derives naturally from a layered client/server architecture (s. Figure 1). It consists of three layers, the Interoperable Layer, the Communication Layer and the Local Layer.

The Interoperable Layer provides applications with a C++ API which is compliant to the CORBA ODMG/ODL C++ binding. An ODL parser creates interoperable schema definitions from textual descriptions. The OQL parser transforms OQL queries into an internal query representa-
tion which is used by the Global Query Processor. The Global Query Processor optimizes the given query and translates it into queries against the local databases. The Global Transaction Manager provides for nested transaction semantics at the global level, using basic transaction primitives supported by the Local Layer. The interoperable DBMS (ONTOS) is used to maintain interoperable schema definitions, export schema definitions of the Local Layer databases and to store intermediate query results in a suitable way using an external storage manager which keeps track of the necessary mappings from local objects into global available interoperable objects.

The Integrators Workbench is a runtime independent design tool which allows for the creation and maintenance of interoperable schemas using the export schema definitions stored in ONTOS. It is also responsible to generate the necessary mapping and localization information to be able to transform global queries into queries against the underlying local databases.

The Communication Layer provides all necessary services for remote database access needed at the Interoperable Layer using an object oriented adaption of the Call Level Interface (CLI) speci-
The communication services are implemented using the standards XDR (EXternal Data Representation) and RPC (Remote Procedure Call) established in the UNIX world.

The Local Layer consists of the different available DBMS and a Local Database Adapter for each database. The adapters provide for ODMG compliant access to the local databases via suitable mappings from the available local schemas modelled in the native DBMS data model into an ODMG/ODL export schema definition and the mapping of OQL queries into the locally used query language.

3 IRO–DB Layers

This section is intended to describe in more detail the different layers and their components.

3.1 Local Layer

The purpose of the Local Layer (see Figure 2) is to provide homogeneous access to different database management systems. This comprises two tasks: (1) The construction of an ODMG compliant export schema on top of the local schema, together with the necessary mappings. (2) On this basis the translation of OQL queries and updates on the export schema into local requests. This translation includes parameter and result conversion between the local datatypes and the ODMG datatypes. A special facet of this conversion is the provision of correct global object identifiers, which need to be mapped to local objects or value sets. Finally, the Local Layer has to invoke the query interpreter of the local system in order to evaluate the translated query. This evaluation must be integrated into the global transaction management.

This functionality is achieved through Local Database Adapters (see Figure 3). Each such adapter provides for functionality which is specific for a particular database management system, and
for generic functionality which is common for all adapters. This way, the Local Layer provides a "standard local access product" where each single local database can be accessed as an ODMG compliant database.

**Generic Functionality**

The generic functionality comprises all ODMG–specific services:

The "export schema repository" contains the export schema description. It serves as a data dictionary for the ODMG representation of the local schema. Although the contents differ from site to site, the implementation of the repository itself, i.e. the representation and the interfaces, is generic.

The "generic mapping repository" contains all information that is necessary to convert data between the local representation and the export representation. It contains only a generic description of the mappings. The implementation will be provided by the site specific mapping toolbox.

The "export schema manager" is responsible for the maintenance of both repositories. It inserts the site specific information at database creation time, and provides run time access to the stored information.

Finally the "OQL parser" provides the external interface to the Local Layer. Based on the information stored in the repositories, each OQL query against the export schema is translated to a syntactic query tree against the local schema. Evaluation of the query tree is done in the specific part.

**Specific Functionality**

The specific functionality needed is on the one hand the translation of data types and data of the local database into ODMG data types and data and vice versa and on the other hand the translation of OQL query trees into subqueries against the local database.
The "mapping toolbox" implements all basic routines for converting site specific datatypes. In a relational system, for example, the ODMG datatypes have to be converted to table values, and tuples, that represent an object in the export schema, have to be provided with an object identifier in a standard format which may consist of a site string concatenated with an identification string.

The "local mapping adapter" provides the final link to the local database. It transforms the syntactic query tree generated by the OQL parser into local queries against the underlying database system. (E.g., SQL queries in case of a relational system.)

### 3.2 Communication Layer

The communication layer is intended to realize the transfer of objects, OQL queries and their results to and from a client site to a local server database by OORDA (object oriented remote database access) services. These services are accessible to applications by an extension of the SQL access groups CLI (Call Level Interface) able to deal with OQL queries, transactions and objects as such, providing also cursor concepts to deal with large query results or large objects.

The Communication Layer realizes the transfer of objects, OQL queries and their results by object–oriented Remote Data Access services with the necessary communication translation. Applications can access these services using an extension of the SQL Access Group Call Level Interface (OO–CLI) extended to support objects (i.e., OQL). The OO–CLI allows the Interoperable Layer to interact with Local Layers. Furthermore, applications which want direct remote access to (some of) the local databases can use OO–CLI also.

The Communication Layer defines a client/server architecture. The Interoperable Layer/remote applications refer to the clients and the Local Layers refer to the servers. The clients have to establish a connection to the Local Layers via the Communication Layer which in turn creates a communication environment used for one particular connection. When a connection has been established, the clients may access the local databases and retrieve data.

The Communication Layer offers five basic services:

*Resource Management*

The resource management is responsible for creating and deleting the communication environments needed by the clients and offers also the routines to establish the connections with the different Local Layers.

*Execution Management*

The execution management is responsible for transferring the OQL queries from the clients to a local layer database. It supports dynamic and static OQL statements. A static statement is a complete OQL query, not referring to any runtime parameters, i.e., a string. A dynamic statement is an OQL query which contains parameter markers that indicate links between the dynamic statement and variables bound to it. OO–CLI routines resolve these links at runtime. Now each OQL statement is declared as a string parameter and can be given to the Local Database Adapters which are responsible for parsing and mapping this query to their own local query language. Then, when the query has been evaluated by the local database server, the correspondent
Local Database Adapter has to map the result into the ODMG data types, and to provide descriptions about the structure of result objects, their attribute names and data types.

**Result Management**

The result management has to provide on the one hand OO–CLI routines to access the result type of an evaluated query and on the other hand OO–CLI routines which allow for the access/navigation of the result data in a uniform way, using an adapted OO–CLI version of the CLI cursor concept, e.g. providing access to attributes of result objects instead of returning a column value in a table.

**Transaction Management**

The transaction management is responsible for propagating client transaction requests to the Local Layer and returning transaction aborts to the clients.

**Transfer of Export Schemata**

The last service allows the export schema transfer between Local Layer and Interoperable Layer.

The development of the OQL/CLI interface is based on the Remote Procedure Call (RPC) approach which is well–adapted for a client/server architecture. All transfers between the ROA of the interoperable and the ROA of the Local Layer is done according to the eXternal Data Representation (XDR). The XDR representation has been chosen because it is considered as a standard between UNIX workstations.

### 3.3 Interoperable Layer

The Interoperable Layer contains all the components necessary to allow for integrated application access to multiple remote databases. These components can be divided into two major categories: components supporting an interoperable database management system (IRO–DBMS) to maintain interoperable schemata and data and a tool to assist the design and maintenance of interoperable schemata. We describe in this subsection the functionality of the IRO–DBMS. The design tool, the Integrators Workbench, is described separately in Chapter 4.

The IRO–DBMS covers the functionality of a real DBMS which is viewed as the only DBMS to applications accessing the interoperable schemata. IRO–DBMS functionality deals with global transaction management, global query processing, management of the global data repository and access to the remote databases’ data and schema descriptions via an ODMG compliant DBMS.

**Application Programming Interface (API) generation**

This function provides the generation of an ODMG compliant C++ API used by applications to access the interoperable ODL schemata and their data and the generation of the necessary method implementations from the available mapping informations.

This functionality can be made available either as part of the user interface of the Integrators Workbench (see below) or as a separate program to be invoked.
The global transaction manager

The global transaction manager realize a nested transaction protocol as proposed by ODMG. These functions use the localization information in the global data repository to realize a transaction upon multiple remote databases (and the database at the interoperable layer) calling basic transaction management services available at the communication layer (and at the database at the interoperable layer).

Applications acting upon interoperable schemata use the provided functionality according to the ODMG C++ binding proposal, which defines a C++ class to support transaction semantic. This class is part of the interoperable schema independent C++ API.

In addition, there are finer granular functions necessary which allow for the transaction control when executing a global query against an interoperable schema resulting in autonomous subqueries against different remote databases.

The global query parser and global query processor

The global query parser takes an OQL query against an interoperable schema in ASCII representation, parses it and generates an Object Expression Tree (OET). The global query processor uses the mapping information in the global data repository to break query expressions referring to the interoperable schema down into subexpressions referring only to the export schemata of the remote database sites. This transformed OET is given to an optimizer which optimizes the OET due to the informations stored in the OET. Finally the optimized OET is given to an execution component who is responsible to send the OQL subqueries of the OET via the Communication Layer to the local databases and to compose the final query result from the received subquery results.

The global data repository

The global data repository has to provide

- access to the ODMG export schemata descriptions available at the Local Layer,
- access to the ODMG description of the interoperable schemata,
- access to the mapping information for the interoperable schemata and
- access to localization information where export schemata come from.

It is used by the Integrators Workbench as storage component for interoperable schema descriptions and associated mapping definitions. The global query processor and parser also has access to this component for syntax and type checks and to be able to transform the OETs.

The integrated object manager

The integrated object manager is responsible for the intermediate storage of partial query results from subqueries against local databases and guarantees the transparent access to the retrieved objects via one to one mapping objects available for the application. (For the first prototype, IRO–DB uses the OODBMS ONTOS for this purpose.)
4 Integration aspects within IRO–DB

4.1 Integration Methodology

While complete integration is neither realistic nor required, a methodology is still necessary to dynamically integrate those portions of local databases which are needed to create a sufficiently broad basis for the application domain to lower the burden for the application designer of understanding in detail structure and semantic of the external schemas.

Two major categories of schema integration methodologies have been proposed until now. On the one hand exist strict integration methods which are using upward inheritance, correspondence assertions and rules [1][2] to achieve integration. They are sound, but at the same time intolerant and restrictive with respect to the real semantic modelled in the schemas which shall be integrated. On the other hand heuristic techniques using best paths, spanning trees and structural similarity [3][4][5], can handle some of the arising semantic problems more pragmatically, but lack in soundness.

The declarative integration methodology developed within IRO–DB at our institute [6] extends strict unification techniques in such a way that structural heterogeneity, caused by modelling the same real world aspects by different means of the data model (semantic relativism), can be dealt with to a large degree. Additionally, the methodology puts specific emphasis on the treatment of differences in structural granularity introduced by the different levels of detail with which schemas may represent the same real world aspect (perspective).

Our unification approach is based on augmentation rules applied to corresponding parts of the schemas (subschemas) participating in the integration process. These corresponding parts are expressed by path correspondences, where a path is defined as a connected subgraph without circles from one vertex to another vertex in a schema (in the simple, currently used data model, properties and classes are vertices, and references from a property to it’s domain class and sub-class relations are edges of a schema graph).

The result of unification by augmentation rules are schema constituents in the integrated schema which retain as much of the original subschema information as possible, even if the information is only modelled by some of the schemas to be integrated.

Integration methodologies based on abstraction use a contrary strategy. They are only able to integrate those schema parts which have a common abstraction. Real world aspects modeled only by one of the schemas to be integrated can not be dealt with, because they do not have a corresponding counterpart in the other schemas. Therefore, these methodologies usually are not able to create an integrated schema preserving all the aspects of the schemas participating in the integration process.

Users affirm correspondences between parts of heterogeneous schemas regardless of structural differences between them. They are assisted in finding additional correspondences, and are guided in turning these correspondences into operational mappings between the integrated schema and the corresponding parts. The declarative definition of correspondences may lead to ambi-
guities or conflicts during the integration process which are not assumed to be solved automatically, but in turn need user interaction to resolve them properly.

The presented methodology can assist top down as well as bottom up semantic integration. Bottom up integration starts with the local schemas to be integrated and successively creates a partial integrated schema which suits some application domains. Top down integration can take advantage from the existing requirements of application domains which must be fulfilled by the integrated schema, i.e. the application already defines its integrated schema, and the methodology provides the means to find the appropriate mappings to the underlying local schemas.

### 4.2 Integrators Workbench

The Integrators Workbench is the tool at the interoperable layer responsible for the creation and maintenance of interoperable/integrated schemata.

![Integrators Workbench Architecture](image)

Figure 4: Integrators Workbench Architecture

It provides a graphical user interface to display ODMG schemata (export schemata available at the Local Layer and the interoperable schemata) and it provides the necessary interaction for schema integration, based on an object oriented, direct manipulation paradigm of the visualized...
objects. The user interface is build around the integration components for consistency checking and the generation of interoperable schemata.

The Integrators Workbench accesses the global data repository. It uses a ODMG/C++ interface to the data repository to retrieve the components of the available ODMG export schemata and to store and access the interoperable ODMG schemata. These interoperable schema descriptions contain additional information for their mapping to export schemata.

Additionally, the graphical interface provides the functionality for a convenient layout of the schemata.

### 4.3 Integration Methodology example

In the following we discuss the main parts of our methodology using a scenario of two example schemas to be integrated. A simplified representation of the two schemas is shown in Figure 5. Classes are represented by labeled rectangles with bold lines, properties of the classes by finer lined labeled rectangles. The edges between properties and classes are undirected, edges with an arrow represent a reference from the property to its domain class.

![Figure 5: Two schemas and a simple augmentation proposal](image)
To illustrate the augmentation principle of the methodology we define the following two correspondences (cf grayed parts of the figure).

Path in first schema: Path in second schema:
Person–street Person–Address–ADDRESS–Street
Person–city Person–Address–ADDRESS–City

The result of this partial integration step leads to a partial integrated schema which reflects the finer granularity modelled in the second schema. It introduces the class ADDRESS to model a person’s address despite the fact that the first schema does not have a notion of ‘address’ objects.

A methodology based on abstraction rules will not be able to preserve all schema information and the modelling granularity of the investigated schemas for this particular case. Its integration proposal most probably will reflect only the design used in the first schema. Thus, the concept of an address as an object with its own identity is no longer available and applications using the integrated schema can not profit from the finer granularity used in the second schema.

The proposed augmentation has also to be reflected at the instance level. In this case, the mapping from person objects of the first schema (where address objects are not modelled) can be achieved by introducing ‘virtual’ address objects in the integrated schema for object instances coming from the first schema. The ‘virtual’ object’s lifetime is limited to that of the associated person object and the deletion of a person object also implies the deletion of the virtual address object.

We extend this example by adding the following three correspondences:

Path in first schema: Path in second schema:
Person<inChargeOf–Salesman–salary PERSON–Salary
Person–products PERSON<–Client–SALE–Product
Person<inChargeOf–Salesman– PERSON<–Client–SALE–employed->Company Merchant–Company

In this case our methodology can not find a proper structural augmentation for the given correspondences. This is due to the fact that the first and second correspondence lead to the intermediate augmentation shown in Figure 6, but

![Figure 6: Augmentation along different paths example](image)

the third correspondence requires that the augmentation rule must follow the path **Person<inChargeOf–Salesman** for the first schema and **PERSON<–Client–SALE** for the second schema. This contradicts to the condition that a path correspondence must be representable by exactly one path in the integrated schema. This inconsistency can only be resolved by involving the user in the integration process. Before the process is able to continue he/she has to decide which of the above three correspondences should not be considered any longer.
In our example it seems to be very unlikely that the property Salary of PERSON in the second schema models the same real world aspect as Person<–inChargeOf–Salesman–salary, so that the first correspondence assumption can be removed.

Now the remaining two path correspondences have to be merged. In this example, three possible combinations for an augmented subschema exist for the two paths (Person ... Company) in the first schema and (PERSON ... COMPANY) in the second schema (see Figure 7).

Again the user has to decide which of the three possible cases the integration process should use. Merchant can correspond either to employed (1) or to Salesman (2) or to none of both (3). Choosing the second case leads to the final partial integrated schema shown in Figure 8. The gray parts show the augmentations with respect to the first schema and the hatched parts are augmentations with respect to the second schema.
5 Conclusion

In this paper we present IRO–DB, an ESPRIT III project, aiming at the integration of object-oriented DBMS and relational DBMS. The proposed architecture of the whole system and the usage of existing and developing standards allows for the easy integration of existing DBMS systems, also for these not participating directly in the project. The usage of a standardized data model (ODMG/ODL) provides for a uniform access across the different layers of the system.

The Local Layer is intended to make databases ODMG/ODL compatible and to provide an uniform interface for remote access via the Communication Layer. The Communication Layer mainly implements the main services needed by the systems client/server architecture and provides clients with the necessary interfaces to access local databases via remote connections. The Interoperable Layer provides for a transparent, integrated access of the local DBMS via integrated schemas, using the Communication Layer facilities.

One major contribution to the project are tools developed for the creation and maintenance of integrated schemas. The Integrators Workbench is a graphical development tool for the integration of existing schemas based on a flexible integration methodology which performs integration via augmentation rules.

The initial specification of the architecture and functionality will be finished soon. The detailed specification of the layer components and first prototypes are undergoing. The time schedule for the whole project plans the project to be completed end of 1996. Further aspects like performance evaluation are done during a test phase for a real application scenario, when the initial prototypes will be finished.

6 Acknowledgements

We thank the participating project partners EDS, Gopas (Ontos), Ibermatica (Ingres), Intellitic (Matisse), Intrasoft and O2 Technology (O2) for their contributions to the architecture and functional specifications and their agreement to the presentation of IRO–DB within this workshop.
7 Literature


