An ORM-Driven Implementation Framework for Database Federations

Herman Balsters and Bouke Haarsma

University of Groningen, The Netherlands
h.balsters@rug.nl, b.h.haarsma@student.rug.nl

Abstract. Database federations are employed more and more in situations involving virtual and integrated information on demand, e.g., real-time integration of two databases. Heterogeneity in hardware and software platforms, as well heterogeneity in underlying semantics of participating local databases, makes it a hard challenge to design a consistent and well-performing global database federation. The ORM modeling framework allows not only for precise modeling of a data federation, but also hosts tools for reverse engineering, enabling local databases to recapture their intended meanings on a linguistic basis. We will demonstrate how ORM models together with reverse engineering techniques can be used in combination with actual, industrial-strength implementation platforms to develop semantically consistent and high performance database federations.

1 Introduction

Many companies are confronted with a large number of internal and external information systems that somehow have to be managed in a uniform and coherent fashion. Companies, e.g., merge in a legal and commercial sense, but merging their data is deferred, or even not done at all. Furthermore, the number of on-line systems providing data is ever-rising. Many companies are involved in building integrations of their legacy systems in order to obtain a coherent overview of available data. Most integration overviews of data are constructed ad-hoc and manually, often resulting in error-prone and time-consuming practices.

A major problem in integrating data is caused by lack of semantics. In order to understand the combined collection of all of the data available in the participating local systems, there has to be consensus on the meaning of the data. Naming conflicts, different formats in data, as well global identification of local data objects, can make life very difficult for the designer of an integrated information system. Ideally, the data sources should be combined automatically into a so-called federation. Such an automatically generated federation should provide the desired real-time overview, outlining the business data needs. Federations exist with a wide range of approaches. A number of papers [2,3,6,10,16,18] have been devoted to offering a conceptual framework for the integration of a collection of heterogeneous data sources into one coherent federation. A major feature common to many of these approaches is the use of the concept of *Global as View* ([18]). The federation is said to be *virtual* as the federation constitutes a view on a collection of existing local databases. The resulting

R. Meersman, P. Herrero, and T. Dillon (Eds.): OTM 2009 Workshops, LNCS 5872, pp. 659–670, 2009.

federation does not involve altering these collaborating sources; the great advantage of not having to change these local sources is that existing applications on those systems can also be left unchanged.

Only after establishing common semantics, can the local data models adequately be transformed into a common global model This is why modeling languages based on fact-orientation, such as ORM([13,15]), FCO-IM([4]), NIAM([14]), and OSM ([9]), are good candidates in this particular modeling environment, because they all provide the linguistic basis providing for close and correct interaction between the non-technical domain expert and the information modeler. Other popular modeling languages (e.g., ER([7]) and UML([20])) are lesser qualified candidates, especially due to lack of sufficient expressiveness, but also to this lack of providing a proper factual, linguistic basis. ORM, furthermore, is grounded in first-order logic ([1,11]), offering ORM a clear mathematically-based semantics. The ORM language also offers well-equipped tooling facilities ([8,12]); reverse-engineering, in particular, is a very useful facility in re-creating the factual, linguistic basis of legacy databases.

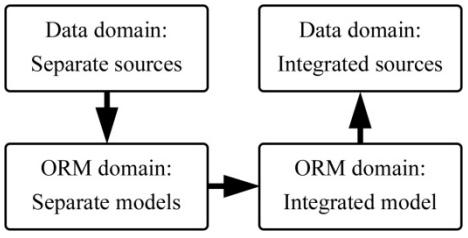
This paper deals with offering an implementation framework exploiting ideas taken from [2,3] and used to offer a conceptual design of a database federation. This conceptual design method is used in combination with the IBM Infosphere Federated Server (IBM-IFS, cf. [5]). The IBM-IFS techniques of nicknaming and wrapping ([5]), and the reverse- and forward-engineering techniques of the ORM tools, combined with the methodological framework of [2,3], results in an effective implementation platform for development of semantically consistent and high-performance database federations.

This paper is organized as follows. In section 2, we describe our general approach –called the ORM Transform– in combining the ORM design method with IBM-IFS. Section 2 also describes technique employed in IBM-IFS to prepare local data sources for federation. Section 3 discusses reverse engineering. Section 4 offers a description of schema integration in ORM. Section 5 treats generation of view definitions for actually implementing the global federated database. This papers ends with some conclusions and suggestions for future research.

2 The ORM Transform: A Laplace Transform Analogue

The Laplace Transform ([19]) is a widely used integral transform to solve difficult differential equations. Solving the differential equation by hand would be difficult, error-prone and even impossible. The Laplace Transform works by taking the differential equation, transforming it to an easy-to-solve linear equation. Solving this linear equation also gives the solution for the differential equation. The same applies for the method described in this paper. Solving the problem of federating data by hand proves to be very difficult and error-prone. Our method takes the sources and transforms them into ORM models. These models can be federated into a single ORM model based a clearly-defined and strict procedure ([2,3]). Finally, by transforming this global federated model back to a relational model, we obtain our desired federation.

ORM Transform



The complete framework is built upon six processes, cf. Table 1. The processes are executed in succession, starting with the top, and ending at the bottom. The output of each process is the input for subsequent processes. These processes will be covered in later sections of this paper.

Table 1. Processes of the Framework

Process	Input	Output
Source Definition	Existing databases	Selection of tables from the existing databases
Nicknaming	Selected tables	Nicknames
Reverse Engineering	Nicknames	Local ORM schemas
Schema Integration	Local ORM schemas	Global ORM schema
Relational Model Generation	Global ORM schema	Relational model
View Generation	Nicknames Global ORM schema Relational model	DBMS specific DDL with view definitions and table definitions for external data

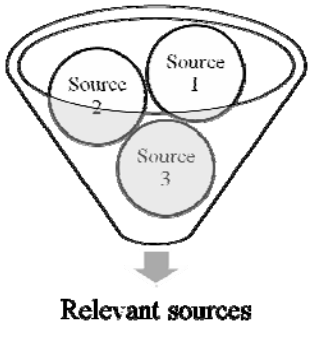


Fig. 1.

A federated database is built upon several collaborating sources. Not all the data in these collaborating sources is used in a federated database. There can be several reasons to not include the data in the database. Not all data, e.g., is relevant to the federation (Needless information), and sometimes we are dealing with duplicates of data could be stored in several sources (Redundant information). We have to filter out the needless and redundant information, cf. Figure 1. The filtering happens on the level of the participating database tables. A decision on which tables to include, is not taken by the federation designer alone; often consultation of a domain/business expert will be necessary. This process results in a list of tables to be included in the federation.

2.1 Example: Company and Credit Database

This example concerns a company with a customer database and an external credit registry status database. The customer database contains information about the company's customers, such as name and address. The external credit registry status database (from now on referred to as 'credit database') is a country-wide maintained

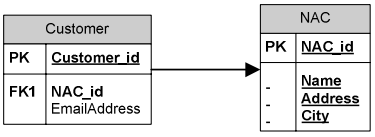


Fig. 2. Relational Model for Customer Database

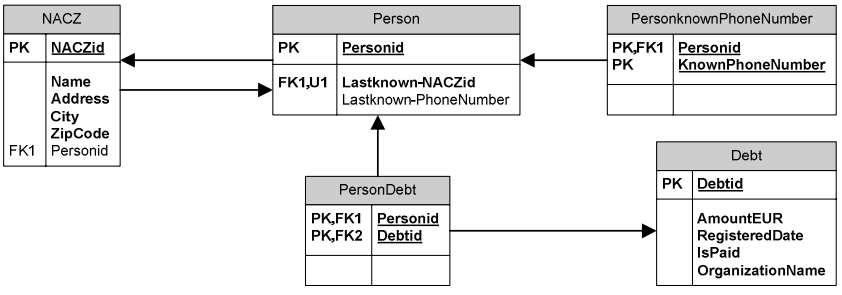


Fig. 3. Relational Model for Credit Database

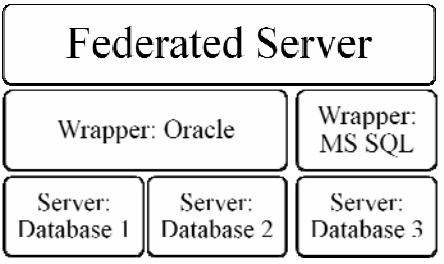
database containing loans and mortgages from financial providers to end customers. The company desires an overview the loans of their customers and prospects. The credit database can be queried through some SQL dialect. The preferred solution is to obtain a federation of this system with their own customer database. In our example, we assume that we already filtered out needless and redundant data in the local data sources. The legenda for figures 2 and 3, below, is as follows: NAC stands for Name Address City; PK stands for primary key; FK for foreign key (the arrow denotes a foreign key reference).

The tables of the chosen sources are only locally accessible. In order to play their part in the federation they have to be globally accessible. This is done through the process of ***nicknaming***. This process includes the definition of Wrappers, Servers and User Mappings, and Nicknames ([5]). Wrappers facilitate the querying of a local source in a SQL dialect native to the federated system. IBM-IFS uses its own DB2 SQL dialect on the global level of the federation, and provides wrappers for ODBC, Oracle, MS SQL, etc.

Having identified wrappers, servers and databases, we now have to make our local data globally available. This is done by creating nicknames within our federated system, as aliases for our local tables. Having created these nicknames, the federated system takes care of transport between the local servers and the federated system. Since different databases can have tables with same name, original table names are overridden by a nickname (e.g., by prefixing the name by the (abbreviated) server name).

2.2 Relation of Wrappers to Servers

In our example, the customer database is available at hostname customers.local and is implemented in an Oracle database. In the federated server we add the wrapper



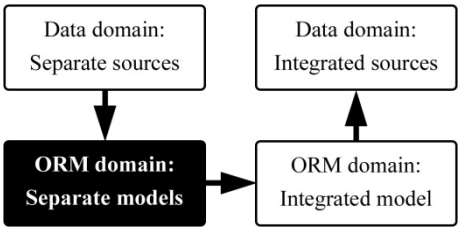
'Oracle', define a server with the given hostname and the user credentials. The credit database is available at the hostname credit.otherparty.external and is implemented in an MS SQL database. In the federated server, we add the wrapper 'MS-SQL', and define a server with the given hostname. We then create local nicknames as follows:

- company_customer, server: company, table: customer
- company_nac, server: company, table: nac
- credit_person, server: credit, table: person
- credit_nacz, server: credit, table: nacz
- credit_debt, server: credit, table: debt
- credit_person_debt, server: credit, table: persondebt
- credit_person_phone_number, server: credit, table: personknownphonenumber

Our federated server now contains 7 virtual tables:

- company_customer (id, nac_id, email)
- company_nac (id, name, address, city)
- credit_person (id, last_known_nacz_id, last_known_phone_number_id)
- credit_nacz (id, name, address, city, zip_code, person_id)
- credit_debt (id, amount, registered_date, is_paid, organization_name)
- credit_person_debt (person_id, debt_id)
- credit_person_phone_number (person_id, phone_number)

3 Reverse Engineering



In order to integrate the models, we extract the models from the existing databases. We assume that no ORM models of these databases exist, and that we are forced to create these models in retrospect. Recreating the models is done by *reverse engineering*, as opposed to *(forward) engineering* where you first design and then build the system.

When a database is created, chances are that it was not built according to a conceptual schema, with only the logical schema of the database at one's disposal. The database evolves during its lifetime; attributes and tables were added, changed or discarded, often leaving us with a rather complex database without formal documentation (semantics). Reverse engineering uses the existing database as its input and generates a conceptual schema providing the desired semantics. This conceptual schema can be optimized both in the sense of semantics as in performance. Forward engineering is then used to re-implement the improved schema.

Reverse engineering is implemented in the two major tools supporting ORM ([8,12]). After extracting the schema from the database, the schema is analysed, and

the semantics of the schema is reconstructed. As an example, assume that the original databases in our Company was an Oracle database. Reverse engineering could then have resulted in the following ORM model.

Such ORM models use the same generic name for the relations involved; i.e., has/is-of. Such generic names can be made less general -and possibly more accurate- by discussing with a domain expert how these names could be altered, and become more tailor-made to the situation at hand, hence providing a more accurate semantics.

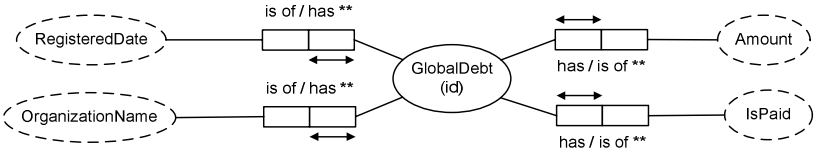
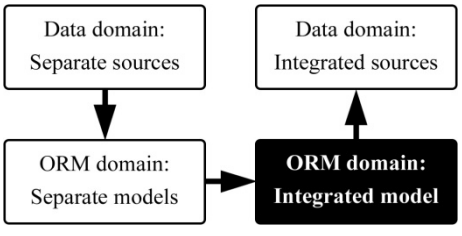


Fig. 4. ORM Model for Company Database

4 Schema Integration



In order to complete the framework and to obtain an integrated system, we need an integrated model. The integrated model results from combining local models; the resulting integrated models have to satisfy certain completeness and consistency requirements in order to reflect correct semantics of the different local schemata on the global integrated level. In constructing such a semnantics, we are faced with the problem of semantic heterogeneity, or differences in meaning attached to shared data. This problem can be split up into four main sub problems: global identification, followed by homonyms/synonyms, data conversion, default values and missing attributes.

4.1 Introduction of Global Identifiers

Global identification refers to the case where one single object in reality can be defined by two (or more) local objects. These corresponding object types often partially share a common structure. In interest of our integration, this shared common structure should be integrated into a global object:

1. Introduction of the global object. This global object contains the shared common structure, as found in the local objects.
2. Sub/supertyping is applied to the global (supertype) and local (subtype) objects.
3. The global object is assigned a global identifier. Introducing a new identifier allows local objects to be identified by their local identifier.
4. Introduce a discriminating role value on the global level. Not all global objects are defined by all their subtypes. Therefore a discriminating role is introduced to define all the roles the global object plays.
5. Make the correspondence explicit in the object populations (extra semantics needed, see next paragraph).
6. Show the correspondence between local and global identifiers.

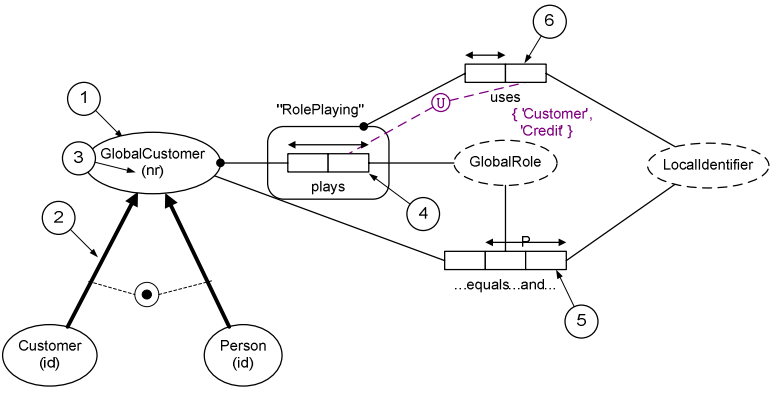


Fig. 5. ORM Design Pattern for a Global Identifier

4.2 External Data

How do we know which object in one population corresponds to another object in the other population? Unless the objects share some global identifier, it is impossible to decide on this matter by looking at the data alone. The business/organization will have to provide these data, also called *external data*, since such data has to be provided from some external source, i.e., outside the specified source databases. Such external data consists of a simple extra base table, called a data-reference table. One of the major functions of a federation is to locate and reason about local objects sharing a global identifier. Local objects sharing a global identifier are objects existing in two (or more) populations which –on the global level- refer to one and the same object. For example, in Table 2, a person, in physical reality, could be both a Customer and Credit person.

We note that once we have the extra data-reference table as a base table, depicted by the ternary predicate **.. equals .. and .. in** figure 5, we can derive, the relations play and uses in figure 5. These derivations are employed to make a sharp distinction between the original, physically stored data (such as the data-reference table, Table 2 above), and data on the level of the federation which is completely virtual (i.e., consists entirely of derived data).

We introduce an entity type called 'DebtCustomer' indicating the intersection of the populations of Customer and Credit persons. The newly introduced entity type can be thought of as a derived subtype; its existence can be reasoned from the supertype playing roles 'Customer', as well as 'Credit'. Next, as shown in Figure 8, a property can be assigned to the newly introduced entity 'DebtCustomer'.

Table 2. Local objects sharing a global identifier

Global_entity_identifier	Source_name	Local_identifier
1	Customer	13
2	Customer	17
2	Credit	31
3	Customer	21

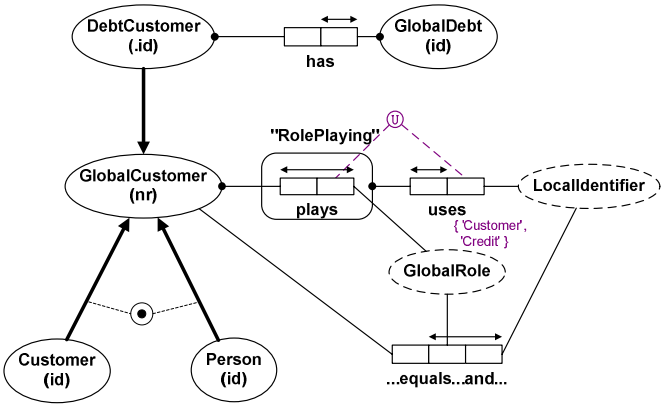


Fig. 6. ORM Design Patter for a Shared Global Identifier

We could complete the schema integration by defining the roles that 'GlobalDebt' entity participates in, as depicted in Figure 7, below.

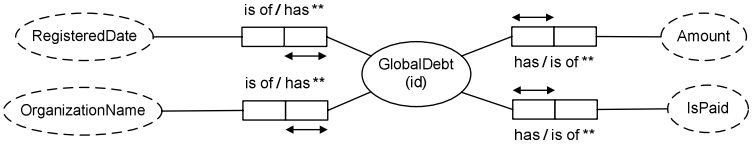
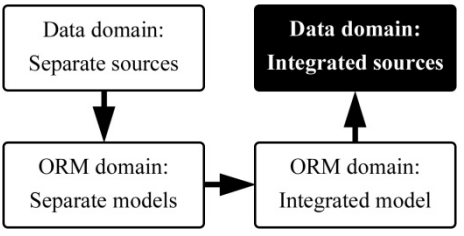


Fig. 7. Definition of Federated Debt

4.3 Remaining Problems

In [2,3], various techniques have been introduced to deal with problem categories of so-called homonyms/synonyms, data conversions, default values and missing attributes. Conflicts due to homonyms are resolved by mapping two same name occurrences with different semantics to different names in the integrated model. For example, suppose that Customer phone number means home number and Credit phone number means office number. These local names (assumed syntactically equivalent) are to be given different names in the global schema in order to make explicit the difference in semantics. Synonyms are treated analogously, by mapping two different names (with the same semantics) to one common global name. Conflicts due to conversion arise when semantically related types are represented in different units or different granularity. These conflicts are solved by applying a suitable conversion function, creating uniform representation of the data on the global level. We will refrain from offering a general treatment of these techniques, since we have only limited space and aim to focus on our implementation framework. We therefore refer to [2,3] for more information on handling homonyms/synonyms, data conversion, default values and missing attributes in a more general, theoretical framework.

5 Relational Model Generation and Views



The final step in the framework is the transformation from the integrated model to the integrated sources. This final step is split into two processes within the framework. The first process is to generate a relational model from the integrated model; the second process is to create views on this relational model.

5.1 Relational Mapping Procedure (RMAP)

Forward engineering uses a procedure called the Relational Mapping Procedure (RMAP), transforming ORM models into relational models. This procedure, in our example, results in importing the integrated model in VEA (Visio for Enterprize Architects, [12]), and building the project, resulting in the relational model of Figure 8). Note that both 'RolePlaying' and 'GlobalCustomer equals GlobalRole and LocalIdentifier' look identical. VEA merged the two derived predicates of the global identifier into a single table called `RolePlaying'. We will implement this table as a view definition later on in the framework.

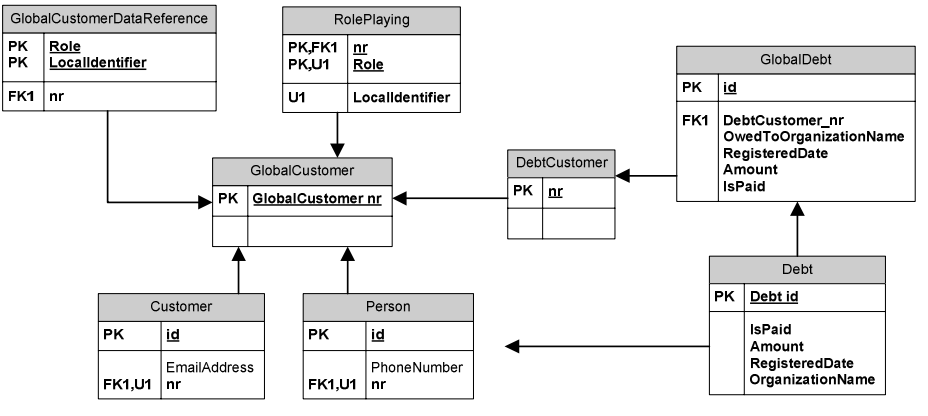
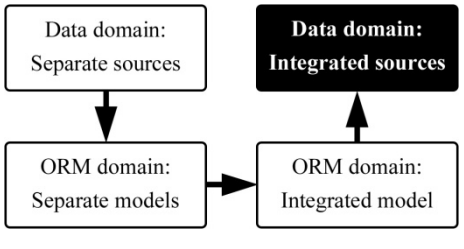


Fig. 8. Relational Model of Integrated Model

5.2 View Generation



The ultimate goal of the framework is applying *Global as View*, eventually yielding the desired federation. Completing the previous step (cf. figure 8) leaves us with table definitions of the generated relational model. This step is devoted to creating view definitions from these table

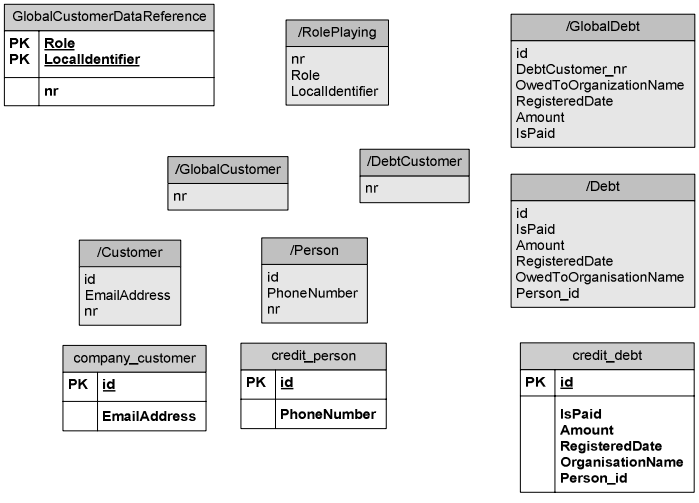


Fig. 9. View Definitions

definitions. For each table –not being a base table– in our relational schema, we have to create a view definition. In previous steps, we have typically offered derivation rules to translate between local and global semantics. These derivation rules need now to be implemented in view definitions. View creation proceeds as follows: create the integrated view model document, and then copy definitions of the data coming from base tables (com_customer, credit_person, credit_debt, GlobalCustomerDataReference)). Implement subtype view definitions first (Customer, Person), followed by supertypes (GlobalCustomer), additional fact types to these supertypes (RolePlaying), intersection types (DebtCustomer), additional types to intersection types (GlobalDebt), and, finally, generate DDL code.

The integrated model, created automatically from our integrated schema, does not contain information pertaining to derivation rules and data sources. The final integrated view model document, therefore, is created separately from this integrated model. Although the result looks similar to the integrated model, it is much more complex, due to the additional derivation rules.

The DDL code contains the complete definition of our integrated model in terms of SQL code. The code can be executed on our federated system, and the bulk of our federation is completed.

After having followed through the whole procedure as described above, we obtain a final integrated view model (cf. Figure 9). This model differs slightly from the relational model of Figure 8; the difference can be found in the view definitions, displayed in our schema with a slash in front of the table names. From this integrated view model we can –mostly manually- construct the extra desired DDL code.

6 Conclusion

The framework presented in this paper offers a procedure to arrive at implementations of consistent and complete federations. The chosen ORM modeling framework

allowed not only for precise modeling of a data federation, but also hosts tool support. We have demonstrated how ORM models, together with reverse engineering techniques, can be used in combination with an actual, industrial-strength implementation platform to develop semantically consistent and high performance database federations.

References

1. Balsters, H., Halpin, T.: Formal Semantics of Dynamic Rules in ORM. In: Meersman, R., Tari, Z., Herrero, P. (eds.) OTM-WS 2008. LNCS, vol. 5333, pp. 699–708. Springer, Heidelberg (2008)
2. Balsters, H., Halpin, T.: Data Integration with ORM. In: Meersman, R., Tari, Z., Herrero, P. (eds.) OTM-WS 2007, Part I. LNCS, vol. 4805, Springer, Heidelberg (2007)
3. Balsters, H., de Brock, E.O.: Integration of Integrity Constraints in Federated Schemata Based on Tight Constraining. In: Meersman, R., Tari, Z. (eds.) OTM 2004. LNCS, vol. 3290, pp. 748–767. Springer, Heidelberg (2004)
4. Bakema, G., Zwart, J., van der Lek, H.: Fully Communication Oriented Information Modelling. Ten Hagen Stam, The Netherlands (2000)
5. Betawadkar-Norwood, A., Lin, E., Ursu, I.: Using data federation technology in IBM WebSphere Information Integrator. IBM developerWorks (June 23, 2005), http://www.ibm.com/developerworks
6. Calì, A., Calvanese, D., De Giacomo, G., Lenzerini, M.: Data integration under integrity constraints. In: Pidduck, A.B., Mylopoulos, J., Woo, C.C., Ozsu, M.T. (eds.) CAiSE 2002. LNCS, vol. 2348, p. 262. Springer, Heidelberg (2002)
7. Chen, P.P.: The entity-relationship model—towards a unified view of data. ACM Transactions on Database Systems 1(1), 9–36 (1976)
8. Curland, M., Halpin, T.: Model Driven Development with NORMA. In: Proc. 40th Int. Conf. on System Sciences (HICSS 40). IEEE Computer Society Press, Los Alamitos (2007)
9. Embley, D.W.: Object Database Development. Addison-Wesley, Reading (1997)
10. Embley, D.W., Xiu, L.: A composite approach to automating direct and indirect schema mappings. Inf. Syst. 31(8), 697–732 (2006)
11. Halpin, T.: A Logical Analysis of Information Systems: static aspects of the data-oriented perspective, doctoral dissertation, University of Queensland (1989), http://www.orm.net/Halpin_PhDthesis.pdf
12. Halpin, T.A., Evans, K., Hallock, P.: Database Modeling with Microsoft® Visio for Enterprise. Morgan Kaufmann, San Francisco (2003)
13. Halpin, T.: ORM 2. In: Meersman, R., Tari, Z., Herrero, P. (eds.) OTM-WS 2005. LNCS, vol. 3762, pp. 676–687. Springer, Heidelberg (2005)
14. Halpin, T.: ORM/NIAM Object-Role Modeling. In: Bernus, K., Mertins, K., Schmidt, G. (eds.) Handbook on Information Systems Architectures, 2nd edn., pp. 81–103. Springer, Heidelberg (2006)
15. Halpin, T., Morgan, T.: Information Modeling and Relational Databases, 2nd edn. Morgan Kaufmann, San Francisco (2008)
16. Lenzerini, M.: Data integration: a theoretical perspective. In: ACM PODS 2002. ACM Press, New York (2002)

17. Object Management Group, UML 2.0 Superstructure Specification (2003), http://www.omg.org/uml
18. Ullman, D.: Information Integration Using Logical Views. In: Afrati, F.N., Kolaitis, P.G. (eds.) ICDT 1997. LNCS, vol. 1186, pp. 19–40. Springer, Heidelberg (1996)
19. Wikipedia authors (n.d.), Laplace transform